

Chicago Crime Data Analysis Using Big Data

INFO-I535

Sumbad, Sharanbasav
INDIANA UNIVERSITY ssumbad@iu.edu

Index:

Contents	Page No.
Introduction	2
Background	2
Methodology	5
Results	10
Discussions	19
Conclusions	22
References	23

Introduction:

We will be Analyzing the Chicago Crime Data and shedding light on some useful insights while analyzing the data. We treat this data a Big Data as the crime reports from 2021 to present as of November 2021 so the number of observations is about 7,428,804.

Chicago crime data set from 2021 to the present, Using Big Query API, the data is extracted from the City of Chicago public website: which would be in JSON/CSV , the analysis is done on virtual machine on Jetstream on the Pyspark-Ubuntu18_04-instance, using the Jupyter notebook a schema is provided for the dataset, analysis the data using the Date Time module in PySpark o the Spark data frame using the RDD functions, insights and statistics from the data will be provided, trend over the years and use pyspark.ml.feature to build a model and predict the what kind of crime will be committed from the given data variables and then push the files to GitHub. The pipeline will resemble several topics and technologies topics learnt in INFO-I535.

PySpark is an excellent language for doing large-scale exploratory data analysis, machine learning pipelines, and data platform ETLs. PySpark is an excellent language to use if we are already familiar with Python and libraries like Pandas. It'll help you construct more scalable analytics and pipelines

Big Query is ideal for swiftly storing and querying massive data sets. Google Cloud SQL is mostly built on RDBMS (Relational Database Management System) ideas. It supports MySQL and PostgreSQL databases. Although Big Query is best suited for analytics, it can also handle transactional data. Because Big Query is a Datawarehouse with the potential to query ridiculously enormous data sets and produce responses instantaneously, it is far quicker than Querying in Cloud SQL. A Big Query API is used to build client libraries, IDE plugins, and other tools that interact with Google APIs.

Here, Apache Spark is used to read, manipulate, and query data. We Make use of current Python packages to visualize the data. Matplotlib is used at for visualization, also use of Seaborn where Spark and Matplotlib interoperability is a problem, utilization of Pandas and NumPy is done.

Spark is a big data processing framework that processes data 100 times faster than Hadoop in memory.

Virtualization allows users to separate operating systems from the underlying hardware, allowing them to run different operating systems simultaneously on a single physical machine, such as Windows and Linux. Guests OSs are the name for such operating systems (operating systems).

Virtualization is a technique that uses software to create an abstraction layer over computer hardware, allowing physical components such as processors, memory, and storage to be split into numerous virtual pieces (also known as virtual machines). Based on Atmosphere and OpenStack, Jetstream is a user-friendly cloud computing platform for researchers. It is intended to provide adjustable cyberinfrastructure that allows you to have on-demand access to interactive computing and data analysis capabilities, anytime and wherever you need to examine your data.

Background:

Crime is an inseparable aspect of our existence in this planet. We hear about them every day, and some of us have been involved in at least one of them at some point in our lives. Being cautious and improving safety is no longer a simple command. To act more wisely against this problem, we need to leverage current technology and data science methodologies. The police department has many records and paperwork that have been accumulated through time and may be utilized as a useful source of data for data analytics jobs. Applying analytical tasks to these data yields useful knowledge that may be utilized to improve our society's safety and reduce crime.

This dataset reflects reported incidents of crime (except for murders where data exists for each victim) that occurred in the City of Chicago from 2001 to present, minus the most recent seven days. Data is extracted from the Chicago Police Department's CLEAR (Citizen Law Enforcement Analysis and Reporting) system.

With 749 murders in 2016, Chicago, the nation's third-largest city, accounted for 22% of the statewide rise, more than the combined number of murders in the largest city, New York (334), and the second-largest city, Los Angeles (294) for the same year. In 2016, the expected number of killings in Chicago climbed by 52%. The number of killings in the United States increased by 8.6%, making Chicago an anomaly and an intriguing case study.

Data:

Rows 7.44M	Columns 22	Each row is a Reported Crime
Columns in this Dataset		
Column Name	Description	Type
ID	Unique Identifier for the record.	Number #
Case Number	The Chicago Police Department RD Number (Records Divisi...	Plain Text T
Date	Date when the incident occurred, this is sometimes a best ...	Date & Time 日
Block	The partially redacted address where the incident occurred...	Plain Text T
IUCR	The Illinois Unifrom Crime Reporting code. This is directly il...	Plain Text T
Primary Type	The primary description of the IUCR code.	Plain Text T
Description	The secondary description of the IUCR code, a subcategory...	Plain Text T
Location Description	Description of the location where the incident occurred.	Plain Text T
Arrest	Indicates whether an arrest was made.	Checkbox ✓
Domestic	Indicates whether the incident was domestic-related as de...	Checkbox ✓
Beat	Indicates the beat where the incident occurred. A beat is th...	Plain Text T
District	Indicates the police district where the incident occurred. Se...	Plain Text T
Ward	The ward (City Council district) where the incident occurred...	Number #
Community Area	Indicates the community area where the incident occurred....	Plain Text T
FBI Code	Indicates the crime classification as outlined in the FBI's Na...	Plain Text T
X Coordinate	The x coordinate of the location where the incident occur...	Number #
Y Coordinate	The y coordinate of the location where the incident occurre...	Number #
Year	Year the incident occurred.	Number #
Updated On	Date and time the record was last updated.	Date & Time 日
Latitude	The latitude of the location where the incident occurred. T...	Number #
Longitude	The longitude of the location where the incident occurred. ...	Number #
Location	The location where the incident occurred in a format that a...	Location 標

ID - Unique identifier for the record.

Case Number - The Chicago Police Department RD Number (Records Division Number), which is unique to the incident.

Date - Date when the incident occurred. this is sometimes a best estimate.

Block - The partially redacted address where the incident occurred, placing it on the same block as the actual address.

IUCR - The Illinois Uniform Crime Reporting code. This is directly linked to the Primary Type and **Description**. See the list of IUCR codes at <https://data.cityofchicago.org/d/c7ck-438e>.

Primary Type - The primary description of the IUCR code.

Description - The secondary description of the IUCR code, a subcategory of the primary description.

Location Description - Description of the location where the incident occurred.

Arrest - Indicates whether an arrest was made.

Domestic - Indicates whether the incident was domestic-related as defined by the Illinois Domestic Violence Act.

Beat - Indicates the beat where the incident occurred. A beat is the smallest police geographic area – each beat has dedicated police beat car. Three to five beats make up a police sector, and three sectors make up a police **district**. The Chicago Police Department has 22 police districts. See the beats at <https://data.cityofchicago.org/d/aerh-rz74>.

District - Indicates the police district where the incident occurred. See the districts at <https://data.cityofchicago.org/d/fthy-xz3r>.

Ward - The ward (City Council district) where the incident occurred. See the wards at <https://data.cityofchicago.org/d/sp34-6z76>.

Community Area - Indicates the community area where the incident occurred. Chicago has 77 community areas. See the community areas at <https://data.cityofchicago.org/d/cauq-8yn6>.

FBI Code - Indicates the crime classification as outlined in the FBI's National Incident-Based Reporting System (NIBRS). See the Chicago Police Department listing of these classifications at http://gis.chicagopolice.org/clearmap_crime_sums/crime_types.html.

X Coordinate - The x coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection. This location is shifted from the actual location for partial redaction but falls on the same block.

Y Coordinate - The y coordinate of the location where the incident occurred in State Plane Illinois East NAD 1983 projection. This location is shifted from the actual location for partial redaction but falls on the same block.

Year - Year the incident occurred.

Updated On - Date and time the record was last updated.

Latitude - The latitude of the location where the incident occurred. This location is shifted from the actual location for partial redaction but falls on the same block.

Longitude - The longitude of the location where the incident occurred. This location is shifted from the actual location for partial redaction but falls on the same block.

Location - The location where the incident occurred in a format that allows for creation of maps and other geographic operations on this data portal. This location is shifted from the actual location for partial redaction but falls on the same block.

Methodology:

Data Ingestion:

Any procedure that moves data from one area to another so that it may be picked up for additional processing or analysis is referred to as "data ingestion." Here we use Batch ingestion to get the data from the Source, The Chicago crime data is ingested from the public website of city of Chicago using the Big Query API.

The serverless design of Big Query allows you to utilize SQL queries to solve the most pressing problems while requiring no infrastructure administration. The scalable, distributed analytical engine in Big Query allows you to query terabytes of data in seconds and petabytes of data in minutes. Big Query maximizes flexibility by separating the compute engine that analyzes the data from the storage choices. we can store and analyze the data within Big Query or use Big Query to assess your data where it lives. For this project big query is used only to access the data and store it, the data at the source is in a different format but we recover it as a JSON and the convert it to a CSV file. Big Query Helper is a helper class that makes typical Big Query read-only jobs easier. It simplifies query execution and serves as a useful steppingstone toward utilizing the main Big Query python API.

Query of Chicago Crime Data Set

Draft saved

Share

Save Version 0

IK

File Edit View Run Add-ons Help

+ Run All Code

Draft Session (22m)

DD C P S B H D U V A M

```
[1]:
import bq_helper
from bq_helper import BigQueryHelper
chicago_crime = bq_helper.BigQueryHelper(active_project="bigquery-public-data",
                                          dataset_name="chicago_crime")

bq_assistant = BigQueryHelper("bigquery-public-data", "chicago_crime")
bq_assistant.list_tables()

Using Kaggle's public dataset BigQuery integration.
Using Kaggle's public dataset BigQuery integration.
[1]: ['crime', 'snapshot-of-crime']
```

Query of Chicago Crime Data Set

Draft saved

Share

Save Version 0

IK

File Edit View Run Add-ons Help

+ Run All Code

Draft Session (23m)

DD C P S B H D U V A M

```
[2]:
from google.cloud import bigquery

[3]:
#bq_assistant.head('crime')
fetch_query = """SELECT * FROM
'bigquery-public-data.chicago_crime.crime' WHERE year between (2001) and (2021)
"""

query_data = chicago_crime.query_to_pandas_safe(fetch_query)
query_data.head()
```

[3]:	unique_key	case_number	date	block	iucr	primary_type	description	location_description	arrest	domestic	...	ward	community_area	fbi_code	x_coordinate	y
0	2820510	HJ478890	2003-07-05 11:55:00+00:00	081XX S KIMBARK AVE	1365	CRIMINAL TRESPASS	TO RESIDENCE	RESIDENCE	False	False	...	8	45	26	1186294.0	
1	2829110	HJ479159	2003-07-07 07:58:28+00:00	023XX S ARCHER AVE	0470	PUBLIC PEACE VIOLATION	RECKLESS CONDUCT	STREET	True	False	...	25	34	24	1173372.0	
2	2714903	HJ304126	2003-04-16 11:14:00+00:00	050XX S PRINCETON AVE	2023	NARCOTICS	POSS: HEROIN(BRN/TAN)	STREET	True	False	...	3	37	18	1175137.0	
3	2720621	HJ350470	2003-05-08 06:00:00+00:00	111XX S AVENUE M	502P	OTHER OFFENSE	FALSE/STOLEN/ALTERED TRP	STREET	True	False	...	10	52	26	1201553.0	
4	2729844	HJ363645	2003-05-14 06:35:00+00:00	054XX W 64TH ST	0313	ROBBERY	ARMED: OTHER DANGEROUS WEAPON	SIDEWALK	False	False	...	13	64	03	1141146.0	

5 rows x 22 columns

Query of Chicago Crime Data Set Draft saved

File Edit View Run Add-ons Help

+ [Icons] Run All Code

Draft Session (23m) [Icons]

```
[2]: from google.cloud import bigquery
```

```
[3]: #bq_assistant.head('crime')
fetch_query = """SELECT * FROM
'bigquery-public-data.chicago_crime.crime' WHERE year between (2001) and (2021)
"""
query_data = chicago_crime.query_to_pandas_safe(fetch_query)
query_data.head()
```

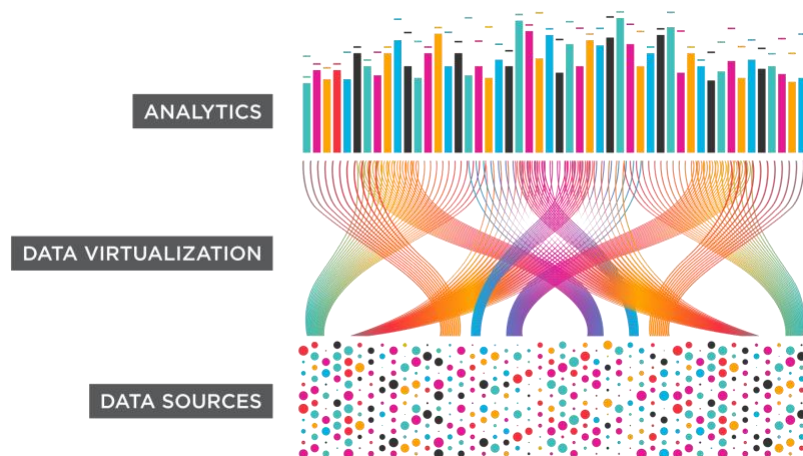
	unique_key	case_number	date	block	lucr	primary_type	description	location_description	arrest	domestic	...	ward	community_area	fbi_code	x_coordinate	y
0	2820510	HJ478890	2003-07-05 11:55:00+00:00	081XX S KIMBARK AVE	1365	CRIMINAL TRESPASS	TO RESIDENCE	RESIDENCE	False	False	...	8	45	26	1186294.0	
1	2829110	HJ479159	2003-07-07 07:58:28+00:00	023XX S ARCHER AVE	0470	PUBLIC PEACE VIOLATION	RECKLESS CONDUCT	STREET	True	False	...	25	34	24	1173372.0	
2	2714903	HJ304126	2003-04-16 11:14:00+00:00	050XX S PRINCETON AVE	2023	NARCOTICS	POSS: HEROIN(BRN/TAN)	STREET	True	False	...	3	37	18	1175137.0	
3	2720621	HJ350470	2003-05-08 06:00:00+00:00	111XX S AVENUE M	502P	OTHER OFFENSE	FALSE/STOLEN/ALTERED TRP	STREET	True	False	...	10	52	26	1201553.0	
4	2729844	HJ363645	2003-05-14 06:35:00+00:00	054XX W 64TH ST	0313	ROBBERY	ARMED: OTHER DANGEROUS WEAPON	SIDEWALK	False	False	...	13	64	03	1141146.0	

5 rows x 22 columns

Virtualization:


Virtualization is a technique for running virtualized versions of computer systems in a layer separate from the hardware. Virtualization allows you to run several apps and operating systems on a single computer by creating virtual versions of IT services that were previously connected to hardware. There are a few advantages that might be game changers in terms of achieving corporate efficiency and cost reductions. For backup, disaster recovery, business continuity, test/development, replication, and archiving, data virtualization provides a central point of access and management. It lowers the IT infrastructure requirements while improving data delivery to end users by compressing and deduplicating data across all apps and storage systems. The administration of massive amounts of highly dispersed data repositories, as well as the deployment of compute- and data-intensive applications, are required to solve big data issues. Virtualization adds an extra degree of efficiency to big data systems, allowing them to become a reality. Even though virtualization is not legally required for big data analysis, software frameworks are more efficient in a virtualized environment.

Processor virtualization aids in processor optimization and performance. Memory virtualization allows memory to be separated from the servers. Repeated queries of massive data sets and the building of powerful analytic algorithms are all part of big data analysis, and they're all geared to hunt for patterns and trends that aren't yet recognized. These complex analytics may need a significant amount of computing power (CPU) and memory (RAM). Without appropriate CPU and memory resources, some of these computations can take a long time.



Virtual System configurations:

Resources > Pyspark-Ubuntu18_04-instance



Pyspark-Ubuntu18_04-instance

Allocation Source

TG-CIS200011

Allocation Used

25% of 750000 SUs from TG-CIS200011

Instance Details

Status	Active
Activity	N/A
Size	m1.large (CPU: 10, Mem: 30 GB, Disk: 60 GB)
IP Address	149.165.156.3 Copy
Launched	Nov 2, 2021 (14 days ago)
Based on	Pyspark-Ubuntu18_04-instance v1.0
Provider	Jetstream - Indiana University
ID	47633
Alias	2dc3ad1d-0391-4d8c-b3c4-e44ab9ac5c1e Copy

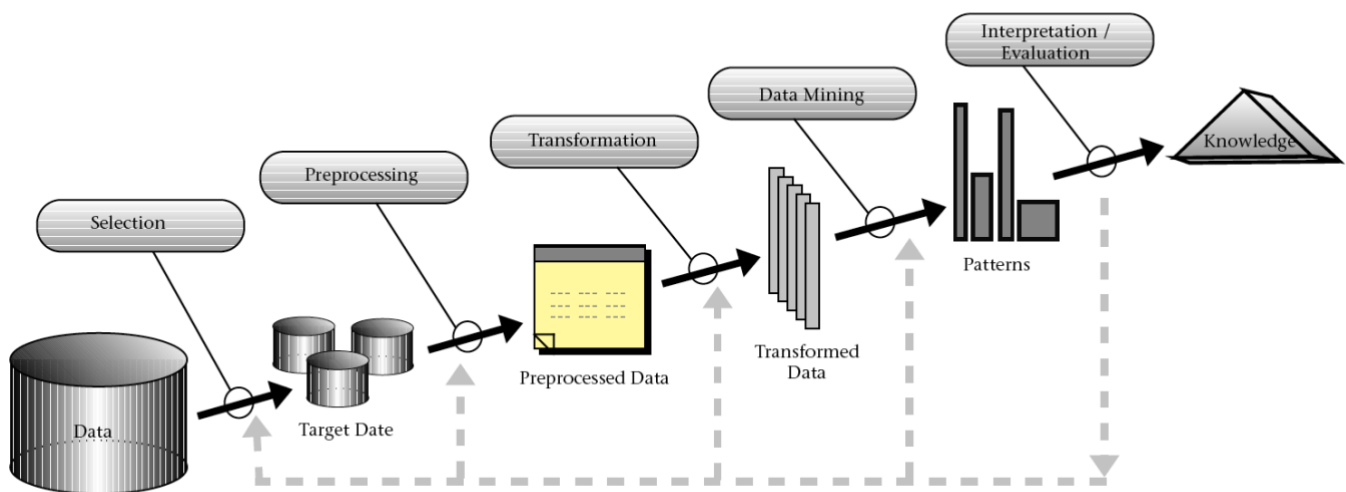
Actions

- Report
- Image
- Suspend
- Shelve
- Stop
- Reboot
- Redeploy
- Delete

Links

- Open Web Shell
- Open Web Desktop

Data Pipeline and Lifecycles:



1. Building up an understanding of the application domain

This is the first stage in the process. It sets the stage for figuring out what to do with various decisions like transformation, algorithms, and representation, among others. Individuals in charge of the project must comprehend and describe the end-goals user's as well as the environment in which the knowledge discovery process will take place. In Our project we needed to have a understanding of the criminal activities and bit of background in the Law and enforcement

2.Choosing and creating a data set on which discovery will be performed

The data that will be used for the knowledge discovery process should be determined once the objectives have been identified. The aspects that will be evaluated for the process include identifying what data is available, acquiring significant data, and then integrating all the data for knowledge discovery into one set. Data Mining learns and finds from the available data; hence this process is crucial. The more qualities analyzed, the more likely the study will be ineffective in this regard if certain essential attributes are absent. Organizing, collecting, and operating advanced data repositories, on the other hand, is costly, and there is a risk of failure. In our case the data is created by the City of Chicago Counsel and for our analysis we extract the data from them.

3.Preprocessing and cleansing

The data that will be used for the knowledge discovery process should be determined once the objectives have been identified. The aspects that will be evaluated for the process include identifying what data is available, acquiring significant data, and then integrating the data for knowledge discovery into one set. Data Mining learns and finds from the available data; hence this process is crucial. The more qualities analyzed, the more likely the study will be ineffective in this regard if certain essential attributes are absent. Organizing, collecting, and operating advanced data repositories, on the other hand, is costly, and there is a risk of failure. We perform preprocessing and conversion of the data according to our needs.

4.Data Transformation

This stage involves preparing and developing relevant data for Data Mining. Dimension reduction (for example, feature selection and extraction, and record sampling) as well as attribute modification are used here (for example, discretization of numerical attributes and functional transformation). This stage is often project-specific and can be critical to the success of the overall project.

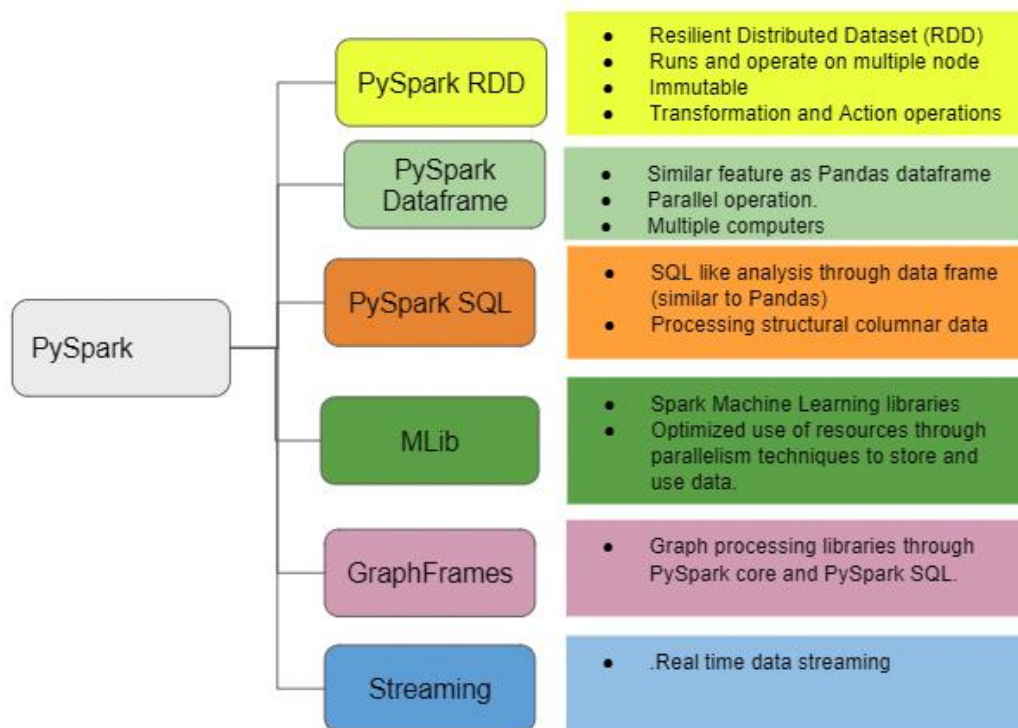
5.Data Evaluation

In this phase, we evaluate and analyze the mined patterns, rules, and dependability in relation to the first-step goal. In this section, we look at the pretreatment procedures and how they affect the method outcomes. Include a feature in step 4, for example, and then repeat the process. This stage focuses on the induced data's readability and utility. The recognized knowledge is also documented in this phase for future use. The utilization, as well as general feedback and discovery outcomes obtained by project, is the last phase. We visualize data in the form of table or graphs and charts.

6.Using the discovered knowledge

We are now ready to incorporate the information into a different framework for future action. Knowledge becomes useful when it allows us to make changes to the system and track the results. The efficiency of the process is determined by how well this stage is completed. This phase presents several obstacles, including the loss of the "laboratory settings" in which we previously operated. Certain numbers may become unavailable as a result of changes to data structures, and the data domain may be altered, such as an attribute with a value that was not expected earlier.

Distributed Computing with PySpark:



When we talk about data frames, we immediately think of Pandas. Pandas and Pyspark data frame vary in that Pandas stores all data in the memory of the machine on which it is run, but Pyspark data frame works with several computers in a cluster (distributed computing) and spreads data processing to those systems' memories. The parallel analysis of a large dataset on several computers is Pyspark's most significant feature. This is the fundamental reason: Pyspark excels at enormous datasets distributed across multiple computers, whereas Pandas excels at datasets that can be stored on a single computer. The distinction between Pandas and Pyspark data frames is not limited to this. Between Pandas and Pyspark, there are some not-so-subtle changes in how the same operations are carried out.

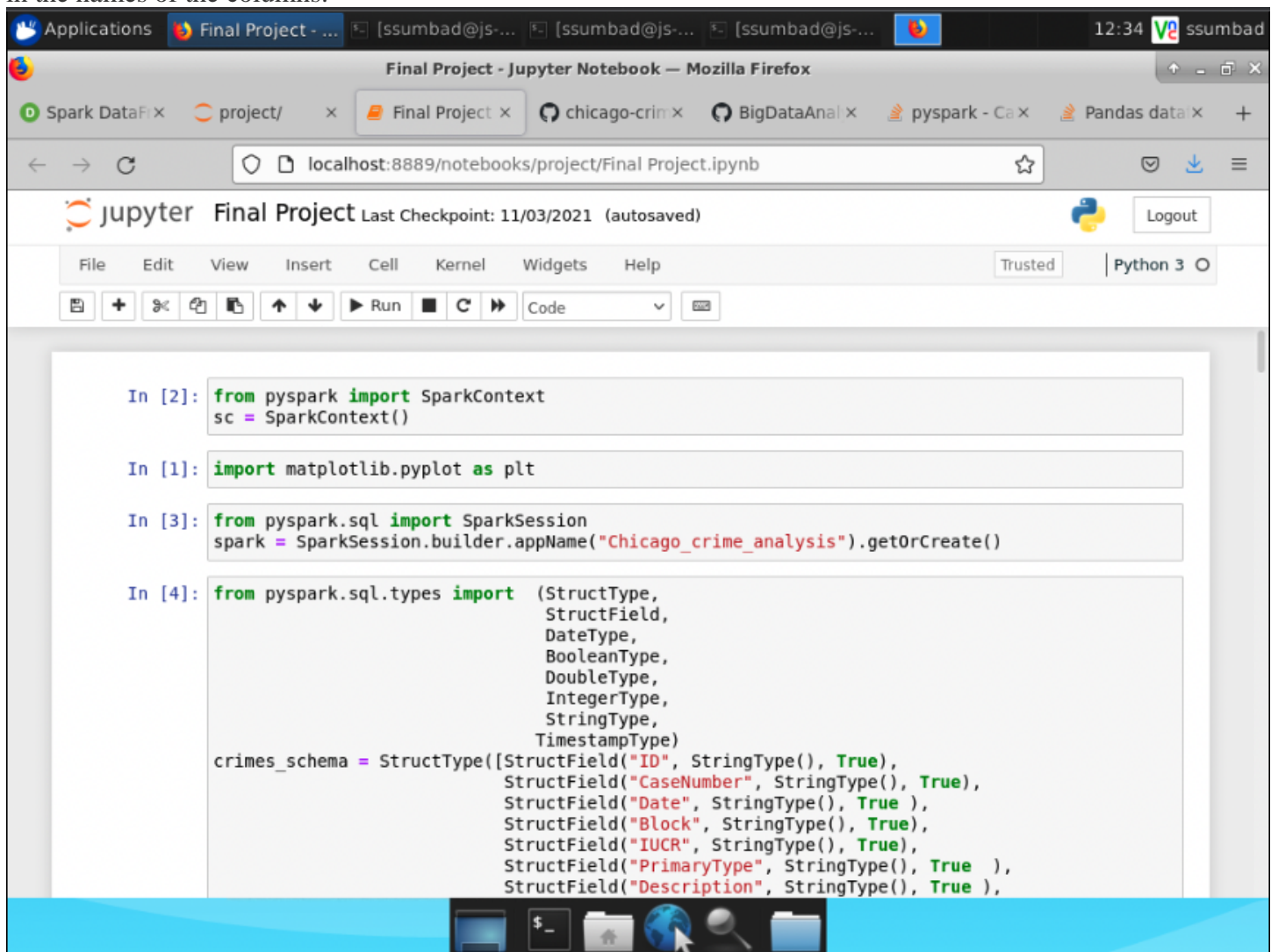
Spark Data Frame	Pandas Data Frame
Spark DataFrame supports parallelization.	Pandas DataFrame does not support parallelization.
Spark DataFrame has Multiple Nodes.	Pandas DataFrame has a Single Node.
It follows Lazy Execution which means that a task is not executed until an action is performed.	It follows Eager Execution, which means task is executed immediately.
Spark DataFrame is Immutable.	Pandas DataFrame is Mutable.
Complex operations are difficult to perform as compared to Pandas DataFrame.	Complex operations are easier to perform as compared to Spark DataFrame.
park DataFrame is distributed and hence processing in the Spark DataFrame is faster for a large amount of data.	Pandas DataFrame is not distributed and hence processing in the Pandas DataFrame will be slower for a large amount of data.
sparkDataFrame.count() returns the number of rows.	pandasDataFrame.count() returns the number of non-NA/null observations for each column.
Spark DataFrames are excellent for building a scalable application	Pandas DataFrames can't be used to build a scalable application.
DataFrames can't be used to build a scalable application.	Pandas DataFrame does not assure fault tolerance. We need to implement our own framework to assure it

When working with many datasets, pandas can be slow, but spark provides an integrated API for working with data that makes it faster than pandas. Spark has a simpler API than Pandas, making it easier to use. In Spark, ANSI SQL compatibility is supported for Python, Scala, Java, and R. Spark performs computations in-memory (RAM). SQL in Pyspark Structured data stored in Hadoop and Spark clusters can be queried using Hive and Spark SQL. Spark SQL is at least 10 times faster than Hive in most circumstances. When Spark SQL is used in conjunction with another programming interface, the result is returned as a data frame.

Results:

A Spark Session can be used create DataFrame, register DataFrame as tables, execute SQL over tables, cache tables, and read parquet files. It is the entry point to programming Spark with the DataFrame API. We can create a Spark Session, us following builder pattern

We can let Spark infer the schema of our csv data but proving pre-defined schema makes the reading process faster. Further, it helps us to make the column names to have the format we want, for example, to avoid spaces in the names of the columns.



```
In [2]: from pyspark import SparkContext
sc = SparkContext()

In [1]: import matplotlib.pyplot as plt

In [3]: from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Chicago_crime_analysis").getOrCreate()

In [4]: from pyspark.sql.types import (StructType,
                                         StructField,
                                         DateType,
                                         BooleanType,
                                         DoubleType,
                                         IntegerType,
                                         StringType,
                                         TimestampType)
crimes_schema = StructType([StructField("ID", StringType(), True),
                              StructField("CaseNumber", StringType(), True),
                              StructField("Date", StringType(), True),
                              StructField("Block", StringType(), True),
                              StructField("IUCR", StringType(), True),
                              StructField("PrimaryType", StringType(), True),
                              StructField("Description", StringType(), True),
```

We can see the scheme of the crime spark data frame

```
In [17]: crimes.printSchema()

root
|-- ID: string (nullable = true)
|-- CaseNumber: string (nullable = true)
|-- Date: string (nullable = true)
|-- Block: string (nullable = true)
|-- IUCR: string (nullable = true)
|-- PrimaryType: string (nullable = true)
|-- Description: string (nullable = true)
|-- LocationDescription: string (nullable = true)
|-- Arrest: boolean (nullable = true)
|-- Domestic: boolean (nullable = true)
|-- Beat: string (nullable = true)
|-- District: string (nullable = true)
|-- Ward: string (nullable = true)
|-- CommunityArea: string (nullable = true)
|-- FBICode: string (nullable = true)
|-- XCoordinate: double (nullable = true)
|-- YCoordinate: double (nullable = true)
|-- Year: integer (nullable = true)
|-- UpdatedOn: date (nullable = true)
|-- Latitude: double (nullable = true)
|-- Longitude: double (nullable = true)
|-- Location: string (nullable = true)
```

The Date column is formatted as a string. Using user-defined functions, we convert it to timestamp format (udf). With withColumn, we may create a new column, and with drop, we can delete one or more columns.:

```
In [19]: from datetime import datetime
from pyspark.sql.functions import col, udf
myfunc = udf(lambda x: datetime.strptime(x, '%m/%d/%Y %I:%M:%S %p'), TimestampType())
df = crimes.withColumn('Date_time', myfunc(col('Date'))).drop('Date')
df.select(df["Date_time"]).show(5)

+-----+
|      Date_time|
+-----+
|2015-09-05 13:30:00|
|2015-09-04 11:30:00|
|2018-09-01 00:01:00|
|2015-09-05 12:45:00|
|2015-09-05 13:00:00|
+-----+
only showing top 5 rows
```

Using describe, we can determine the statistics of string and numeric columns. We must supply the column names as a python list when selecting several columns.

```
In [20]: crimes.select(["Latitude", "Longitude", "Year", "XCoordinate", "YCoordinate"]).describe().show()

+-----+-----+-----+-----+-----+
|summary|Latitude|Longitude|Year|XCoordinate|
+-----+-----+-----+-----+-----+
| count|7354165|7354165|7428804|7354165| |
| mean|41.842043502881815|-87.67162901299518|2009.3644472246137|1164564.5787497289|1885731.510358688|
| stddev|0.0888102297216899|0.061101090227819926|5.803935915450128|16853.734354934866|32280.050624566422|
| min|36.619446395|-91.686565684|2001|0.0|
| max|42.022910333|-87.524529378|2021|1205119.0|
+-----+-----+-----+-----+-----+
1951622.0|
```

The figures above are unappealing. Let's round them up with PySpark's format number function.

```
In [21]: from pyspark.sql.functions import format_number
result = crimes.select(["Latitude", "Longitude", "Year", "XCoordinate", "YCoordinate"]).describe(
result.select(result['summary'],
format_number(result['Latitude'].cast('float'),2).alias('Latitude'),
format_number(result['Longitude'].cast('float'),2).alias('Longitude'),
result['Year'].cast('int').alias('year'),
format_number(result['XCoordinate'].cast('float'),2).alias('XCoordinate'),
format_number(result['YCoordinate'].cast('float'),2).alias('YCoordinate')
).show()
```

summary	Latitude	Longitude	year	XCoordinate	YCoordinate
count	7,354,165.00	7,354,165.00	7428804	7,354,165.00	7,354,165.00
mean	41.84	-87.67	2009	1,164,564.62	1,885,731.50
stddev	0.09	0.06	5	16,853.73	32,280.05
min	36.62	-91.69	2001	0.00	0.00
max	42.02	-87.52	2021	1,205,119.00	1,951,622.00

Check What is the total number of homicides in the dataset? Check the number of domestic assaults. Filtering can be done with or without the use of a filter.

```
In [24]: crimes.where(crimes["PrimaryType"] == "HOMICIDE").count()
```

```
Out[24]: 11514
```

```
In [25]: crimes.filter((crimes["PrimaryType"] == "ASSAULT") & (crimes["Domestic"] == "True")).count()
```

```
Out[25]: 107651
```

```
In [26]: columns = ['PrimaryType', 'Description', 'Arrest', 'Domestic']
crimes.where((crimes["PrimaryType"] == "HOMICIDE") & (crimes["Arrest"] == "true"))\
.select(columns).show(10)
```

PrimaryType	Description	Arrest	Domestic
HOMICIDE	FIRST DEGREE MURDER	true	false
HOMICIDE	FIRST DEGREE MURDER	true	false
HOMICIDE	FIRST DEGREE MURDER	true	false
HOMICIDE	FIRST DEGREE MURDER	true	false
HOMICIDE	FIRST DEGREE MURDER	true	false
HOMICIDE	FIRST DEGREE MURDER	true	false
HOMICIDE	FIRST DEGREE MURDER	true	false
HOMICIDE	RECKLESS HOMICIDE	true	false
HOMICIDE	FIRST DEGREE MURDER	true	false

To calculate various statistics, we use PySpark's utilities and calculate the average latitude. Pyspark utilities may also be used to determine maximum and minimum values. Pyspark utilities may also be used to determine maximum and minimum values. Calculate the number of offenses committed per year.

```
In [29]: lat_max = crimes.agg({"Latitude" : "max"}).collect()[0][0]
print("The maximum latitude values is {}".format(lat_max))
```

The maximum latitude values is 42.022910333

```
In [30]: df = crimes.withColumn("difference_from_max_lat", lat_max - crimes["Latitude"])
df.select(["Latitude", "difference_from_max_lat"]).show(5)
```

```
+-----+-----+
| Latitude|difference_from_max_lat|
+-----+-----+
|41.815117282| 0.20779305099999959|
|41.895080471| 0.127829861999999868|
| null| null|
|41.937405765| 0.08550456799999975|
|41.881903443| 0.14100688999999988|
+-----+-----+
only showing top 5 rows
```

```
In [31]: df = crimes.withColumnRenamed("Latitude", "Lat")
columns = ['PrimaryType', 'Description', 'Arrest', 'Domestic', 'Lat']
```

```
In [32]: df = crimes.withColumn("difference_from_max_lat", lat_max - crimes["Latitude"])
df.select(["Latitude", "difference_from_max_lat"]).show(5)
```

```
+-----+-----+
| Latitude|difference_from_max_lat|
+-----+-----+
|41.815117282| 0.20779305099999959|
|41.895080471| 0.127829861999999868|
| null| null|
|41.937405765| 0.08550456799999975|
|41.881903443| 0.14100688999999988|
+-----+-----+
```

```
In [33]: df = crimes.withColumnRenamed("Latitude", "Lat")
columns = ['PrimaryType', 'Description', 'Arrest', 'Domestic', 'Lat']
```

```
In [34]: from pyspark.sql.functions import mean
df.select(mean("Lat")).alias("Mean Latitude").show()
```

```
+-----+
| avg(Lat)|
+-----+
|41.842043502881815|
+-----+
```

```
In [35]: df.agg({"Lat": "avg"}).show()
```

```
+-----+
| avg(Lat)|
+-----+
|41.842043502881815|
+-----+
```

```
In [36]: from pyspark.sql.functions import max,min
df.select(max("Xcoordinate"),min("Xcoordinate")).show()
```

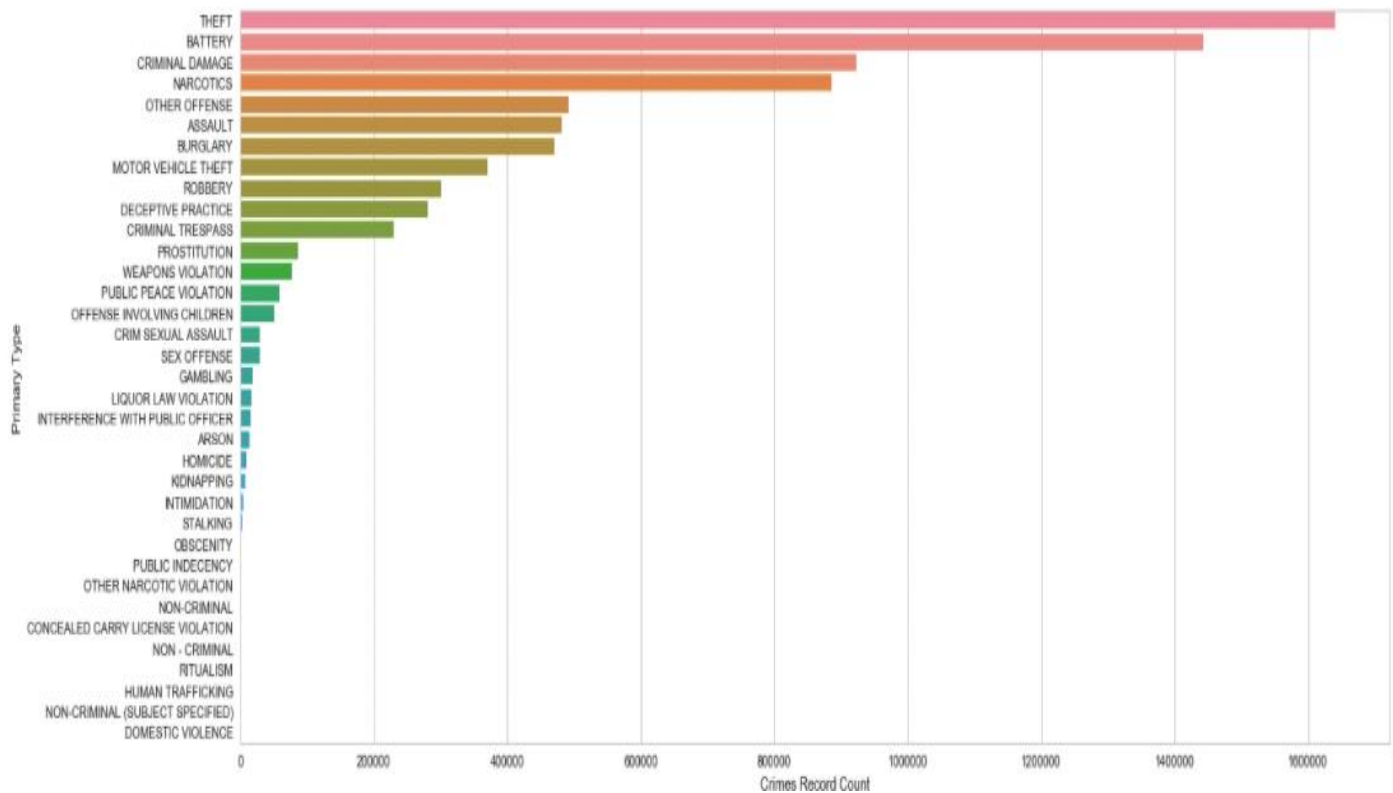
```
+-----+-----+
|max(Xcoordinate)|min(Xcoordinate)|
+-----+-----+
| 1205119.0| 0.0|
+-----+-----+
```

```
In [37]: df.filter(df["Domestic"]==True).count()/df.count() * 100
```

```
Out[37]: 13.63060594949066
```

```
In [38]: from pyspark.sql.functions import corr
df.select(corr("Lat", "Ycoordinate")).show()
```

```
+-----+
|corr(Lat, Ycoordinate)|
+-----+
|      0.9999937661869008|
+-----+
```



Using group by function to check the number of crimes year wise

```
In [39]: df.groupBy("Year").count().show()
df.groupBy("Year").count().collect()
```

```
+-----+-----+
|Year| count|
+-----+-----+
|2003|475969|
|2007|437064|
|2018|268483|
|2015|264590|
|2006|448151|
|2013|307384|
|2014|275655|
|2019|260835|
|2004|469408|
|2020|211375|
|2012|336203|
|2009|392794|
|2016|269633|
|2001|485818|
|2005|453755|
|2010|370444|
|2011|351923|
|2008|427133|
|2017|268866|
|2002|486776|
+-----+-----+
only showing top 20 rows
```

We can also view the overall number of offenses each year using matplotlib and Pandas. Plotting the Number of crime year wise in Pyspark

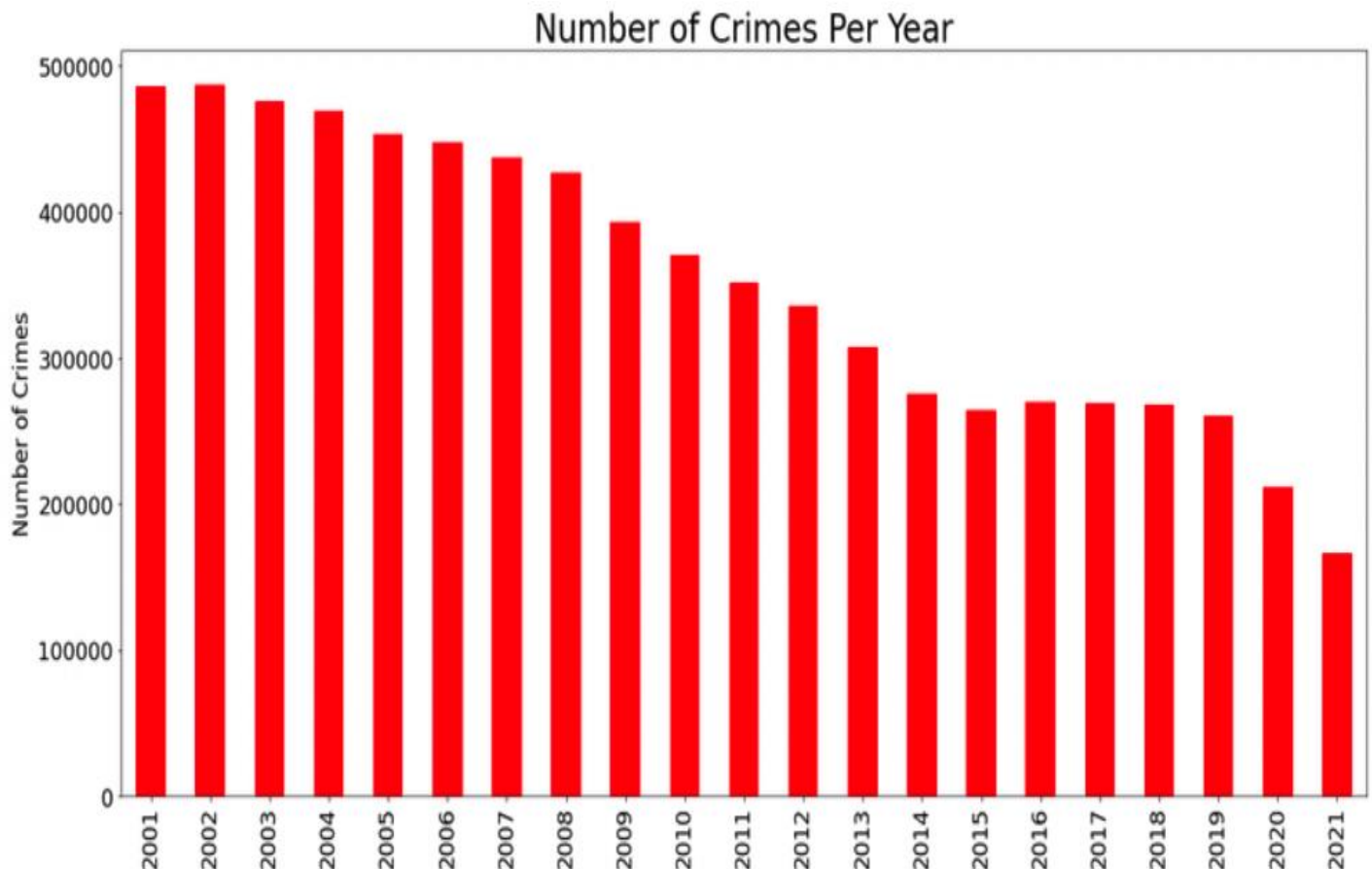
```
In [40]: count = [item[1] for item in df.groupBy("Year").count().collect()]
year = [item[0] for item in df.groupBy("Year").count().collect()]
number_of_crimes_per_year = {"count":count, "year" : year}
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
number_of_crimes_per_year = pd.DataFrame(number_of_crimes_per_year)
number_of_crimes_per_year.head()
```

```
Out[40]:
```

	count	year
0	475969	2003
1	437064	2007
2	268483	2018
3	264590	2015
4	448151	2006

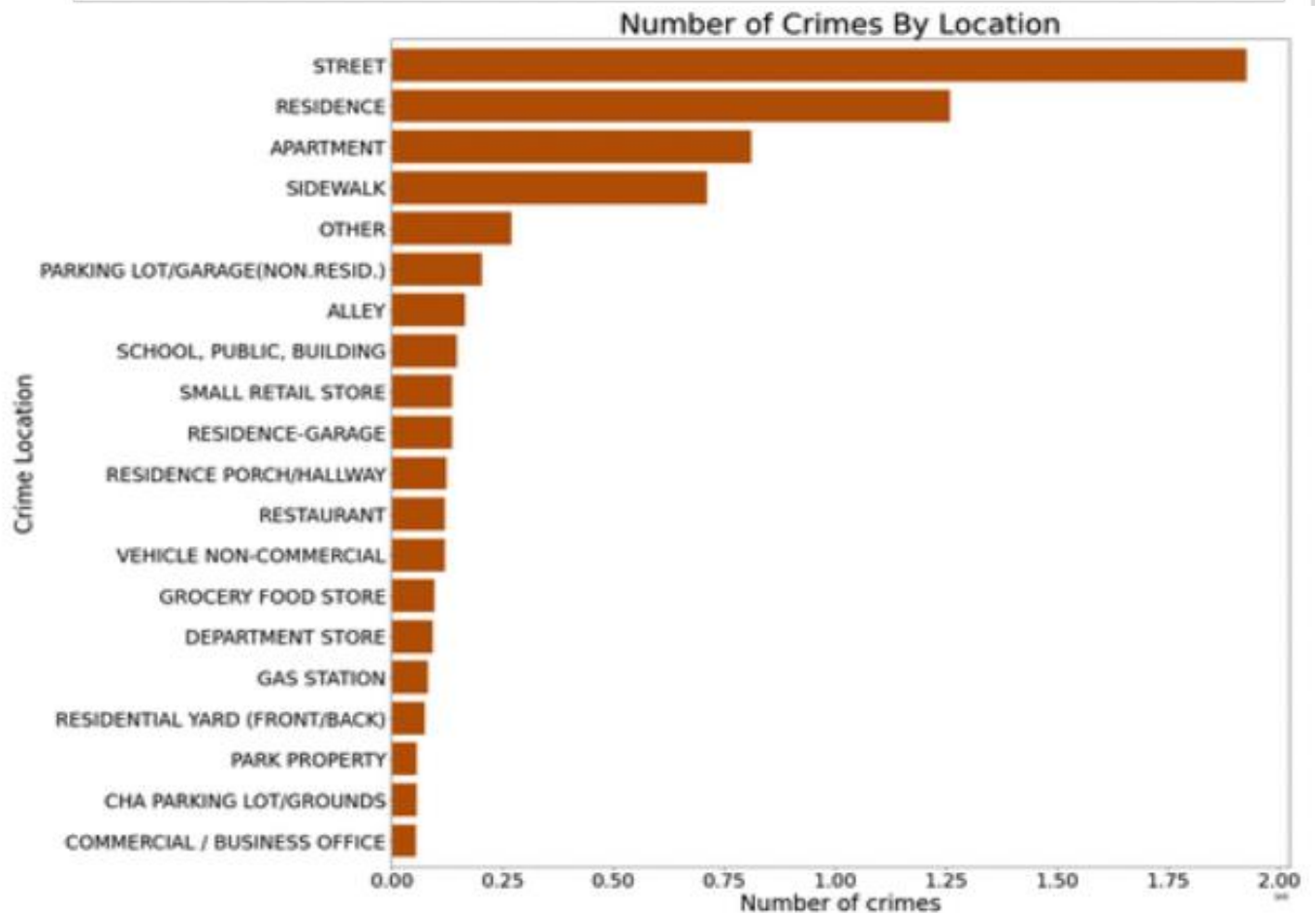
```
In [42]: number_of_crimes_per_year = number_of_crimes_per_year.sort_values(by = "year")
number_of_crimes_per_year.plot(figsize = (20,10), kind = "bar", color = "red",
                                x = "year", y = "count", legend = False)

plt.xlabel("", fontsize = 18)
plt.ylabel("Number of Crimes", fontsize = 18)
plt.title("Number of Crimes Per Year", fontsize = 28)
plt.xticks(size = 18)
plt.yticks(size = 18)
plt.show()
```



Where do most crimes take place? This can be checked by doing the following.

```
In [45]: crime_location = crimes.groupBy("LocationDescription").count().collect()
location = [item[0] for item in crime_location]
count = [item[1] for item in crime_location]
crime_location = {"location": location, "count": count}
crime_location = pd.DataFrame(crime_location)
crime_location = crime_location.sort_values(by = "count", ascending = False)
crime_location = crime_location.iloc[:20]
myplot = crime_location.plot(figsize = (20,20), kind = "barh", color = "#b35900", width = 0.8,
                             x = "location", y = "count", legend = False)
myplot.invert_yaxis()
plt.xlabel("Number of crimes", fontsize = 28)
plt.ylabel("Crime Location", fontsize = 28)
plt.title("Number of Crimes By Location", fontsize = 36)
plt.xticks(size = 24)
plt.yticks(size = 24)
plt.show()
```



Recorded Date

```
In [19]: import datetime
from pyspark.sql.functions import *
```

```
In [416]: df.select(min('date').alias('first_record_date'), max('date').alias('latest_record_date')).show(truncate=False)
```

```
+-----+-----+
|first_record_date|latest_record_date|
+-----+-----+
|01/01/2001 01:00:00 AM|12/31/2016 12:56:00 AM|
+-----+-----+
```

So it seems that the dataset we're dealing with comprises records from 2001-01-01 to 2016-12-31

Converting dates to a timestamp type. As seen in the schema output above, the date field is of string type, which won't be very helpful the format specifier that seems valid for date like '02/23/2006 09:06:22 PM' is '**MM/dd/yyyy hh:mm:ss a**'

```
In [420]: df = df.withColumn('date_time', to_timestamp('date', 'MM/dd/yyyy hh:mm:ss a'))\
        .withColumn('month', trunc('date_time', 'YYYY')) #adding a month column to be able to view stats on a monthly basis
```

```
In [421]: df.select(['date', 'date_time', 'month'])\
        .show(n=2, truncate=False)
```

```
+-----+-----+-----+
|date      |date_time      |month      |
+-----+-----+-----+
|10/07/2008 12:39:00 PM|2008-10-07 12:39:00|2008-01-01|
|10/09/2008 03:30:00 AM|2008-10-09 03:30:00|2008-01-01|
+-----+-----+-----+
only showing top 2 rows
```

```
In [80]: # A pandas data frame of the collected dictionary version of the date-grouped DF above
type_arrest_pddf = pd.DataFrame(type_arrest_date.rdd.map(lambda l: l.asDict()).collect())
```

```
In [72]: type_arrest_pddf['yearpd'] = type_arrest_pddf['month'].apply(lambda dt: datetime.datetime.strptime(pd.Timestamp(dt), '%Y'))
```

```
In [73]: type_arrest_pddf['arrest'] = type_arrest_pddf['arrest'].apply(lambda l: l=="True")
type_arrest_pddf.head(5)
```

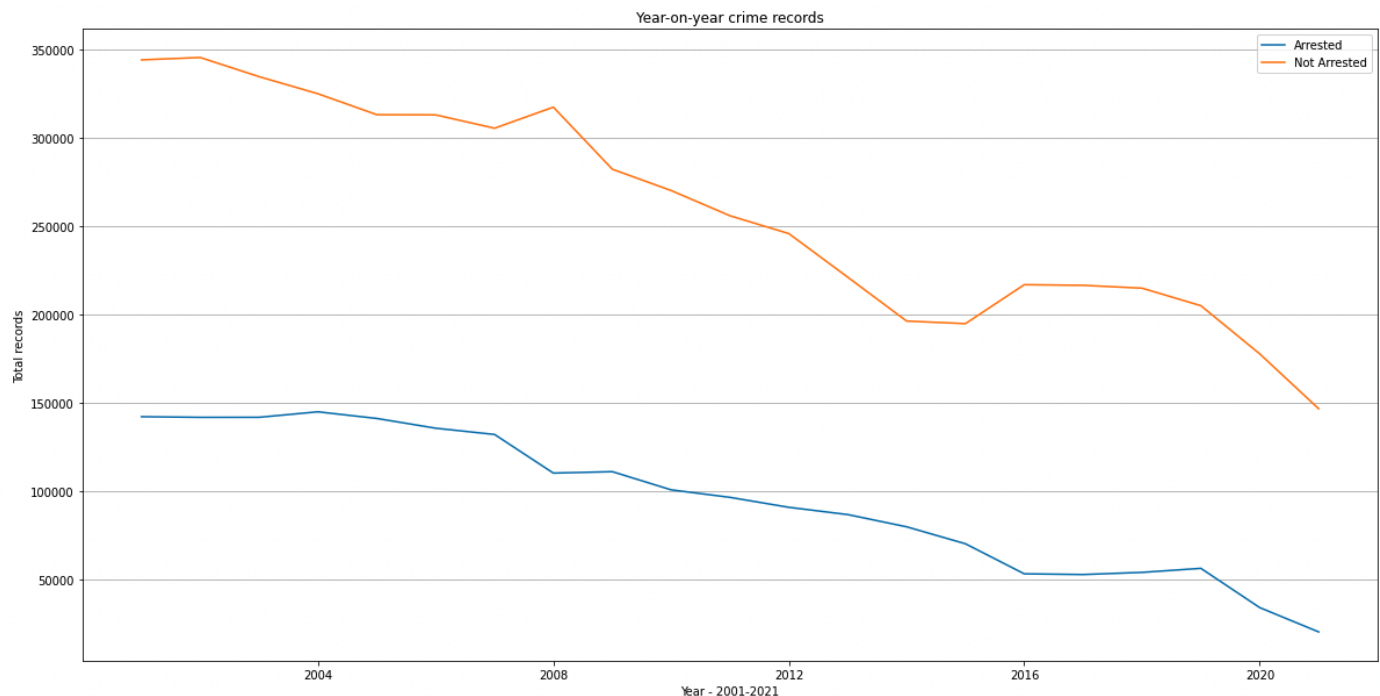
```
Out[73]:
  arrest  month  count  yearpd
0  False  2001-01-01  343894  2001
1  False  2001-01-01  141924  2001
2  False  2002-01-01  345216  2002
3  False  2002-01-01  141560  2002
4  False  2003-01-01  334388  2003
```

```
In [100]: # Data for plotting
t = type_arrest_pddf['count'] #~ 20 # np.arange(0.0, 2.0, 0.01)
s = type_arrest_pddf['month']

arrested = type_arrest_pddf[type_arrest_pddf['arrest'] == True]
not_arrested = type_arrest_pddf[type_arrest_pddf['arrest'] == False]

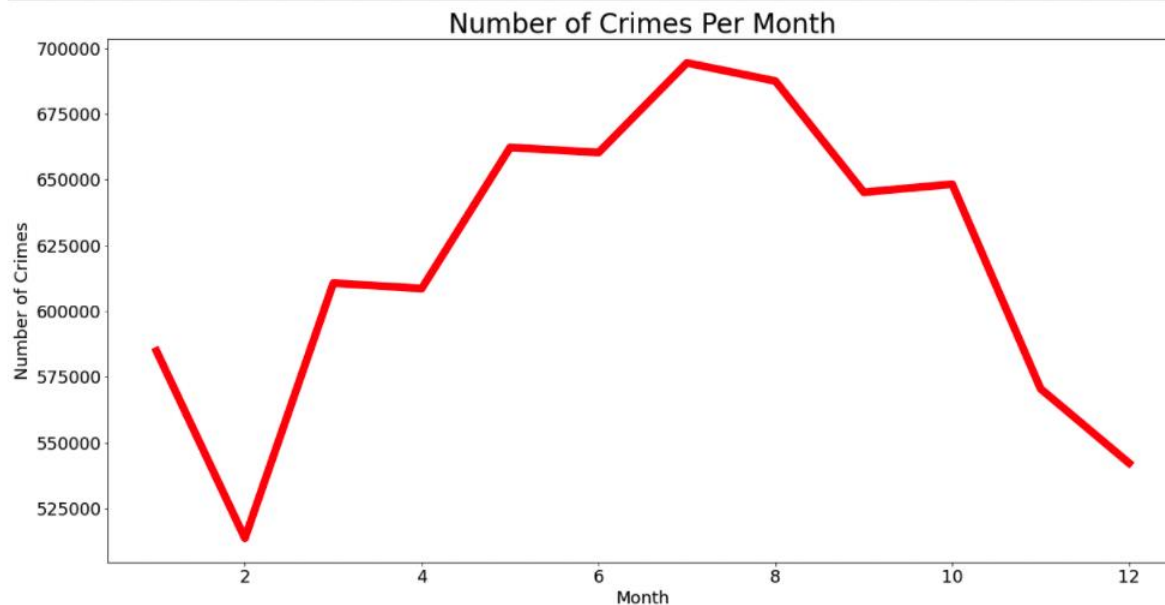
# Note that using plt.subplots below is equivalent to using
# fig = plt.figure() and then ax = fig.add_subplot(111)
fig, ax = plt.subplots()
ax.plot(arrested['month'], arrested['count'], label='Arrested')
ax.plot(not_arrested['month'], not_arrested['count'], label='Not Arrested')

ax.set(xlabel='Year - 2001-2021', ylabel='Total records',
       title='Year-on-year crime records')
ax.grid(b=True, which='both', axis='y')
ax.legend()
```



Crimes count month wise:

```
In [86]: from pyspark.sql.functions import month
monthdf = df.withColumn("Month", month(df["Date_time"]))
monthCounts = monthdf.select("Month").groupBy("Month").count()
monthCounts = monthCounts.collect()
months = [item[0] for item in monthCounts]
count = [item[1] for item in monthCounts]
crimes_per_month = {"month": months, "crime_count": count}
crimes_per_month = pd.DataFrame(crimes_per_month)
crimes_per_month = crimes_per_month.sort_values(by="month")
crimes_per_month.plot(figsize=(20,10), kind="line", x="month", y="crime_count",
                        color="red", linewidth=8, legend=False)
plt.xlabel("Month", fontsize=18)
plt.ylabel("Number of Crimes", fontsize=18)
plt.title("Number of Crimes Per Month", fontsize=28)
plt.xticks(size=18)
plt.yticks(size=18)
plt.show()
```



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" crimes occurred.

Discussions:

Data Types and Sources:

Bigdata is a phrase that refers to a large collection of data that continues to increase rapidly over time. In our case the data collected over years of crime is very large and it is regularly being updated in weekly basis as we speak of it is growing exponentially with time. The source of the data that we used is a public domain website, The City of Chicago website, it was a structured data one of the three types of big data that we learnt in the INFO-I535 class, structured data is relatively easy to use as it reduces the preprocessing work also very easy to understand by the first look which is comparatively a bit difficult in case of unstructured data. Structured data is data that follows a pre-determined data model and is thus easy to analyze. Structured data is organized in a table with relationships between the rows and columns. We also make use of spark data frame which is again one of the most important data structures.

Virtualization:

A virtual representation of a computing resource, such as a computer, server, or other hardware component, or a software-based resource, such as an operating system, is created by virtualization. We had learnt virtualization in our class which helped me setup a system on the cloud and use its resources instead using my limited local computer resources. When we are dealing with big data, we need higher storage a higher processing power both issues are solved by the help of virtualization. By using the virtual machine, it does not alter my local computer dependencies, also it benefits me by adjusting the resources required for a certain project and hence helping in resource management which is one of the major costs affecting factor in big data management. I achieved virtualization through Jetstream which I was totally unfamiliar with before this class.

Distributed Computing and File Systems:

A PySpark DataFrame is a collection of data that has been structured into named columns and is disseminated. It's similar in principle to a table in a relational database or a data frame in R/Python, but with more advanced optimizations. DataFrames may be built from a variety of sources, including structured data files, Hive tables, external databases, and RDDs. In the project at steps such as processing a function we have used RDD functions available in the PySpark. Apache Spark is a data processing framework that can handle enormous data sets quickly and distribute processing duties across many computers, either on its own or with other distributed computing tools. PySpark is a Python-based version of Apache Spark. Large-scale exploratory data analysis, machine learning pipelines, and data platform ETLs are all possible with it. If you're already familiar with Python and packages like Pandas, PySpark is a great language to learn. It will assist you in developing more scalable analytics and pipelines. This article explains how to get started with PySpark and complete common tasks. RDDs are Spark's primary logical data units. They are a distributed collection of things that are kept in memory or on the disks of many cluster machines. A single RDD can be partitioned into many logical partitions, which can then be stored and processed on various cluster computers.

We learnt RDD functions through the labs in the class that had the objective to use RDD function and create a Bag of Words model, we worked with unstructured data that was an E-book and uses RDD functions such as filter (), count (), withcolumns () to analyze the data. The introduction of the functions in that lab helped use functions in this project with better understanding.

Ingest and Storage:

The first layer in the Big Data Architecture is Data Ingestion, which oversees gathering data from various data sources. For this project we ingested the dt using the Big Query API helper module available in Kaggle. Data is transported from the data sources to the data target in batches at predetermined intervals, which is known as batching data intake. When enterprises need to acquire daily, batch ingestion comes in handy. We have used Batch ingestion as we have such kind of ingestion which was learned during the course work. Although data importation is not an easy process to write and may be costly to set up and maintain over time, a well-written data ingestion process can assist a firm in making choices and improving business operations. Our process was a simple ingestion process through SQL query and through a single source. Furthermore, this procedure can make it simpler to deal with a big number of data sources and provides engineers and analysts with quick access. As we used Big query to ingest we have not completely adopted to the GCP cloud but we have only taken the help of GCP API big query, GCP may be good in many aspects but for beginners it might take time to get used to, such as the methods to visualize and may take time to properly implement hence for the convenience of the project GCP big Query is used only to extract data and then rest process is done using other technologies which offer far better flexibility and ease than GCP.

Processing and Analytics:

The process of identifying trends, patterns, and correlations in enormous volumes of raw data in order to make data-informed decisions is known as big data analytics. These procedures use well-known statistical analysis approaches, such as clustering and regression, and apply them to larger datasets using modern tools. We made several analyses and processed the data at several stages on our data by using filter and other functions such as group by. We converted certain columns in our data frame to data time format to help in our visualization, we found the means and other statistical function on our data in order to better analyze the data. We used Batch processing that I was familiarized in the class with Batch processing is one approach for processing huge data chunks over time. When there is a longer interval between gathering data and evaluating it, batch processing comes in handy.

Lifecycles and Pipeline:

This is another one of the topics that we came across in the course, the whole Lifecycle and pipeline uses in the project is explained in the methodologies. A data pipeline is a series of steps that transport data from one location to another. Filtering, cleaning, aggregating, enriching, and even analyzing data-in-motion can all be part of a pipeline. Data pipelines provide speedy data analysis for insights by aggregating data from all your diverse sources into one common destination. They also guarantee that data quality is constant, which is critical for accurate insights.

Impact of Big Data:

Analyzing vast volumes of data from many sources in a variety of forms and kinds in a timely manner. Making better-informed judgments more quickly for more successful strategizing, which may benefit and enhance the supply chain, operations, and other strategic decision-making sectors. As we can see in your trends of crime there was a decrease in the crimes over years, this is a certain impact on the society, though the analysis our for example better decision were taken in account, better policing strategies were employed to reduce the crime over years. This is a major impact of the big data

Modeling:

Modeling is a future scope of this project though we have not done any modeling in the project with this which may benefit in strategic decision-making. We can predict what kind of crime is going to occur given the selected features by using different algorithms available in the ML libraries in Pyspark. The act of generating a visual representation of an entire information system or sections of it in order to express linkages between data points and structures is known as data modeling. This creates a standardized, consistent, and repeatable method for identifying and managing data resources.

Interpretation of the Results:

The insights that we can observe from the above visualizations are that **Theft** Type of crime is the one which is committed in majority and when we look at the locations its make sense that **street** is the best Location that, (most commonly theft occurs on street), so we see that the street had the highest Number of crimes in its occurrences. We see a steady decline in the number on crimes on yearly basis with **2002-2002** being the highest and **2021** being the lowest in the city of Chicago. The Same trend can be observed in the Arrested and Not Arrested time series graph, there is a decrease in both the type of crime result over the years. We have also plotted the trend of crimes over the months we can see that it's the highest in **July**, this is a useful one as a general should be it go high in the holiday months such as **November or December** but it's not, we see the lowest peak in the month of **February**, but why do we see a steep decline in February we can gather more insights by more careful analysis the data.

Barriers:

The only issues I faced during the project was in my VM, once the VM was shelved then my spark Host and master driver IP we reset, to which I would have to reconfigure them to resume my work again on the VM, or another alternative was to leave on the VM instance until my next visit to the project.

Conclusions:

This data analysis project, we believe, will give us with a scientific view on Chicago's security situation and crime rate. Based on the analysis findings and visualization, we can see the most often occurring crimes and the frequently occurring places where crimes happened. Stealing, battery, criminal damage, and narcotics were the most prevalent offenses, accounting for major percent of all reported crimes. Crimes are most committed on the street, sidewalks, dwellings, and apartments, all of which are areas where people congregate. Even though there were many reported crimes in Chicago each year, the arrest rate was on a decreasing trend leading us to think that Chicago's police arrest or investigative tactics were ineffective in early years but improved over time. We believe that if our data analytics can offer us with all this information on Chicago's security state, a larger data analytics project will yield much more relevant information that can be utilized as a powerful source for taking intelligent steps to improve our cities' security status.

We have used the technologies learnt in INFO-I535 class such as Ingestion and storage, Virtualization, Distributed systems, Data life Cycle and Pipeline, Data types and Sources, Process and Analytics and Impact of Big Data.

In summary we ingested the data from a public domain website and store it on a virtual machine, virtual machine instance was created specifically for this project with the suitable configurations (as one might do according to the business needs), the concept of Life cycle and pipe line was used to give a structure to the whole big data and management of the data, various data structures were used in the project e.g.: structured data into spark data frame, the process and analytics were defined and applied in the project and finally the impacts of big data through this project was identified i.e. the law enforcements took measures to decline the crime over the years.

References:

1. <https://medium.com/@stafa002/my-notes-on-chicago-crime-data-analysis-ed66915dbb20>
2. <https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-Present/ijzp-q8t2>
3. <https://googleapis.dev/python/bigquery/latest/generated/google.cloud.bigquery.client.Client.html>
4. <https://www.analyticssteps.com/blogs/what-virtualization-cloud-computing-characteristics-benefits>
5. <https://cvw.cac.cornell.edu/jetstream/>
6. <https://digitalnow878391108.wordpress.com/2021/01/01/the-kdd-process-in-data-mining/>
7. <https://towardsdatascience.com/pyspark-f037256c5e3>
8. <https://www.geeksforgeeks.org/difference-between-spark-dataframe-and-pandas-dataframe/>
9. <https://www.kaggle.com/sohier/beyond-queries-exploring-the-bigquery-api>
10. <https://datascienceplus.com/spark-dataframes-exploring-chicago-crimes/> -<https://github.com/ernest-kiwele/chicago-crime-analysis-apache-spark/blob/master/spark-ml/chicago-crime-data-on-spark.ipynb>
11. <https://github.com/RiyaJoshi/BigDataAnalysis-CrimeData/blob/master/Crime%20Analysis.ipynb>
12. <https://www.kaggle.com/sharanbasavsumbad/query-of-chicago-crime-data-set/edit>
13. <https://medium.com/@stafa002/my-notes-on-chicago-crime-data-analysis-ed66915dbb20>
14. <https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-to-apache-spark-rdd-and-pyspark/>
15. <https://www.dummies.com/programming/big-data/big-data-visualization/the-importance-of-virtualization-to-big-data/>
16. <https://www.tableau.com/learn/articles/big-data-analytics>