Big Data Analytics Exam Project - A.A. 2021/22

Team members:
Agostino Antonino, 223958
Andronico Giorgio, 227815
Gianfranco Sapia, 223954

Introduction

The project deals with a dataset taken from Kaggle, providing nearly 12 years of crime reports from across all of San Francisco's neighborhoods. The aim is to classify the category of crime that occurred given time, location, and other features such as how the crime was resolved (i.e., if the subject was arrested or released).

Cluster configuration

Data cleaning, elaboration and modeling will be run on an Hadoop cluster, composed by three machines: one master and two slaves. The master machine is set-up as follows:

- OS: Ubuntu 18.04 LTS
- RAM: 4GB
- Hard Drive: 20GB

Both slaves are set-up as follows:

- OS: Ubuntu 18.04 LTS
- RAM: 2GB
- Hard Drive: 20GB

Technologies used:

Apache Hadoop is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. Both master and slave have configured the version *3.2.2*.

Apache Hive is a data warehouse software built on top of Apache Hadoop for providing data query and analysis. This software is set-up only on master with the version of *2.3.9*

Apache Sqoop is a tool design for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases. Also this software is set-up only on master machine with the version *1.4.7*

The last software set-up only on master is **Apache Spark**, with version *3.2.0*, which is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters.

Conda is a toolkit installed on master machine that equips users to work with thousand of open-source packages and libraries. This toolkit helped to create an enviroment with various library installed useful for the Python language.

Data ingestion

During this phase the file `train.csv` has been first uploaded on **MySQL Server** on the master machine. The first step was creating the database with the following query:

```
CREATE DATABASE crimes;
```

The next step was to create the table with this query

```
CREATE TABLE crimes(dates varchar(255), category varchar(255), descript
varchar(255), dayoftheweek varchar(255), pddistrict varchar(255), resolution
varchar(255), address varchar(255), longitude varchar(255), lat varchar(255));
```

After this the file located in `/var/lib/mysql-files` has been loaded inside the table crimes as follow:

```
LOAD DATA INFILE '/var/lib/mysql-files/train.csv' INTO TABLE crimes FIELDS
TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' LINES TERMINATED BY '\n' IGNORE 1
LINES;
```

This last step was necessary due to presence of comma in the fields of *Description* and *Resolution*. For example, one such case was a row where the field *Resolution* was "*ARREST, BOOKED*". We noticed that MySQL eliminates quotes, so the field "*ARREST, BOOKED*" will appear as *ARREST, BOOKED*. However, in the MapReduce jobs, often we will need to split the row on the comma; to avoid malformed splits, we must keep the air quotes in. The query is the following:

```
UPDATE crimes c SET c.descript = CONCAT('\"', c.descript, '\"'), c.resolution =
CONCAT('\"', c.resolution, '\"');
```

Now, the dataset is ready to be moved from MySql to HDFS. For this purpose the command `sqoop-import` has been executed.

```
sqoop-import --connect jdbc:mysql://master/crimes --username hive -P --table
crimes -m 1
```

Note that the option `-m 1` is necessary due to the absence of a primary key in the dataset.

The dataset now is ready on the HDFS for MapReduce Jobs. After the dataset is cleaned with the corresponding MapReduce cleaning Job, an external table on hive has been created as follow:

```
CREATE EXTERNAL TABLE crimes(crimedate DATE, category STRING, description STRING,
dayoftheweek STRING, district STRING, resolution STRING, address STRING, longitude
FLOAT, latitude FLOAT, timeoftheday STRING, month STRING, year INT) ROW FORMAT
SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' LOCATION
'/user/hadoop/cleandata';
```

Data Understanding

The dataset is structured in 878048 rows and 12 columns. The columns are named as follows:

- *Dates* - timestamp of the crime incident
- **Category** - category of the crime incident (the target variable)
- *Descript* - detailed description of the crime incident
- *DayOfWeek* - the day of the week
- *PdDistrict* - name of the Police Department District
- *Resolution* - how the crime incident was resolved
- *Address* - the approximate street address of the crime incident
- *X* - Longitude
- *Y* - Latitude

X and *Y* are the only numerical columns, *Dates* is a date column and all others are categorical columns.

Data cleaning and transformation

Before trying to understand the distribution of data by making statistics, we had to clean the dataset from null values and malformed rows.

Cleaning and feature engineering job

The job first takes the data from the `crimes` folder inside the HDFS the data and output the cleaned result in a new folder `cleandata` on HDFS.

Mapper

For each row of the dataset, these operations are performed:

1. Stop-word removal from address column (e.g. "OAK ST / LAGUNA ST" to "OAK / LAGUNA"). Stopwords include "LN", "AVE", "ST", "Block", "of".
2. Creation of a categorical column called "TimeOfTheDay", which varies according to the hour in which the crime occurs. i.e., hours in range 5 and 11 become "Morning", in range from 12 to 17 become "Afternoon", etc.
3. Typo correction on the field *Category* ("TREA" is the same as "TRESPASS", most likely this was due to a typo in the dataset)
4. Creation of two new fields *Month* and *Year* after the split of column *Date*

The mapper writes to the context the district as key and as value the whole row. This is because part of the cleaning process happens in the reducer as well.

Reducer

Indeed, some values in the latitude column were in the range of 90 and 93, which are obviously wrong (90 is the latitude of the North Pole). So, for each district, the reducer computes the mean values of both latitude and longitude, and in case it finds an outlier, it substitutes that value with the mean for that district. So for example, if a "Robbery" crime happened in the "Mission" district at a latitude of 92, and the mean for the district is 82, then the new value will have 82 as latitude instead of 92. The output of the reducer is the completely cleaned dataset.

Data analysis

In this step, we extracted some statistics from the dataset to identify trends, highest values for different columns, etc. This was performed by two separate (but chained) jobs.

First Job

Objective: obtaining the top-k occurences of a given column.

The first job takes as input the cleaned data from the previous MapReduce Job. This job works as a word count: the mapper just writes on the context the value of the column as key, and 1 as value, the reducers sum up all these values received as value, keeping track only of the top-k occurrences in a TreeMap. The output of this job (of the form "column occurrence, number of occurrences") will be in a new folder called "top-k_out".

Second job

This job is chained to the first, and its objective is to know the distribution of one column w.r.t. another column. For example, one might be interested in knowing how the rate of each crimes (or, in our case, of the top-k crimes) increases or decreases year-by-year. Note that the jobs are completely parametric. They can be used to plot the distribution of any column against any other column (district by year, crimes by address, etc.), just by modifying the `featureIndex` and `parameterIndex` in `StatisticsJob.java`. Also, the k is a parameter, so the jobs allow to compute statistics for any given k.

Mapper

Firstly, the mapper reads the output of the first job, as it needs to know which are the top-k occurring values for a given column. Continuing from the last example, it needs to know which are the top-k occurring crimes, to know their distribution against time. Then, as a second input, it reads again the cleaned dataset, and considers only the rows where one of the top-k occurring values appears. Concluding the example, the mapper will ultimately output all pairs with ("type of crime", "year in which it happened") as **key** and 1 as **value**.

Reducer

The reducer simply aggregates by both columns and sums up the occurrences, similarly to what a word count would do. Following again from the previous example, the output will be of the form ("type of crime", "year in which it happened", "how many times it occurred").

Validation of the output

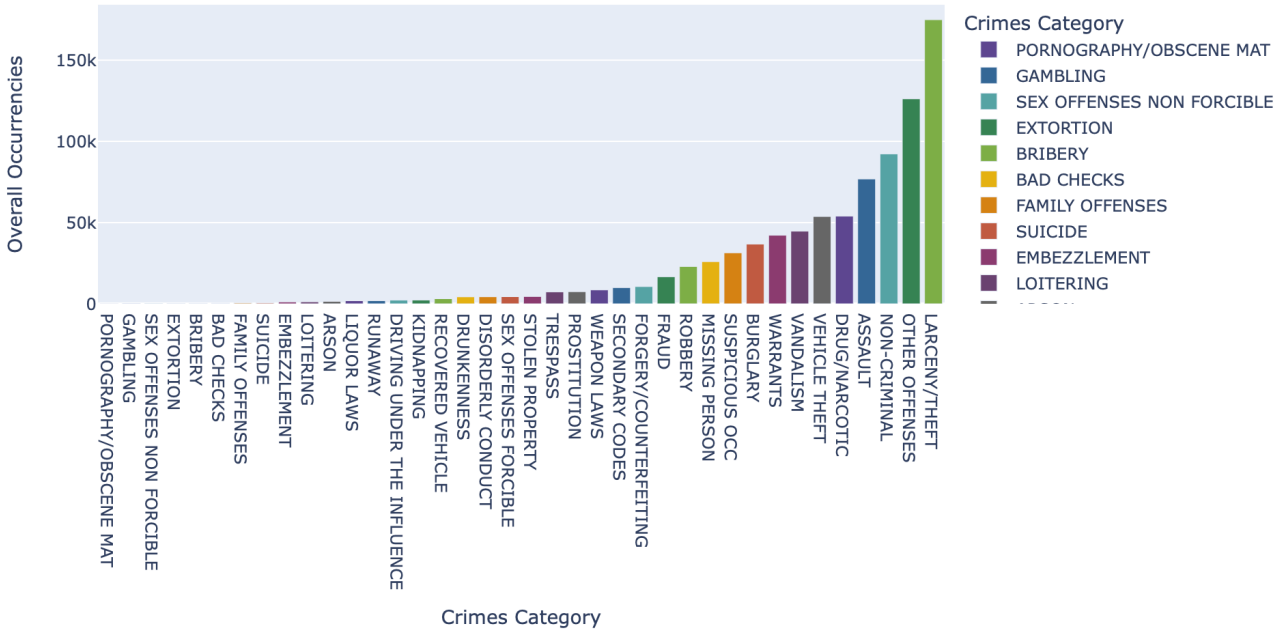
To debug and verify correctness of these jobs, for each of the analyses we have performed hive queries and compared results. These are reported below.

Analyses performed

To plot the results we have used the *plotly** library for Python, which provides **interactive plots** that allow to filter information on-the-fly **just by hovering and/or clicking the mouse cursor**. These can be viewed by running the Jupyter notebook provided in the repo using the instructions provided below.

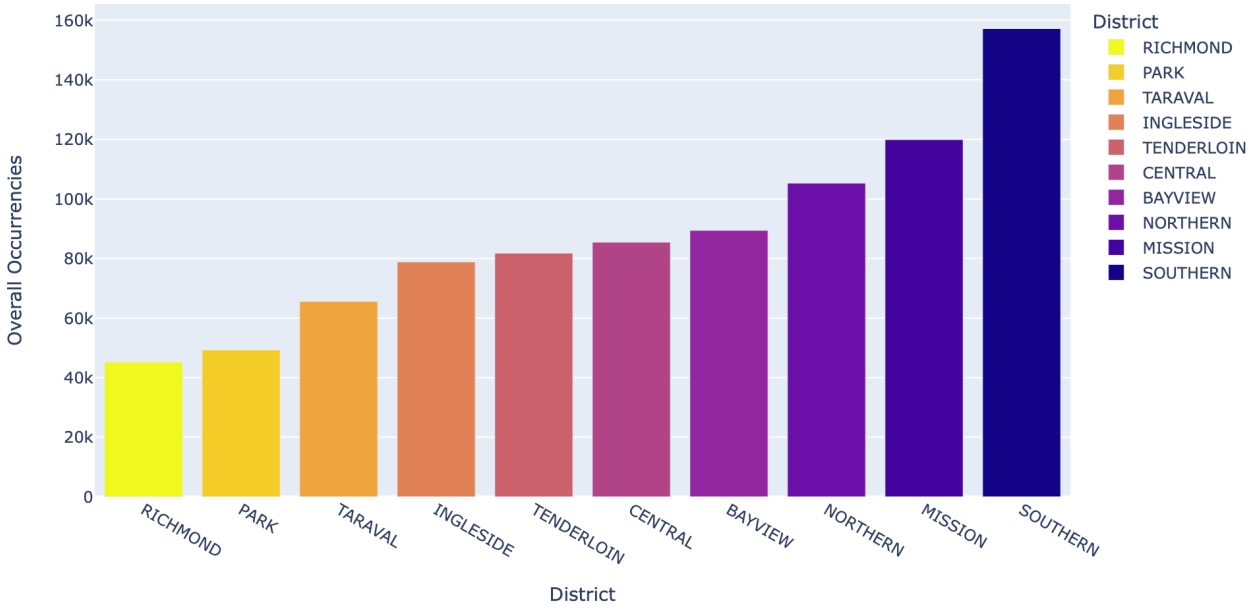
How many times each crime occurs

We can see that larceny/theft is the top category, immediately followed by other offences. All the other categories do not reach >100k occurrences, so the first two categories are dominant by quite a large margin. Hive query: `SELECT COUNT(*), c.category FROM crimes c GROUP BY c.category ORDER BY COUNT(*) DESC;`



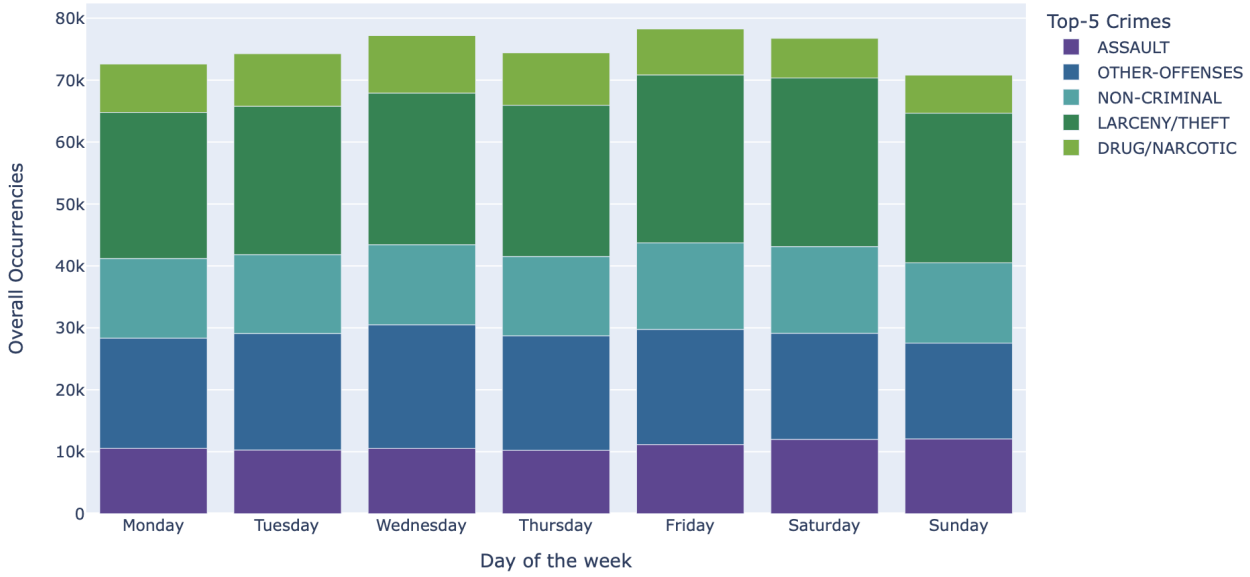
How many times a crime is registered in each district

The distribution here is slightly less "spiked", that is, there is no single district that trumps all the other ones in terms of occurrences. The middle five districts (Ingleside, Tenderloin, Central, Bayview, Northern) belong to the 80k-100k range and the highest ranking is Southern. Hive query: `SELECT COUNT(*), c.district FROM crimes c GROUP BY c.district ORDER BY COUNT(*) DESC;`



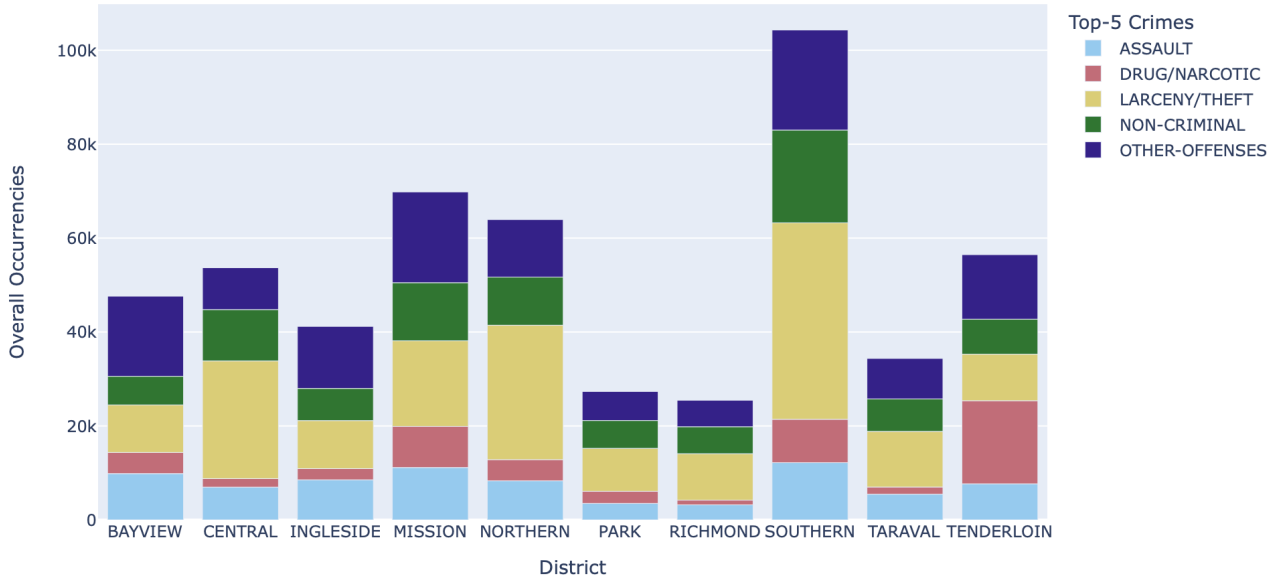
Extract the top-5 occurring crimes, and plot their distribution by day of the week

The distribution is quite uniform, with the highest peak being on friday and the lowest peak being on sunday. No substantial variation during the weekend. Hive query: `SELECT COUNT(*), c.category, c.dayoftheweek FROM crimes c GROUP BY c.category, c.dayoftheweek ORDER BY c.category ASC, c.dayoftheweek ASC;`



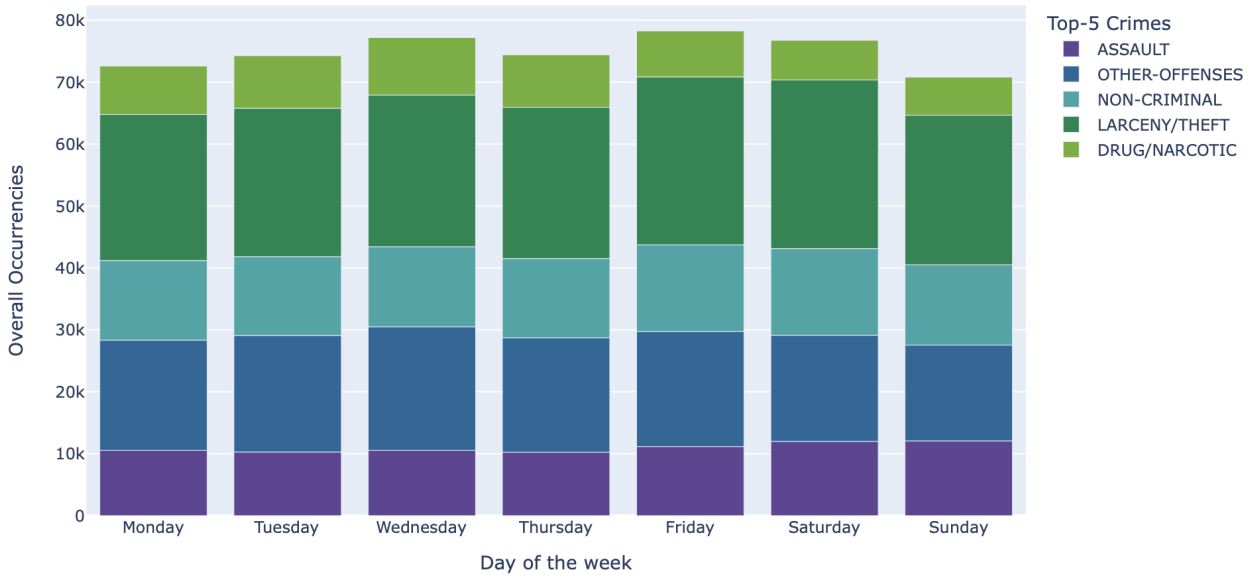
Extract the top-5 occurring crimes, and plot their distribution by district

Southern is still confirmed to be the most criminal district, and theft the most frequent category. Hive query: `SELECT COUNT(*), c.category, c.district FROM crimes c GROUP BY c.category, c.district ORDER BY c.category ASC, c.district ASC;`



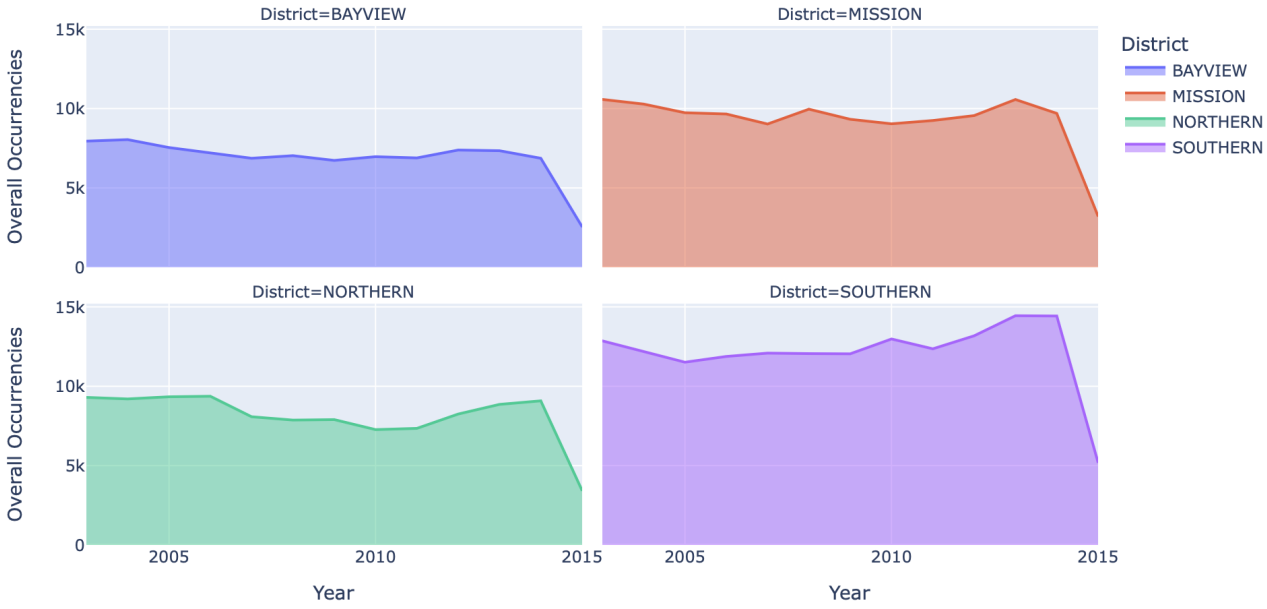
Extract the top-5 most criminal districts, and plot their crime rate by time of the day

This plot just confirms the above statistics. Hive query: `SELECT COUNT(*), c.district , c.dayperiod FROM crimes c GROUP BY c.district , c.dayperiod ORDER BY c.district ASC, c.dayperiod ASC;`



Extract the top-4 most criminal districts, and plot their crime rate by year

We see quite a uniform distribution on each of the districts, and a decrease in crime rate towards 2015. Bayview is the most uniformly distributed, and Southern has seen a 2-3k increase between 2010 and 2014. Northern is the lowest of the four in terms of occurrences. Hive query: `SELECT COUNT(*), c.district , c.\`year\` FROM crimes c GROUP BY c.district , c.\`year\` ORDER BY c.district ASC, c.\`year\` ASC;`



Modeling

Steps needed to run the model

- Download this script at link on the master machine;
- Execute `sh Miniconda3-py38_4.10.3-Linux-x86_64.sh` ;
- Follow the instructions that appears on the screen;
- After the setup is finished, execute the command `conda create -y -n pyspark_env -c conda-forge pyarrow pandas conda-pack`
  - note that you must run this command while being in the base conda environment
- Then execute `conda activate pyspark_env`
- Export the environment in an archive with `conda pack -f -o pyspark_env.tar.gz` (in the same directory where is placed the Jupyter Notebook);
- After that, install Jupyter, Seaborn, Plotly, PySpark with the command:
  - `conda install jupyter, plotly, pyspark, seaborn`
- Start Jupyter Notebook
- Open the notebook 'Data Analytics and Modeling.ipynb'

The modelling part has been perfomed on a Jupyter Notebook using Spark on the cluster, Python language and SparkMLlib library. Initially the dataset is loaded from an **external** Hive table to a dataframe. Different models has been used for the prediction task: **Random Forest Classifier**, **Naive Bayes Classifier** and **Multinomial Logistic Regression**. The dataset needs to be prepared before its use for the model, so these operations were performed:

- Removal of column *Crimedate*, since redundant because of the presence of similar columns such as *Timeoftheday*, *Month* and *Year*;
- Convertion of data type to their properly ones, i.e., *Longitude* and *Latitude* on float;
- Indexing of column *Category* to convert from a string format to a proper one for the prediction task.

Different configurations have been applied to the models trying to achieve an high percentage of accuracy. All of the different categorical columns, before being trained, have been transformed using the function OneHotEncoder. Then one last step was to split the dataset into a train set (70% of the dataset) and a test set (30% of the dataset).

Random Forest Classifier

Starting from the previous preparation of the dataset the model has been trained several times with different feature sets. The following has been perfomed (accuracy refers to test accuracy, train accuracy is in the Jupyter notebook):

- Removal of *Description* and *Address* columns: 22% of accuracy achieved;
- Removal of *Address* columns: 40% of accuracy;
- Removal of *Latitude*, *Longitude*, *Address* columns: 44% of accuracy;
- Removal of *Latitude*, *Longitude* and *Dayoftheweek* columns: 39.77% of accuracy.

Naive Bayes Classifier

The Naive Bayes Classifier has been perfomed with the following configuration:

- Removal of *Address*, *Longitude*, *Latitude*, *Description* columns: 22% of accuracy;
- Removal of *Address*, *Longitude*, *Latitude* columns: 99.5% of accuracy achieved.

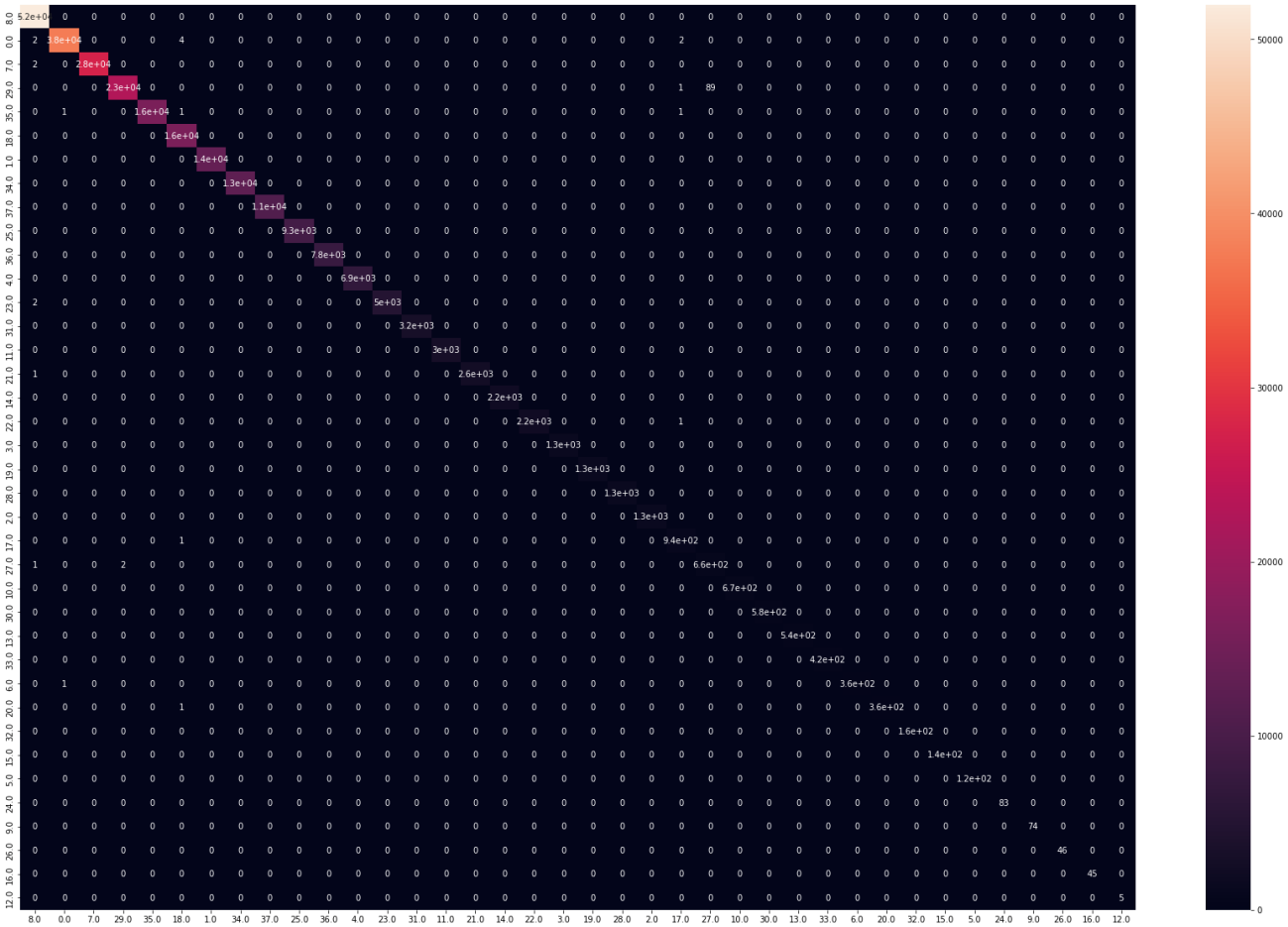
Multinomial Logistic Regression

This model has been used with following configuration:

- Removal of *Address*, *Latitude*, *Longitude* columns: 99.9% of accuracy;

Confusion Matrix

The Multinomial Logistic Regression has given the best result as accuracy metrics. Starting by this point a confusion matrix has been constructed with as row the *Category* and as column the predicted values. The figure below is a representation of the confusion matrix that has been described.



Final considerations

The description column proved to be very relevant for a good model, as it is the most valuable piece of information of a crime. Indeed, when removing the column in the Naive Bayes model, the test accuracy dropped from 99.5% to 22%. Also, when doing so in the Random Forest classifier, accuracy dropped from 40% to 22%.