MODERN DATA ARCHITECTURES FOR BIG DATA I

UK POLICE CRIME API

By Ignacio Alonso

Agenda for today!

[CONTEXT]

The Use Case: Problem Explanation

JSON

Data Sources: DATA.POLICE.UK

APACHE nifi

Data Ingestion:

1. Overcoming API limits
2. Retrieving the crime points

hadoop

Data Storage:

1. Storing every flow!
2. A Distributed File System

APACHE Spark™

Data Processing:

1. Refining the dataset
2. Analytics: EDA

Project Results & Our Conclusions

ie

The Use Case: Problem Explanation

We wanted to examine where crime occurs in the most touristic areas of London. Where does it happen? What is the percentage of crimes successfully solved?

This is very useful for various reasons:

1. **Overall Safety:** Police can target conflict zones on the go, keeping track of every incident in each area.

2. **Police Resource Allocation:** Look at historical increases or decreases on crime activity by area, for police forces to concentrate on some neighborhoods or others.

3. **Business planning:** opening a car shop away from where 'vehicle-crime' occurs! Same with malls and 'shoplifting'!

In our case, we believe that the UK Police should power this API even further, to calculate increases and decreases in neighborhoods and act upon those increases and decreases.

Data Sources: DATA.POLICE.UK

We are going to look ONLY inside this area due to API limitations. Comprehends all crimes between Paddington to Charing Cross, this includes popular areas like Hyde Park, Mayfair, Covent Garden or Soho for October 2023.

Our data sources are a collection of REST-APIs from the UK Police department. We will look at the crimes reported by the Metropolitan Police (London's police excluding City of London).

In particular, we used: **crime-street API,**
https://data.police.uk/docs/method/crime-street/

Data Ingestion: Overcoming API limits

Custom area

poly	The lat/lng pairs which define the boundary of the custom area
date	**Optional**. (YYYY-MM) Limit results to a specific month. The latest month will be shown by default

If a custom area contains more than 10,000 crimes, the API will return a **503** status code.

The `poly` parameter is formatted in lat/lng pairs, separated by colons:

```
[lat],[lng]:[lat],[lng]:[lat],[lng]
```

The first and last coordinates need not be the same — they will be joined by a straight line once the request is made.

The API will return a **400** status code in response to a GET request longer than 4094 characters. For submitting particularly complex `poly` parameters, consider using POST instead.

Crime category

You can provide any crime category as part of the request URL. In the examples below, we use `all-crime`.

1. Need to use POST method instead of GET due to HTTP 400.
2. Need to limit the polygon we build due to 10K limit and HTTP 503.
3. Need to construct a flow per category to get all crimes from the polygon without getting HTTP 503.

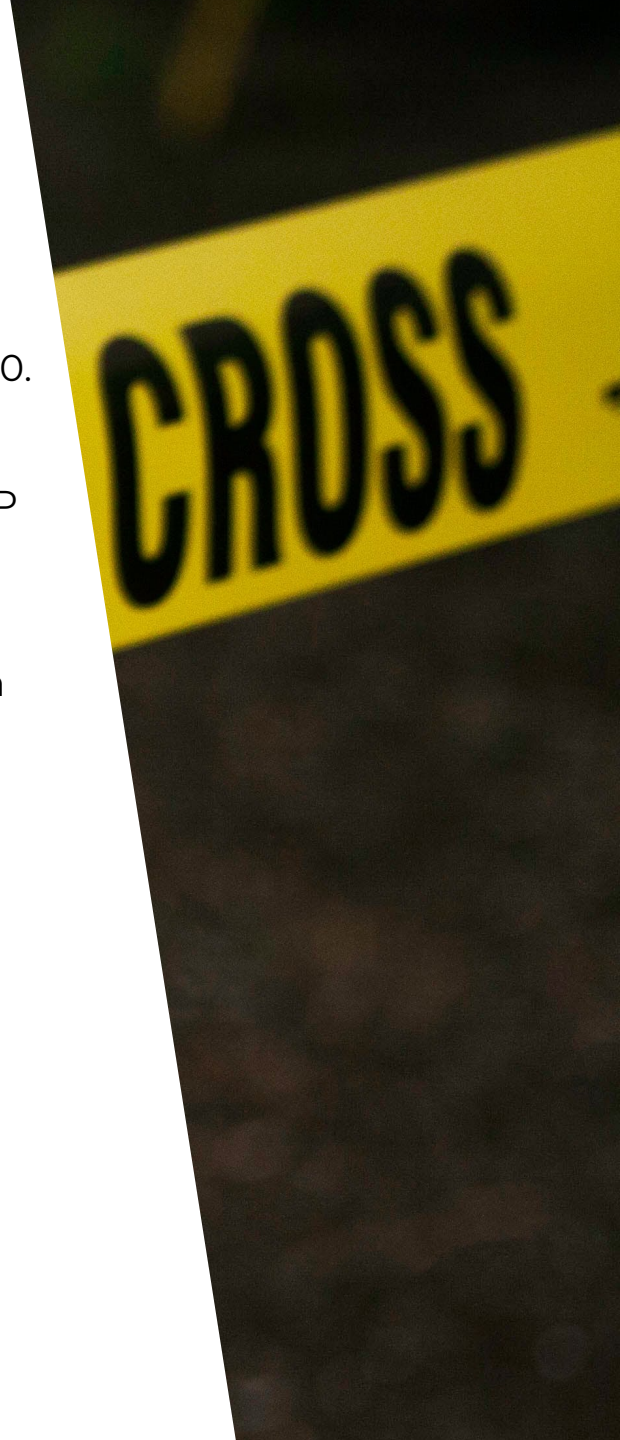
Polygon made based on latitudes and longitudes

HTTP Method: POST

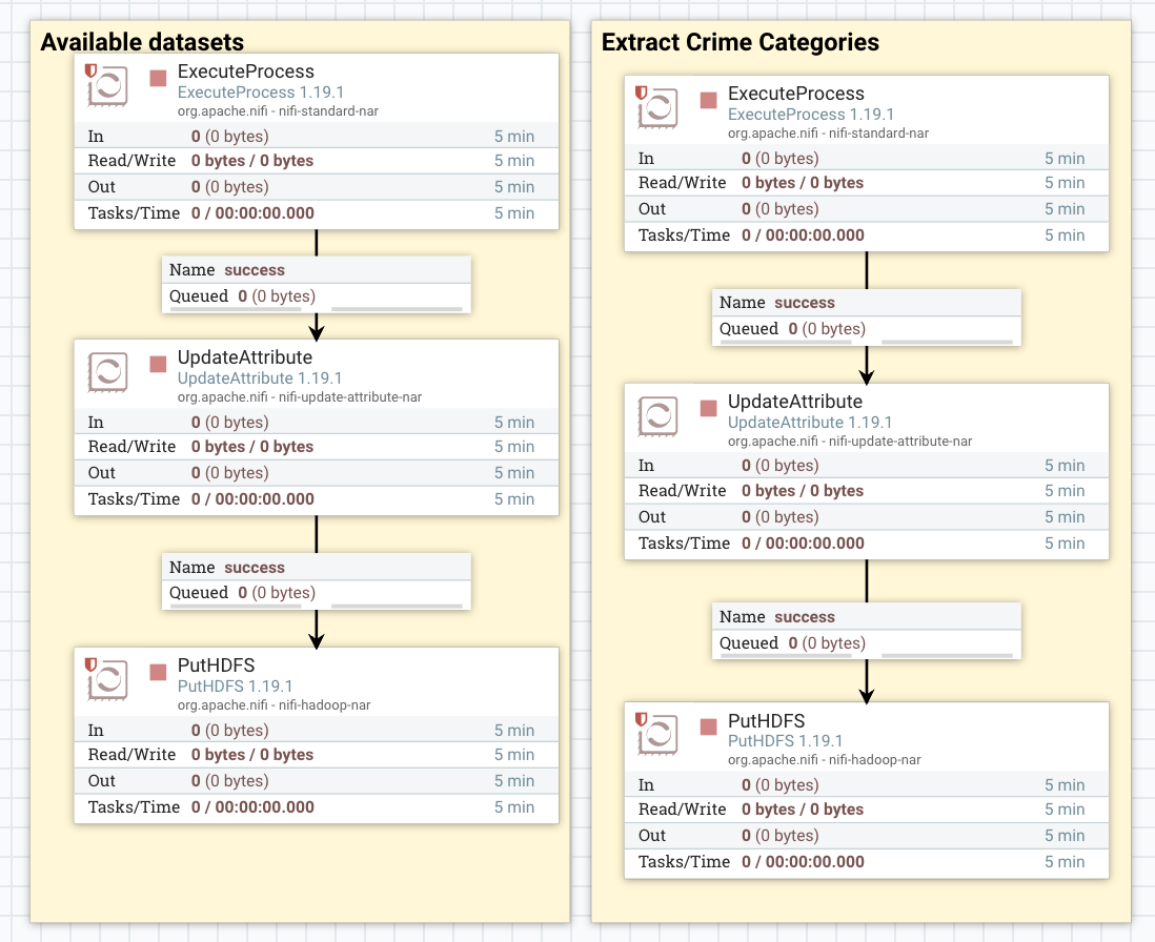
-s -X POST “https://data.police.uk/api/crimes-street/$.{crime_category}?**poly**=$.{poly}&**date**=$.{date} “

Crime_category: all-crime or burglary

Date of crime: 2023-10



Data Ingestion: Retrieving the crime points



We used two initial flows to know:

1. Available datasets (to work with the latest)

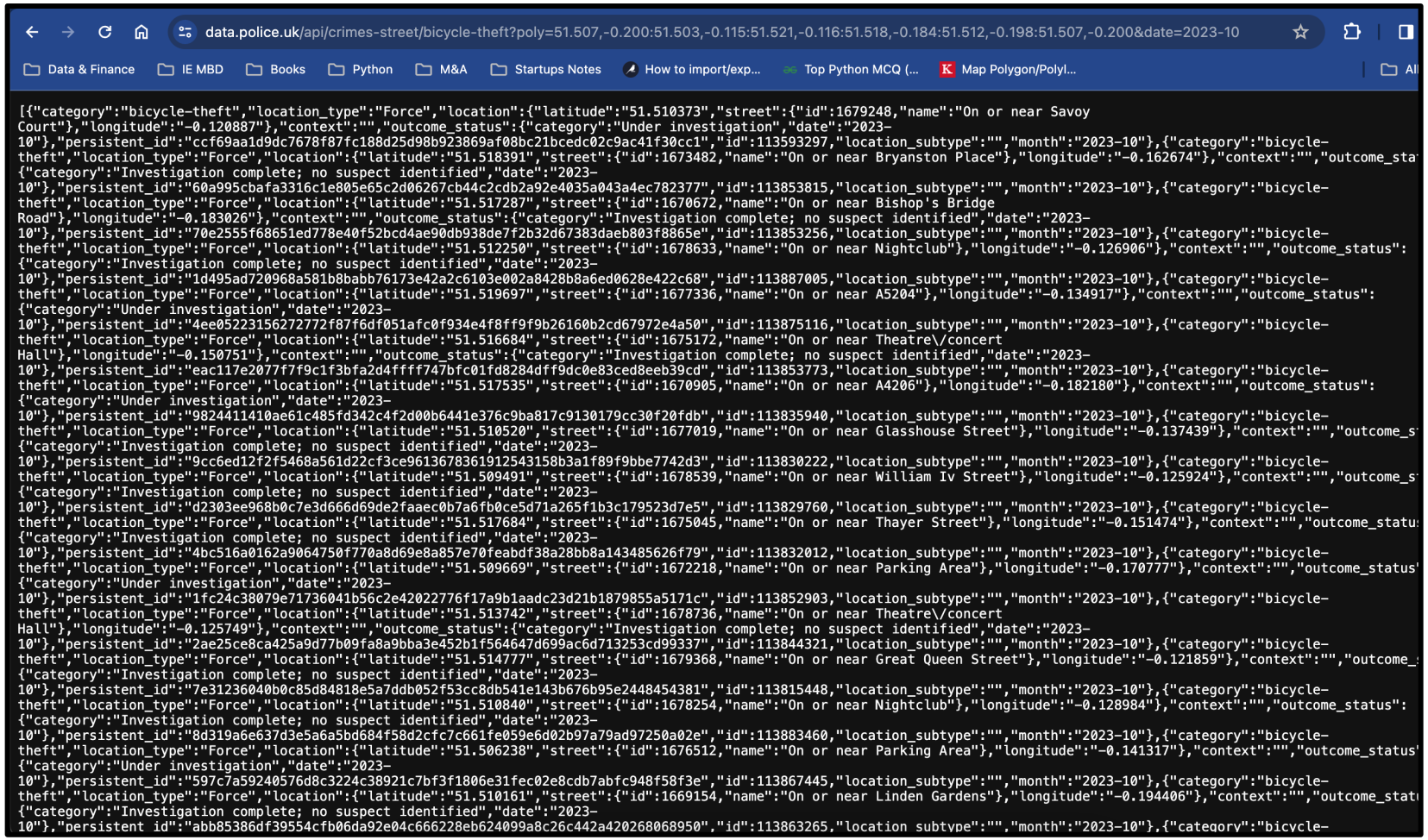
2. All possible crime-categories (to know the URL code to put in each request).

Since it is an open API, you can launch the request directly in your browser through the URLs provided here.

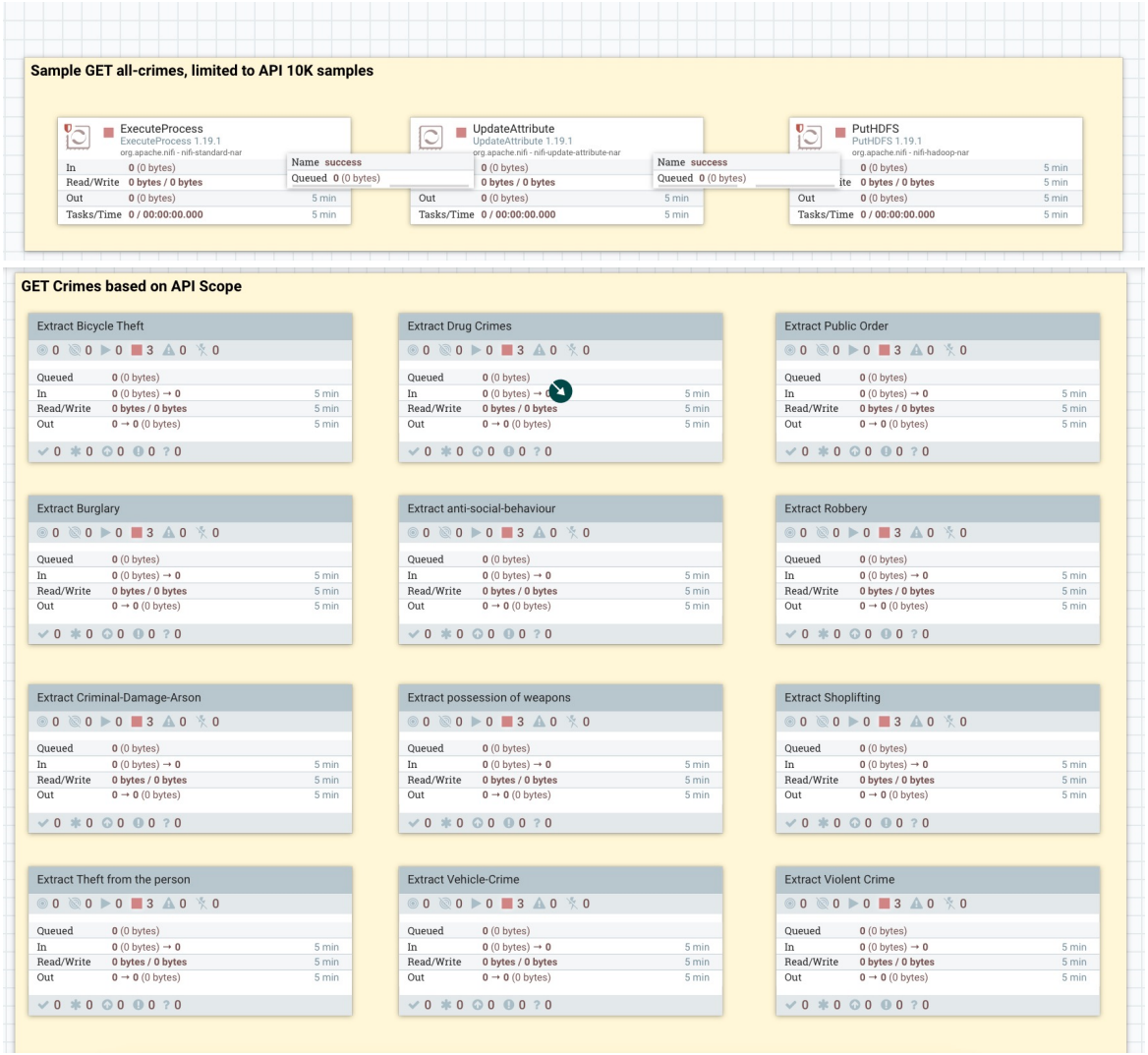
https://data.police.uk/api/crime-categories &
https://data.police.uk/api/crimes-street-dates

Data Ingestion: Retrieving the crime points

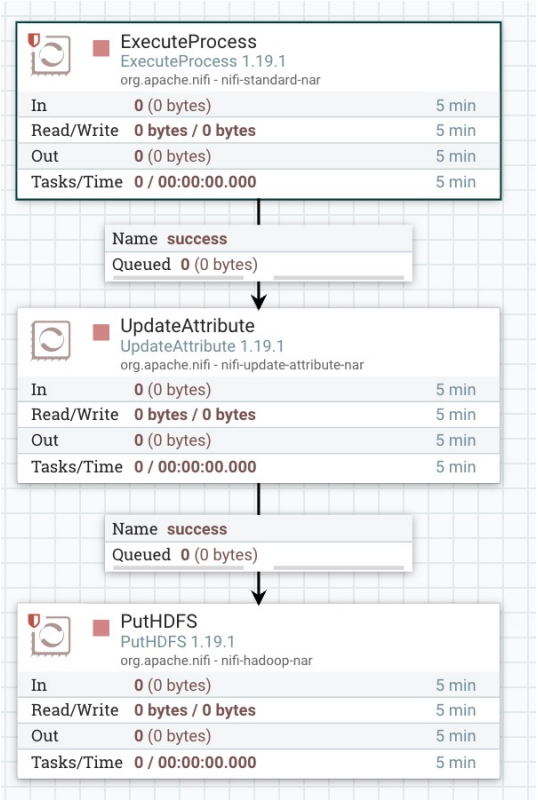
Example of API Request directly through browser

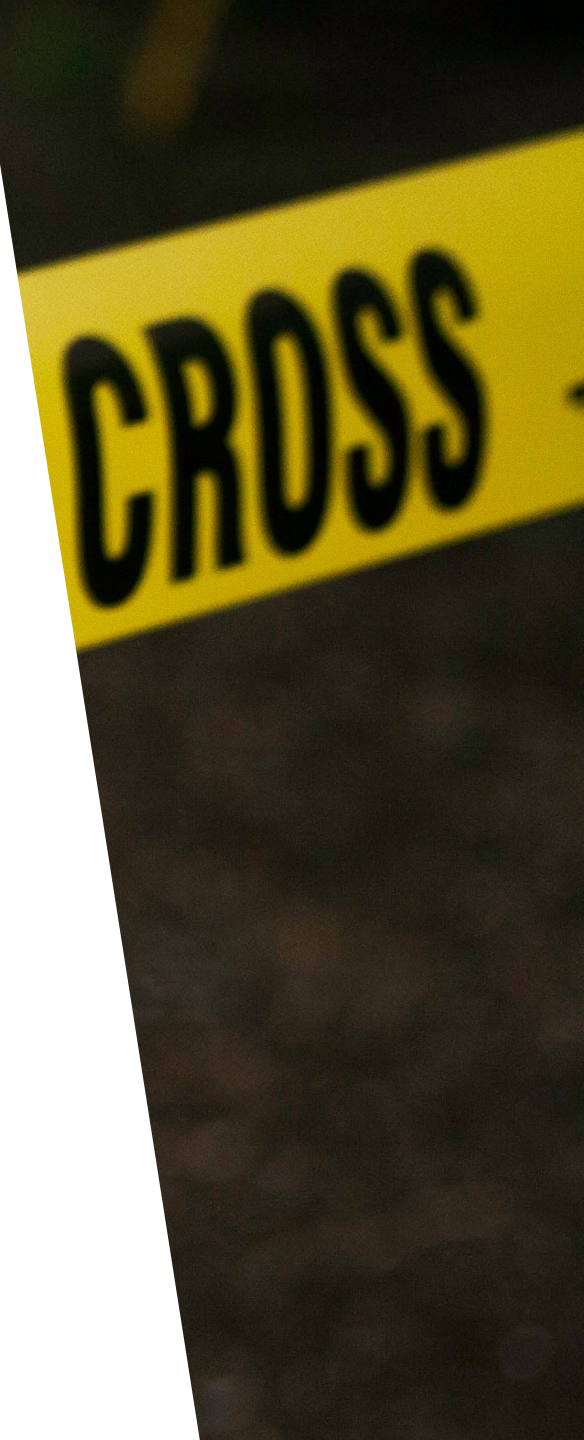


Data Ingestion: Retrieving the crime points

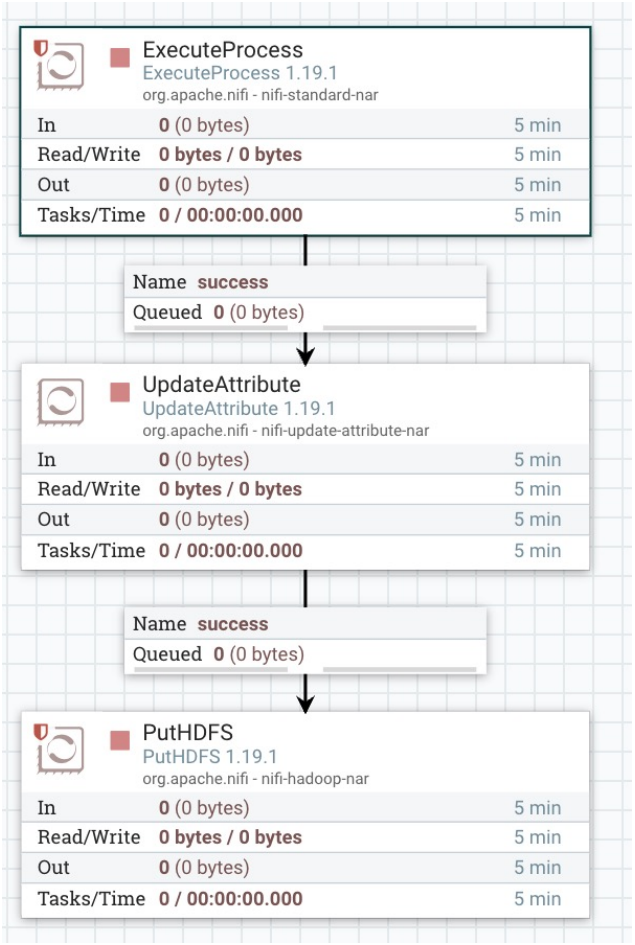


For each crime category or type, we will do the exact same flow, changing the API request

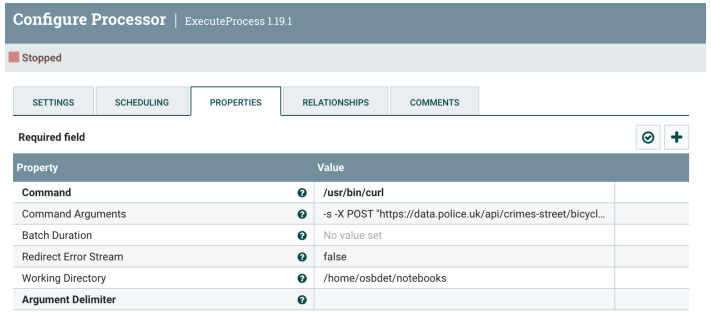




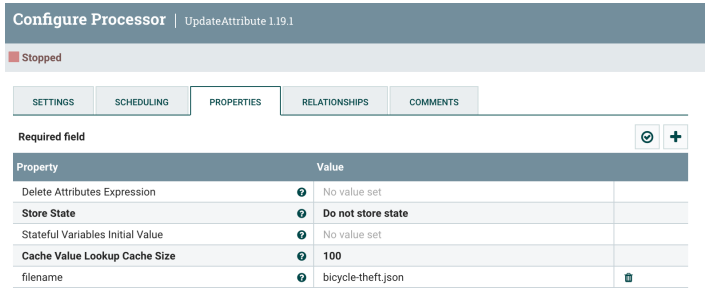
Data Ingestion: Retrieving the crime points



Execute Process: Makes the API call, on success, it retrieves and converts to flowfile.



Update Attribute: Gets the flowfile and saves it as a crime-category-name.json file to HDFS



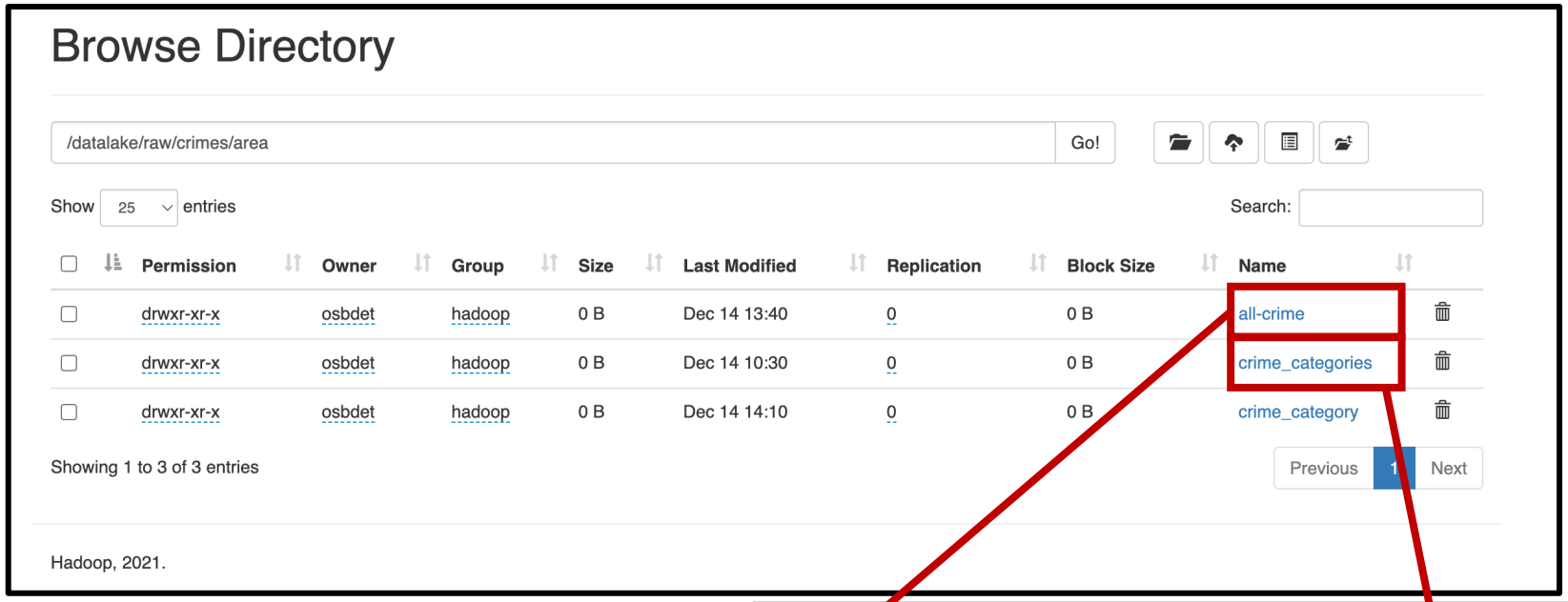
Put HDFS: Finds the HDFS module through configuration. Creates directory “crimes” to store the .json flowfile

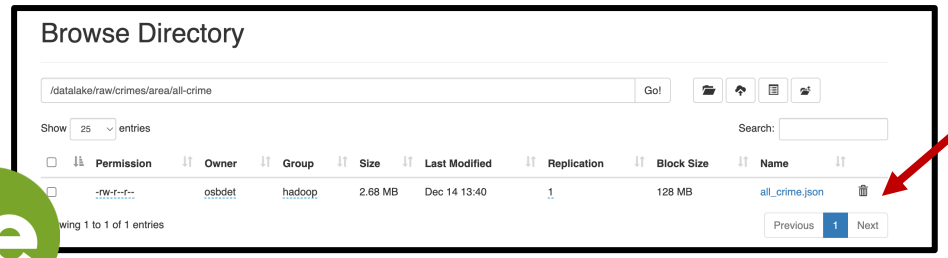
Importat Note: we could also use an InvokeHTTP processor instead of Execute Process to make the API call. Or even a GenerateFlowFile and ReplaceText before to customize the API call.

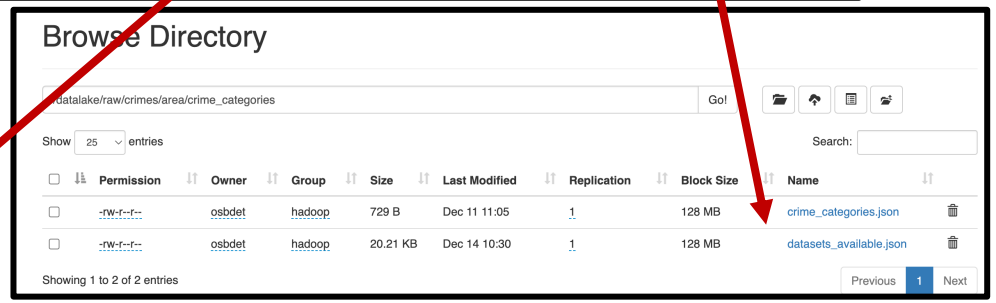
Data Storage: Storing every flow!

Using Hadoop Distributed File System, we created a common repository: Crime, under it, there is area (our selected geo-polygon) and under it we created 3 folders: all-crime, crime_categories and crime_category.

Check the two first repositories!

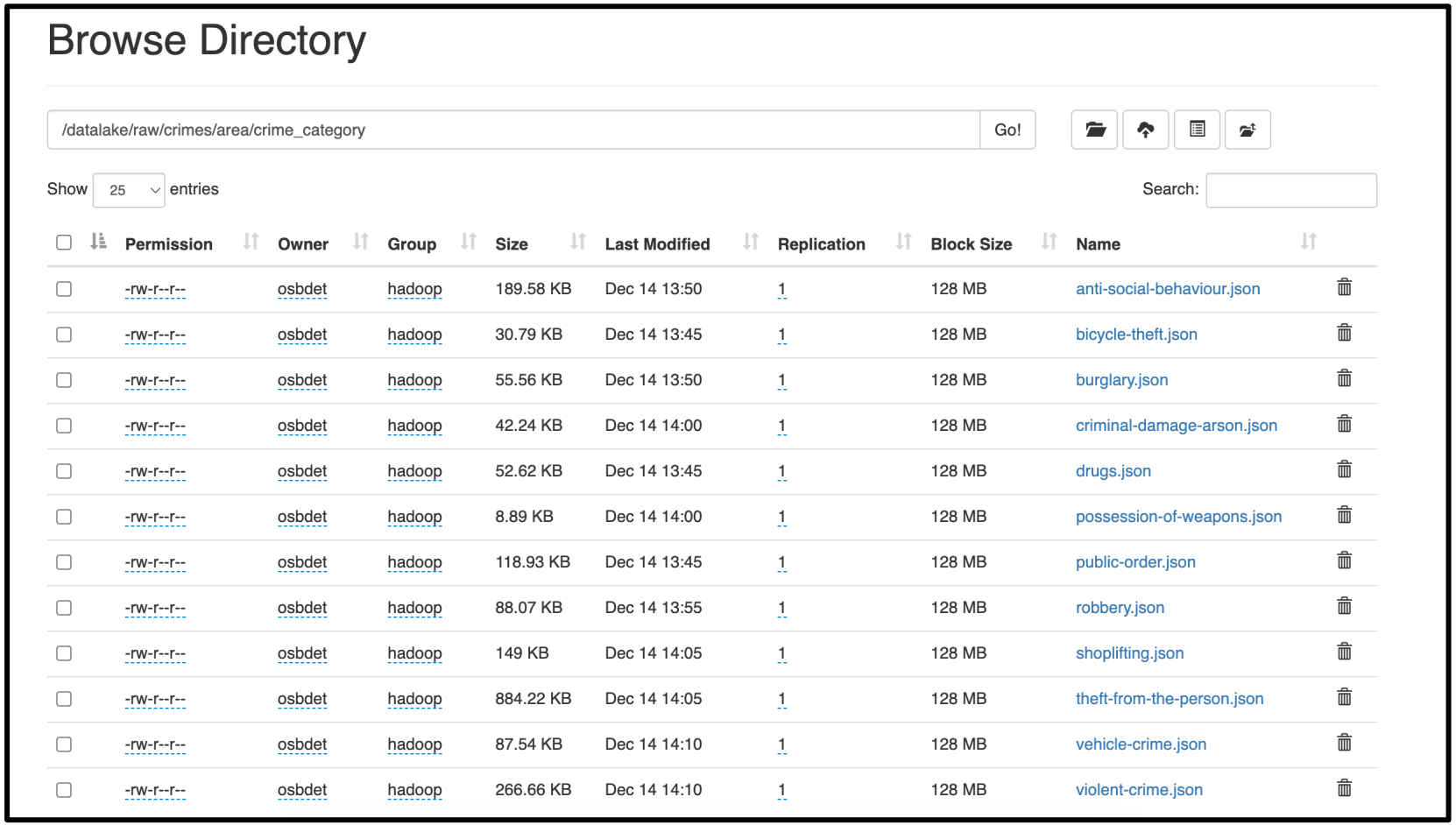






Data Storage: A Distributed File System

In crime_category, we can check every single .json file extracted per category and flow in NiFi. These are the files that we will use in our Spark Session.



Browse Directory

/datalake/raw/crimes/area/crime_category Go!

Show 25 entries Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	osbdet	hadoop	189.58 KB	Dec 14 13:50	1	128 MB	anti-social-behaviour.json
-rw-r--r--	osbdet	hadoop	30.79 KB	Dec 14 13:45	1	128 MB	bicycle-theft.json
-rw-r--r--	osbdet	hadoop	55.56 KB	Dec 14 13:50	1	128 MB	burglary.json
-rw-r--r--	osbdet	hadoop	42.24 KB	Dec 14 14:00	1	128 MB	criminal-damage-arson.json
-rw-r--r--	osbdet	hadoop	52.62 KB	Dec 14 13:45	1	128 MB	drugs.json
-rw-r--r--	osbdet	hadoop	8.89 KB	Dec 14 14:00	1	128 MB	possession-of-weapons.json
-rw-r--r--	osbdet	hadoop	118.93 KB	Dec 14 13:45	1	128 MB	public-order.json
-rw-r--r--	osbdet	hadoop	88.07 KB	Dec 14 13:55	1	128 MB	robbery.json
-rw-r--r--	osbdet	hadoop	149 KB	Dec 14 14:05	1	128 MB	shoplifting.json
-rw-r--r--	osbdet	hadoop	884.22 KB	Dec 14 14:05	1	128 MB	theft-from-the-person.json
-rw-r--r--	osbdet	hadoop	87.54 KB	Dec 14 14:10	1	128 MB	vehicle-crime.json
-rw-r--r--	osbdet	hadoop	266.66 KB	Dec 14 14:10	1	128 MB	violent-crime.json

Data Processing: Refining the dataset

Group Project - Group 3 - UK Police - reported Crimes in London

Setting the environment

We will import findspark to find the spark module in obsdet VM and use it with Jupyter

```
In [1]: import findspark

        # Findspark will help Jupyter Notebook find Spark within OSBDET
        findspark.init()
```

```
In [2]: import pandas as pd

        pd.set_option('display.max_colwidth', None)
```

Initiating **Spark Session** referencing this notebook as entry-point

```
In [3]: from pyspark.sql.session import SparkSession

        spark_session = \
          SparkSession.builder\
                      .appName("#Group Project - Group 3")\
                      .getOrCreate()

        print(f"This cluster relies on Spark '{spark_session.version}'")
```

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (fi
2.3.jar) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.sp
WARNING: Use --illegal-access=warn to enable warnings of further illegal re
WARNING: All illegal access operations will be denied in a future release
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLo
```

```
This cluster relies on Spark '3.2.3'
```

```
In [4]: from pyspark.sql.types import StructType, ArrayType
        import pyspark.sql.functions as F
```

Import the spark module, Start the SparkSession and import the functions and Pandas

Joining .JSON Files Pre-processing

Our approach is a bit different due to the API limitations. We will comment each file directory in HDFS and make a for loop to proccess and merge each file.

```
In [5]: file_paths = [
            'hdfs://localhost:9000/datalake/raw/crimes/area/crime_category/shoplifting.json',
            'hdfs://localhost:9000/datalake/raw/crimes/area/crime_category/possession-of-weapons.json',
            'hdfs://localhost:9000/datalake/raw/crimes/area/crime_category/bicycle-theft.json',
            'hdfs://localhost:9000/datalake/raw/crimes/area/crime_category/robbery.json',
            'hdfs://localhost:9000/datalake/raw/crimes/area/crime_category/anti-social-behaviour.json',
            'hdfs://localhost:9000/datalake/raw/crimes/area/crime_category/burglary.json',
            'hdfs://localhost:9000/datalake/raw/crimes/area/crime_category/public-order.json',
            'hdfs://localhost:9000/datalake/raw/crimes/area/crime_category/drugs.json',
            'hdfs://localhost:9000/datalake/raw/crimes/area/crime_category/criminal-damage-arson.json',
            'hdfs://localhost:9000/datalake/raw/crimes/area/crime_category/violent-crime.json',
            'hdfs://localhost:9000/datalake/raw/crimes/area/crime_category/theft-from-the-person.json',
            'hdfs://localhost:9000/datalake/raw/crimes/area/crime_category/vehicle-crime.json'
        ]
```

We need to make the file ready to see them together, or just use the all_crime.json. For the sake of being ordered, we will make a for loop going through every .json file the API extracted. After that we will extract the features using our spark_session, **flattening the files**

```
In [6]:
        dataframes = []

        for element in file_paths:
            df = spark_session.read.json(element)

            df_selected = df.select(
                F.col('id').alias('crime_id'),
                F.col('category').alias('crime_type'),
                F.col('month').alias('incident_month'),
                F.col('location.latitude').alias('latitude'),
                F.col('location.longitude').alias('longitude'),
                F.col('location.street.name').alias('street_name')
            )
            dataframes.append(df_selected)
```

Now, we do another for loop and merge them together

```
In [7]: # Merge all dataframes into a single dataframe using union
        merged_df = dataframes[0]
        for df in dataframes[1:]:
            merged_df = merged_df.union(df)
```

We will safe all file paths to a list, loop it around and FLATTEN the file in a merged data frame with UNION

Data Processing: Refining the dataset

Let's display the schema and translate the merged dataframe into Pandas!

In [8]:
```
# Displaying the structure and some rows of the merged dataframe
merged_df.printSchema()

merged_df\
    .limit(10).toPandas()
```

```
root
 |-- crime_id: long (nullable = true)
 |-- crime_type: string (nullable = true)
 |-- incident_month: string (nullable = true)
 |-- latitude: string (nullable = true)
 |-- longitude: string (nullable = true)
 |-- street_name: string (nullable = true)
```

Out[8]:

	crime_id	crime_type	incident_month	latitude	longitude	street_name
0	113817032	shoplifting	2023-10	51.514586	-0.152839	On or near Shopping Area
1	113880702	shoplifting	2023-10	51.512544	-0.145989	On or near Avery Row
2	113860497	shoplifting	2023-10	51.511893	-0.144070	On or near New Bond Street
3	113842744	shoplifting	2023-10	51.513350	-0.159461	On or near Marble Arch
4	113847514	shoplifting	2023-10	51.512326	-0.187879	On or near Bayswater
5	113835626	shoplifting	2023-10	51.513848	-0.129593	On or near Old Compton Street
6	113883092	shoplifting	2023-10	51.509592	-0.197729	On or near Bulmer Mews
7	113854698	shoplifting	2023-10	51.516295	-0.129487	On or near Conference/exhibition Centre
8	113874947	shoplifting	2023-10	51.514970	-0.127169	On or near Shaftesbury Avenue
9	113872097	shoplifting	2023-10	51.518567	-0.132282	On or near B506

In case we inserted duplicates in the frame while merging and re-executing. **This will drop duplicate crime ids**

In [9]:
```
merged_df.drop_duplicates()
```

Out[9]:
```
DataFrame[crime_id: bigint, crime_type: string, incident_month: string, latitude: st
_name: string]
```

And now, we will print the Schema of the resulting dataframe and send it to Pandas for display!!

Just in case, (and because we executed several times the loop to check if it works), we have also dropped any duplicate crimes from the resulting dataframe.

At the moment, we will play with this merged dataframe, but we will also look at other .json files for further analysis.

Data Processing: Analytics EDA

The first group bys tell us several things about crimes that occurred in October 2023 for the designated polygon or area. We retrieved 5202 crimes in total.

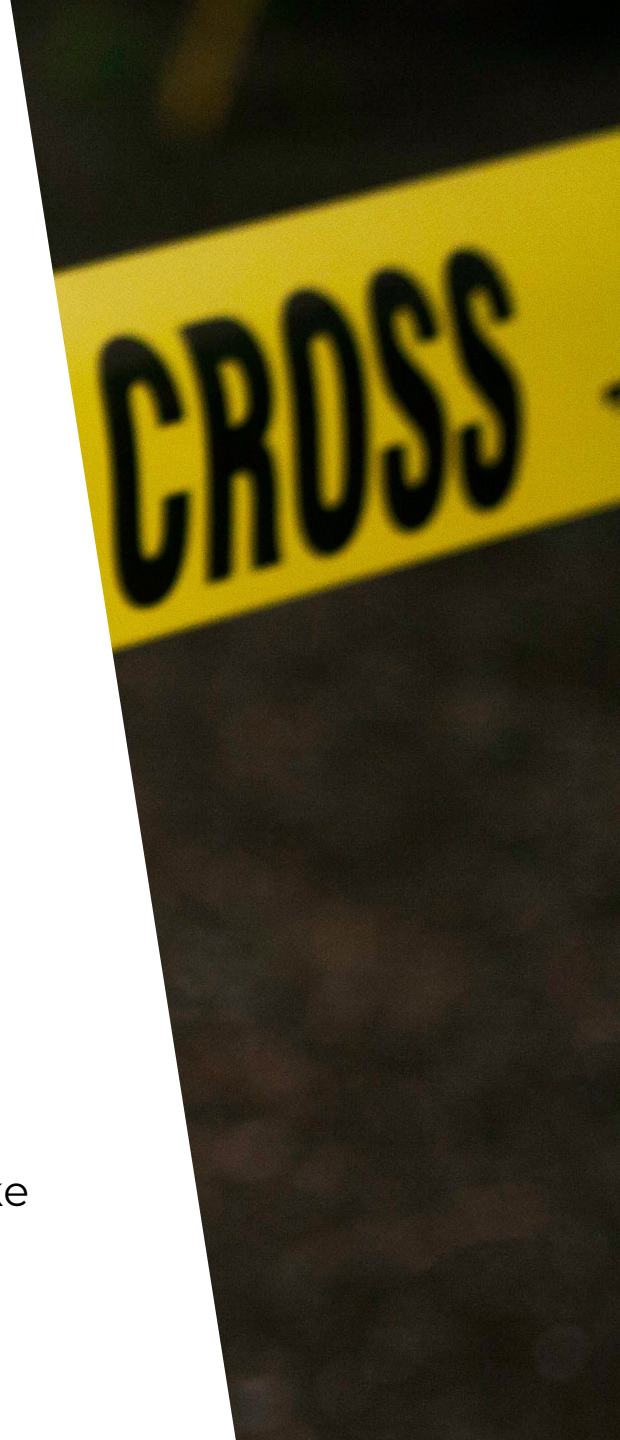
crime_type	count
theft-from-the-person	2197
violent-crime	697
anti-social-behaviour	671
shoplifting	382
public-order	314
robbery	230
vehicle-crime	221
burglary	142
drugs	140
criminal-damage-arson	107
bicycle-theft	78
possession-of-weapons	23

Insight 1: thefts account for 42.2% of the total crimes extracted. We are surprised by the amount of violent crime in this extract.

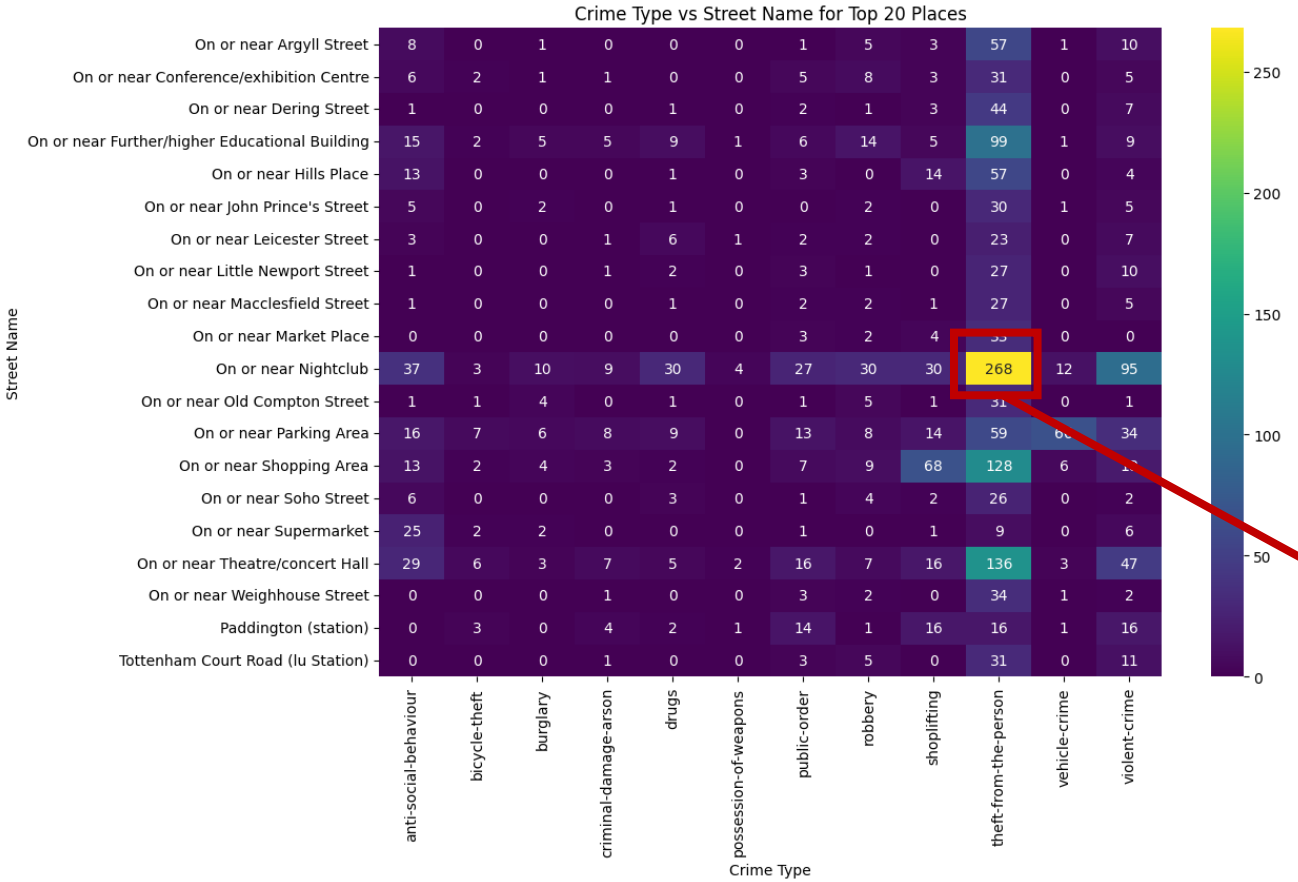
street_name	count
On or near Nightclub	555
On or near Theatre/concert Hall	277
On or near Shopping Area	260
On or near Parking Area	240
On or near Further/higher Educational Building	171
On or near Hills Place	92
On or near Argyll Street	86
Paddington (station)	74
On or near Conference/exhibition Centre	62
On or near Dering Street	59
Tottenham Court Road (lu Station)	51
On or near Old Compton Street	46
On or near Supermarket	46
On or near John Prince's Street	46
On or near Leicester Street	45
On or near Little Newport Street	45
On or near Soho Street	44
On or near Weighhouse Street	43
On or near Market Place	42
On or near Macclesfield Street	39

only showing top 20 rows

Insight 2: The probability of a crime increases whenever it is a populated area, like a nightclub, a theatre or a shopping hall!



Data Processing: Analytics EDA



Insight 3: Combining the two categorical values gives us a nice heatmap where to extract insights from the top 20 places where crimes occur. There is a pattern!

	outcome_status	count
0	(Local resolution, 2023-10)	1
1	(Awaiting court outcome, 2023-10)	1
2	(Under investigation, 2023-10)	104
3	(Investigation complete; no suspect identified, 2023-10)	162

Insight 4: Based on outcome_status, a variable we extracted later, we see that out of those 268 thefts that happen in nightclubs, only 1 has been satisfactory.

Data Processing: Analytics EDA

	outcome_status	category	count
0	(Investigation complete; no suspect identified, 2023-10)	theft-from-the-person	1254
1	(Investigation complete; no suspect identified, 2023-10)	other-theft	1014
2	(Under investigation, 2023-10)	theft-from-the-person	938
3	(Under investigation, 2023-10)	other-theft	923
4	None	anti-social-behaviour	671
5	(Under investigation, 2023-10)	violent-crime	549
6	(Under investigation, 2023-10)	public-order	225
7	(Investigation complete; no suspect identified, 2023-10)	shoplifting	186
8	(Under investigation, 2023-10)	shoplifting	181
9	(Under investigation, 2023-10)	robbery	173
10	(Investigation complete; no suspect identified, 2023-10)	violent-crime	131
11	(Investigation complete; no suspect identified, 2023-10)	vehicle-crime	122
12	(Under investigation, 2023-10)	vehicle-crime	99
13	(Investigation complete; no suspect identified, 2023-10)	public-order	78
14	(Investigation complete; no suspect identified, 2023-10)	burglary	73
15	(Under investigation, 2023-10)	criminal-damage-arson	64
16	(Under investigation, 2023-10)	burglary	64
17	(Under investigation, 2023-10)	drugs	61
18	(Investigation complete; no suspect identified, 2023-10)	robbery	54
19	(Local resolution, 2023-10)	drugs	51
20	(Investigation complete; no suspect identified, 2023-10)	bicycle-theft	48
21	(Investigation complete; no suspect identified, 2023-10)	criminal-damage-arson	35
22	(Under investigation, 2023-10)	bicycle-theft	30
23	(Offender given penalty notice, 2023-10)	drugs	17
24	(Under investigation, 2023-10)	possession-of-weapons	15
25	(Awaiting court outcome, 2023-10)	violent-crime	14
26	(Local resolution, 2023-10)	shoplifting	10
27	(Under investigation, 2023-10)	other-crime	9

Insight 5: Mmm, the last insight did not look good for the police.

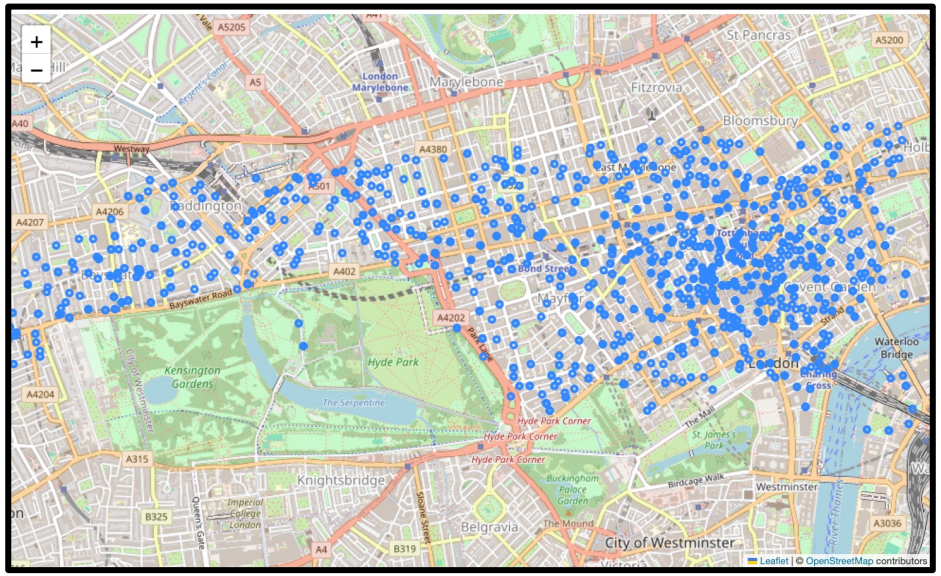
For total thefts, now we ask, what is the most common outcome?

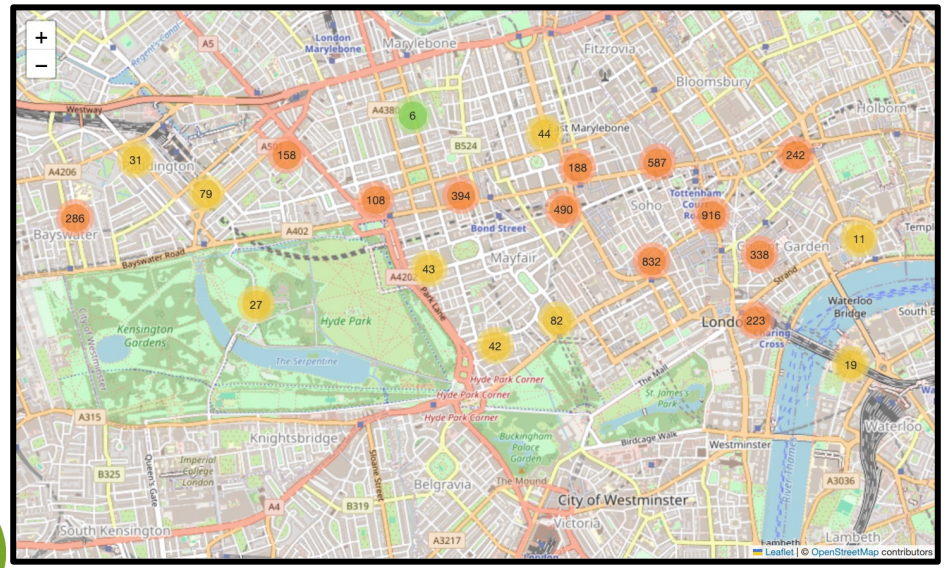
Out of the 2,197 cases, we can see that 1,254 have been completed without identifying the suspect and another 923 are under investigation.

The effectiveness rate of you finding your retrieved belongings is 0%, and the number of cases already closed is:

1254/2197 = 57.1%, only after 2 months!!

Data Processing: Analytics EDA



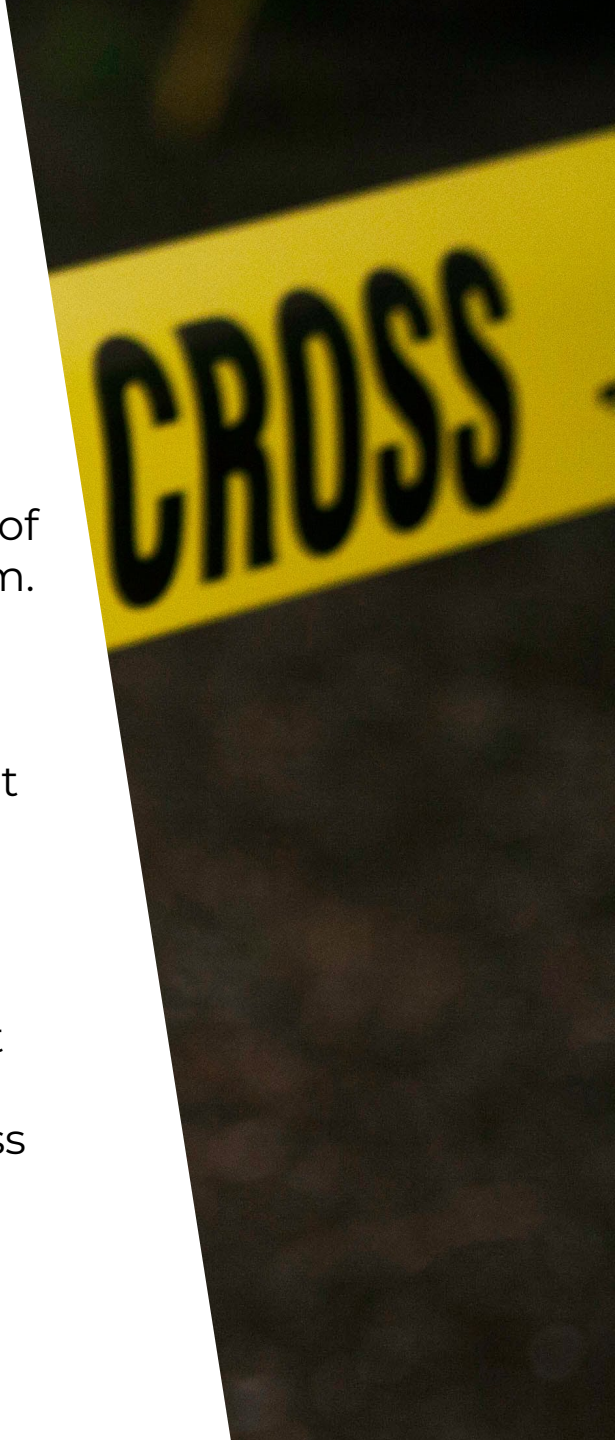


Insight 6:

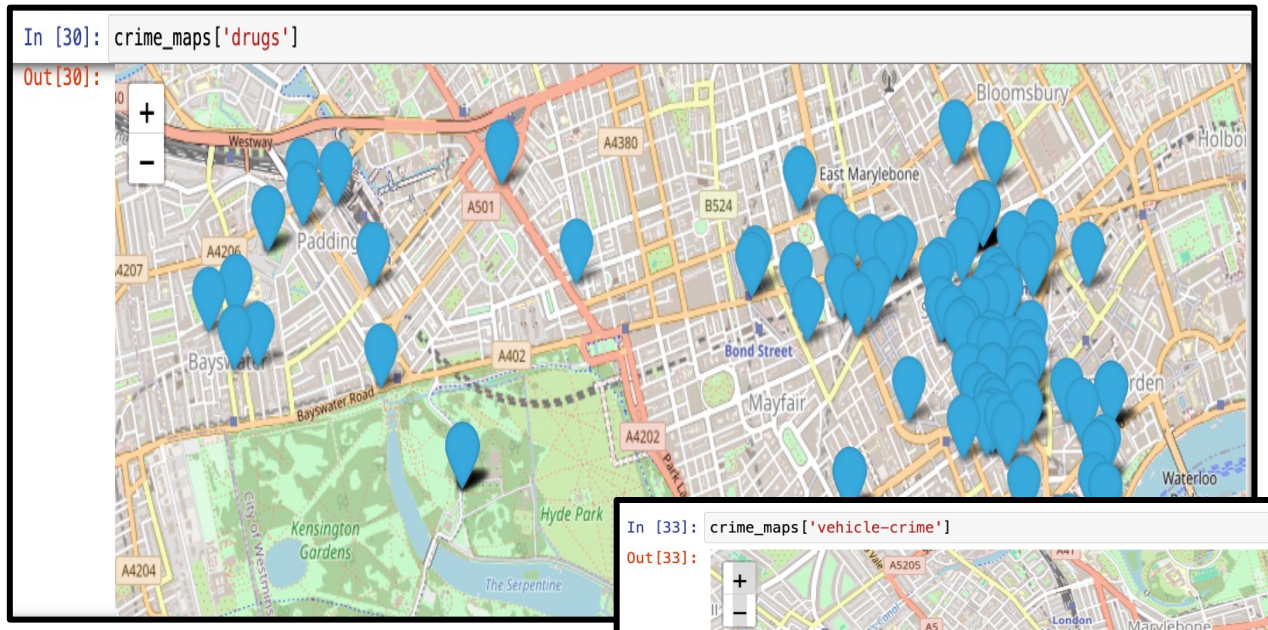
We have the longitude and latitude of crime locations so we can paint them.

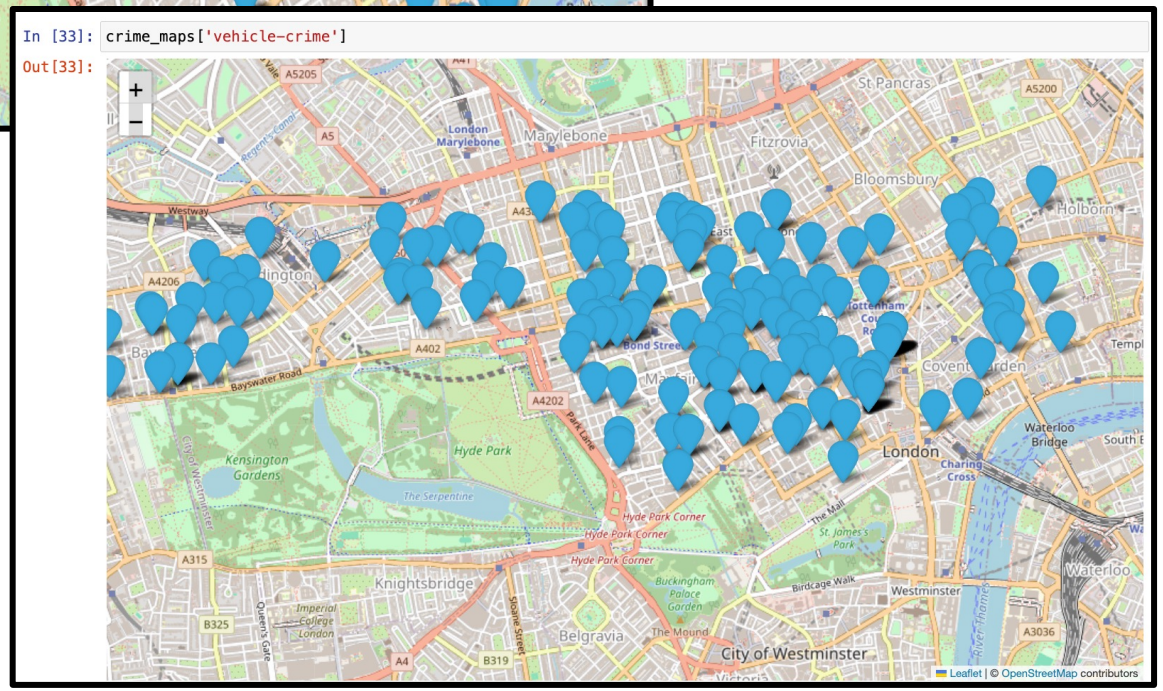
We see that SOHO is a congested area in the blue map, but we might not be able to determine this, since it is super crowded.

In this second map, we use zones to comprehend several crimes or markers and we can indeed see that Mayfair is doing fairly well, when compared to Soho and Charing Cross Station



Data Processing: Analytics EDA





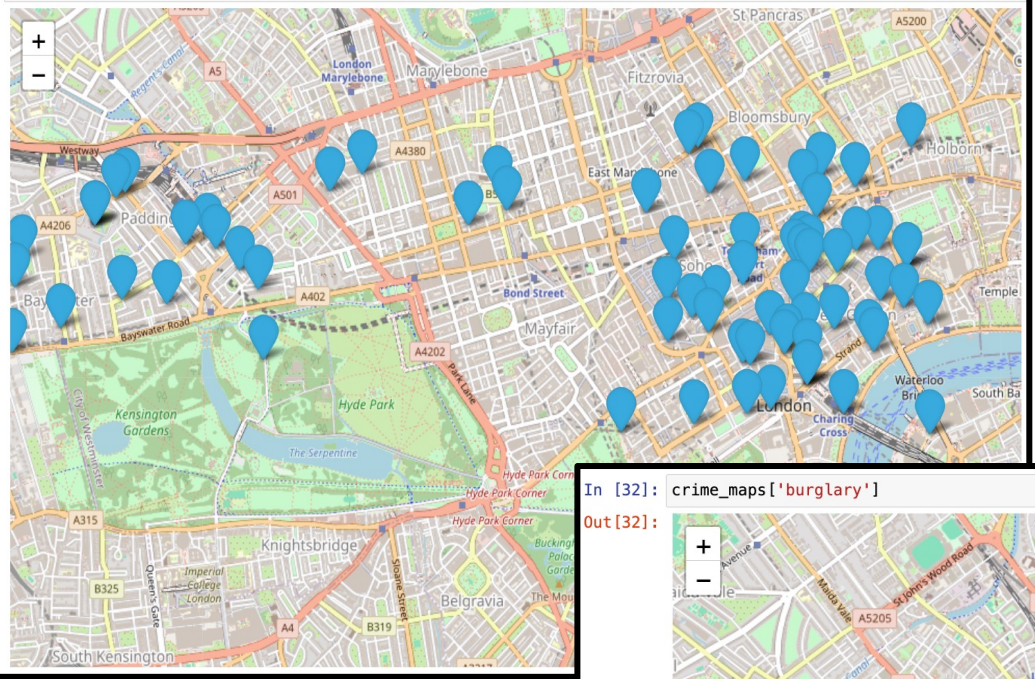
Insight 7:
Drug crime is very "localized" to an area, in this case Soho.

Vehicle Crime is more skewed towards the west, since Mayfair is wealthier than other neighborhoods

Data Processing: Analytics EDA

```
In [31]: crime_maps['bicycle-theft']
```

Out[31]:




```
In [32]: crime_maps['burglary']
```

Out[32]:

Insight 8:
Bicycle theft tends to be more prone in Stations, such as Paddington and Charing Cross. Burglary happens on wealthy neighborhoods like Bloomsbury or above Mayfair and next to the main road.

Project Results & Our Conclusions

Data Ingestion & API

1. An API this powerful should be used and implemented in every neighbourhood.
2. Street_name gives vage descriptions such as “On or near Nightclub” while others, it gives the exact name of the street. They should change it.
3. It would also be nice to get the outcome for every single crime category

Data Storage

1. A distributed file system is not optimal, and we would choose object storage next time.
2. Creating different directories to separate .json documents helped us a lot in dissecting the information

Data Processing

1. Different crimes are more frequent than others.
2. Crimes are more probable at open and crowded places, especially at night!
3. Different crimes types have different locations based on maps.
4. If Spark was very fast at our computer, we could do incredible things if given the opportunity to work for the police on a project with real clusters!

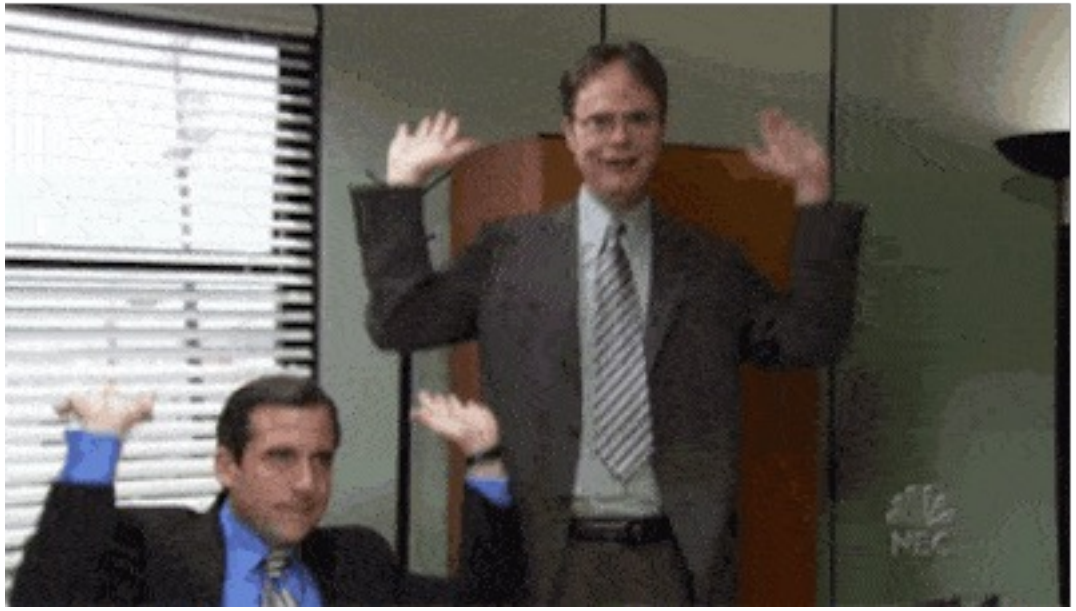
For Further Reference

API documentation: https://data.police.uk/docs/

Leaflet Maps: https://python-visualization.github.io/folium/latest/

A nice article on NiFi Flows: https://community.cloudera.com/t5/Community-Articles/Beast-Mode-Quotient-Part-1-Create-Nifi-Flows-to-ingest-API/ta-p/249212

CONGRATS! WE ARE DONE!



THANKS FOR LISTENING :D