*Nom de la team : RSA*

*Sprint n°2 - Projet N°6 Criminalité à Chicago*

*Scrum Master : Sacha Cymermann*

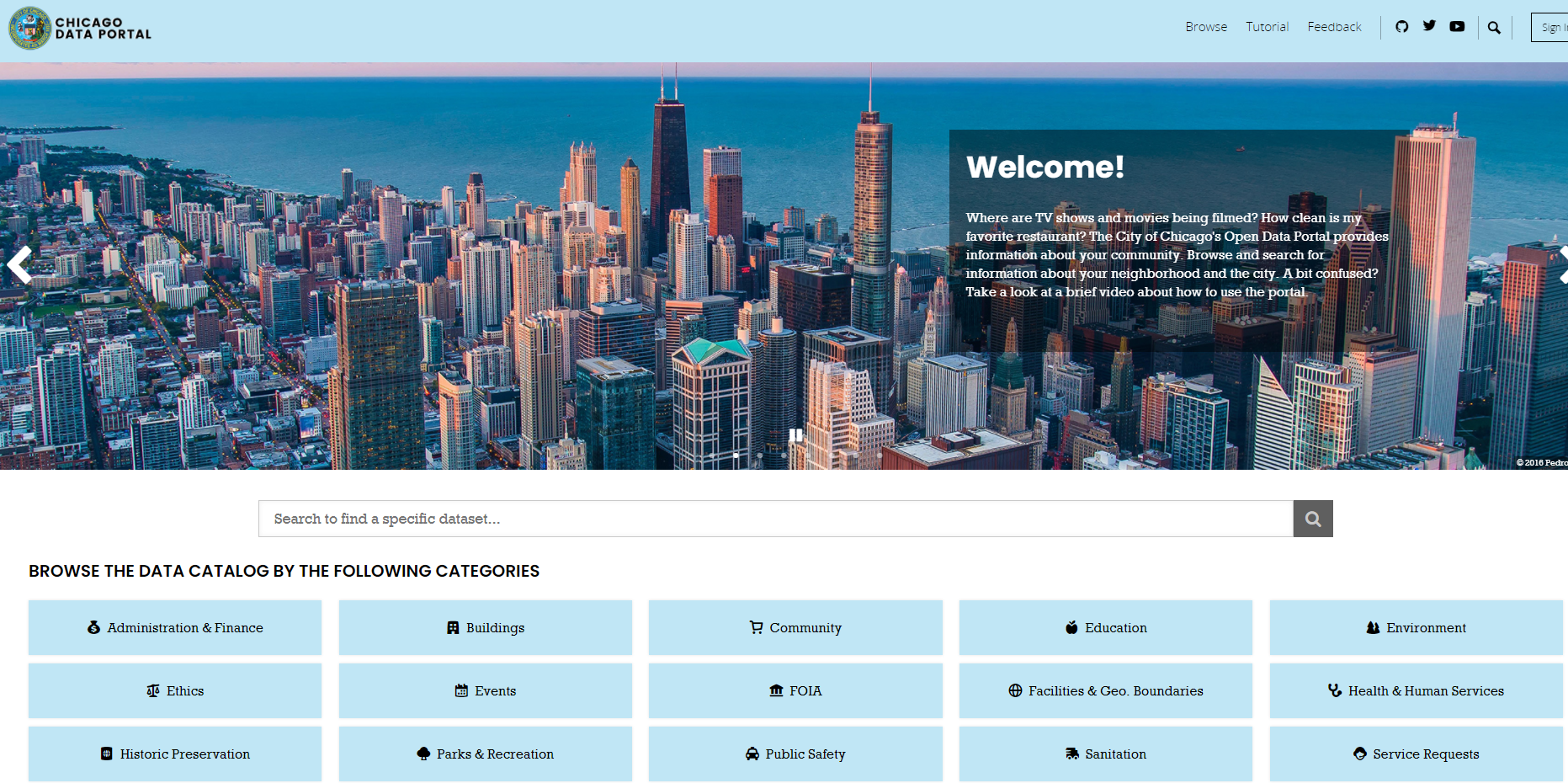
*Teammates : Adrien Golebiewski, Ryme Youb*

*Début : 10/01/2022*

*Fin : 24/01/2022*

**Sprint 2 - Familiarisation de Spark, création de nos indicateurs avec PySpark et Prédiction du type de crimes avec SparkMlib**

**Rappel du Projet et des avancements du Sprint 1**



Nous avons choisi de traiter cette problématique de criminalité

actuelle en étudiant les données fournies par le “Chicago Data

Portal”, portail de données en OPEN DATA par la ville de Chicago (<https://data.cityofchicago.org/>). Notre objectif est double :

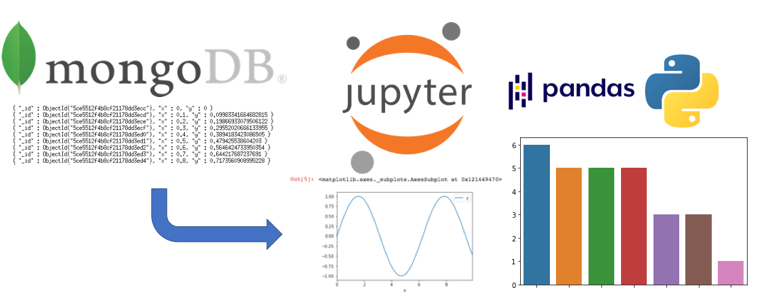
créer des indicateurs expliquant la criminalité et prédire le type de crime.

A l’issue du Sprint T1, plusieurs étapes ont été réalisées, notamment :

* Lancement de requêtes Map Reduce en local depuis une base Mongo DB
* Accès au serveur du CNAM et import de l'écosystème de notre projet dans la machine Mongo DB

Les problèmes externes au projet et la faible durée du sprint nous a empêché de finaliser notre Sprint T1 convenablement avec une démonstration du Stockage des données dans l'écosystème Hadoop.

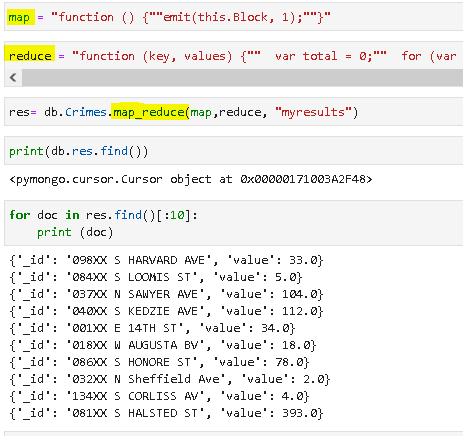
Cependant, en stockant les données depuis la machine Mongo DB du serveur du CNAM et en les requêtant à l’aide de la librairie “PyMongo” Python et du client Compass, nous avons réussi à prototyper la création d’un de nos indicateurs à l’aide de deux différentes manières.



* Une méthode en utilisant les requêtes classiques d'agrégation de la librairie “PyMongo”



* Une méthode en utilisant la requête spécifique “map reduce” de la librairie PyMongo



**Stratégie et Familiarisation avec Spark**

Le grand volume de données associé à la criminalité à Chicago (environ plus de 1,62 Go = 4M de crimes) a été un argument dans l’utilisation d’outils Big Data afin de répondre à nos objectifs.

Parmis ces outils, la technologie “Spark” est l’une des seules qui répond à la fois à :

* la création d’indicateurs de visualisation et de data frame pour la compréhension des données de criminalité (technologie nouvelle de RDD - langage PySpark)
* la prédiction du type de crime à partir de variables explicatives préalablement définies (librairie SparkMlib)

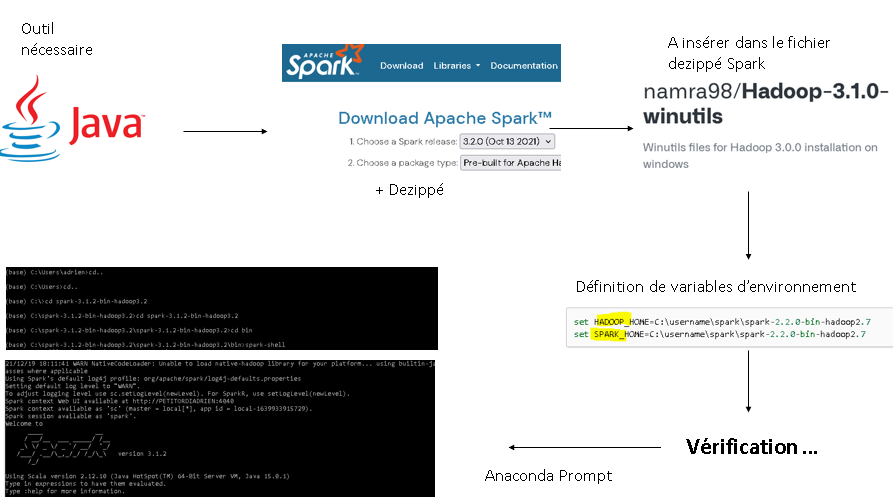
Nous avons ainsi eu le choix d’utiliser soit la machine Spark du serveur récemment crée du CNAM ou construire notre propre machine Spark.

Envisagé d’utiliser la machine Spark du serveur du CNAM impliquait pour nous de réaliser l’ensemble de nos démarches techniques sur un des ordinateurs nous empêchant de travailler à distance pour avancer plus vite…

De plus, l’incapacité d’utiliser la suite Anaconda sur les ordinateurs du CNAM nous a confortés dans notre choix de monter nous même une machine Spark, en local sir nos propres ordinateurs.

Une phase de familiarisation de l’outil a donc constitué la 1ère phase du sprint 2. Cette **familiarisation** s’est déroulée en 3 phases :

* **Installation de l’outil**

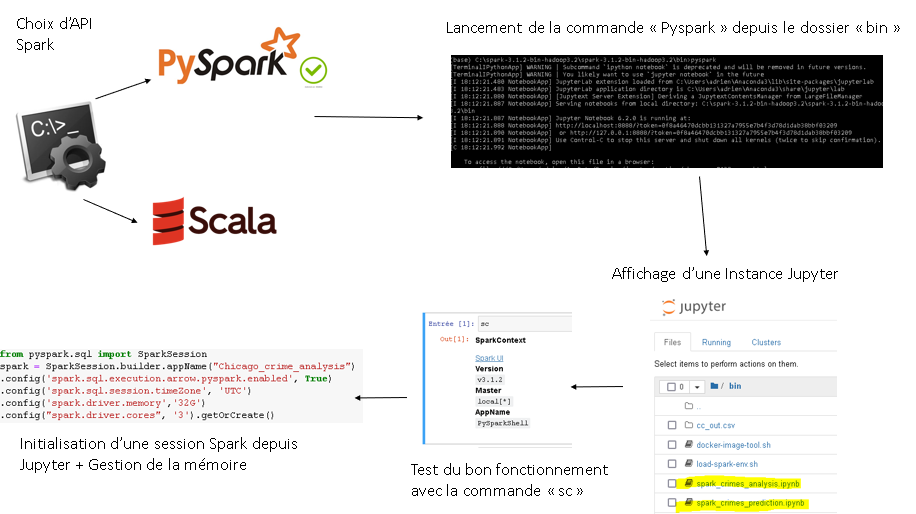


La commande spark-shell dans l’invité de commande et positionné dans le dossier “bin” nous permet de savoir si une instanciation Spark est bien installée sur notre machine.

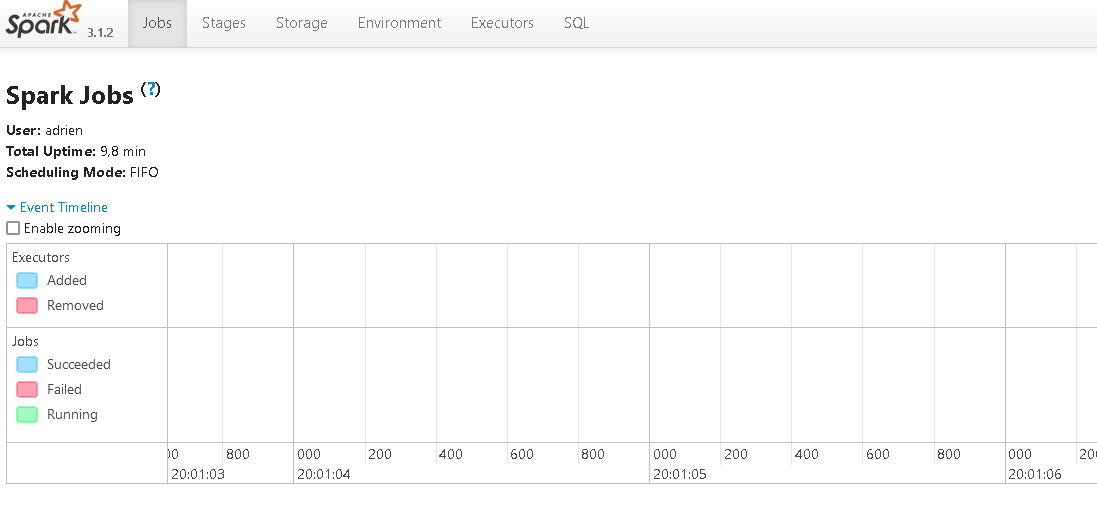
* **Paramétrage de l’instanciation PySpark depuis Jupyter**

Depuis l’invité de commande, nous avons le choix de lancer soit **PySpark** ou soit Scala, deux Api, langages de la technologie Spark. Scala est reconnu pour être plus rapide mais notre intérêt s’est tourné vers PySpark , un langage dérivé de Python et surtout contenant des librairies de Machine Learning (contrairement à Scala) pour la réalisation de nos prédictions finales.

La commande “Pyspark” dans l’invité nous renvoie vers une instance Jupyter (car une variable d’environnement "ipython-notebook" avait été préalablement définie) qui constitue notre environnement de travail avec nos codes python et nos données.



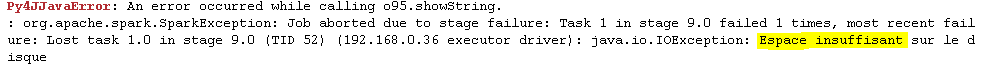
Dans nos codes Python, l’expression “sc” nous permet de tester si Jupyter Notebook et de manière plus générale Python a bien été instancié avec Spark. Une autre vérification est de lancer sur notre navigateur web : <http://localhost:4040/jobs/> qui renvoie l’ensemble l’état de l’application Spark et de ces instanciations sur la machine locale.



La dernière étape est de lancer la première session Spark Python. Il s’agit d’un point d'entrée vers Spark pour travailler avec l’ensemble de ses technologies :RDD, DataFrame. L'objet de SparkSession *spark* est disponible par défaut dans spark-shell et il peut être créé par programmation à l'aide du SparkSession, il s’agit d’un générateur.

Parmi les méthodes de ce générateur, celle qui nous intéresse le plus est celle concernant la gestion de la mémoire. En effet, en local, Spark est très gourmand en mémoire RAM. La méthode “driver memory” définit la quantité de mémoire allouée à utiliser pour le processus du pilote nous permet de :

Cette valeur varie selon la capacité de stockage et de mémoire de l’ordinateur. Ne pas mettre assez de mémoire allouée empêchera de lancer certaines requêtes sur nos millions de données. A l’inverse, définir trop de mémoire allouée (par rapport à celle de notre machine en locale) nous renverra également une erreur “out of memory”

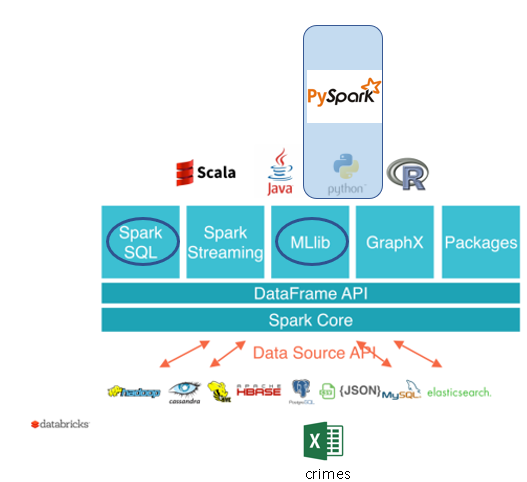


Dans le cadre de notre étude, le meilleur choix s’est porté sur une mémoire de 32 Go, juste équilibre nous permettant de lancer l’ensemble de nos requêtes souhaitées et soutenant convenablement la machine en locale.

* **Premier tests et nouveautés**

Une phase de découverte nous a également permis de nous familiariser avec des technologies propres à Spark. Spark a été développé en Scala et est au meilleur de ses capacités dans son langage natif. Cependant, la librairie PySpark propose de l’utiliser avec le langage Python, en gardant des performances similaires à des implémentations en Scala.

Pyspark est donc une bonne alternative à la librairie pandas lorsqu’on cherche à traiter des jeux de données trop volumineux qui entraînent des calculs trop chronophages.



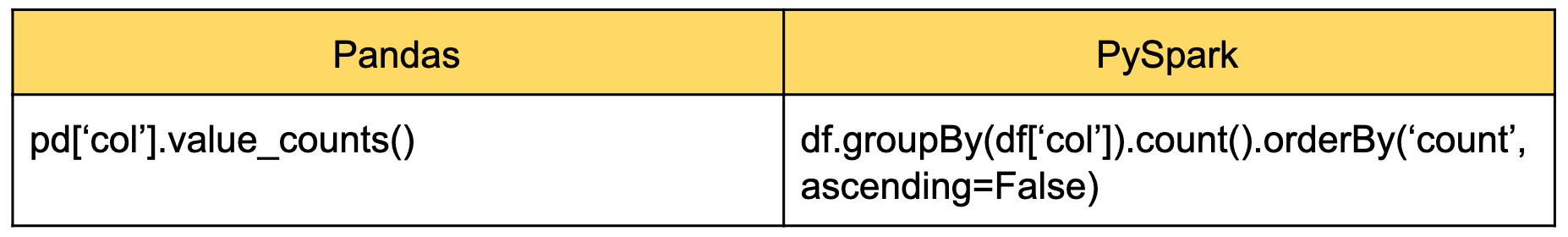
L’une des structures les plus élémentaires de SPark est le RDD. Un RDD est la représentation Spark d’un **tableau de données**. C’est une collection d’éléments que l’on peut utiliser pour contenir des tuples, des dictionnaires, des listes…

La force d’un RDD réside dans sa capacité à évaluer le code de façon paresseuse : le lancement des calculs est reporté jusqu’à ce que ce soit absolument nécessaire.

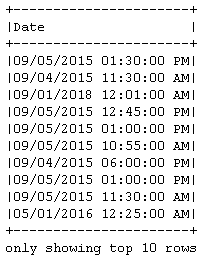
Par exemple, lorsque l’on importe un fichier, seul un pointeur vers ce dernier est créé. Ce n’est vraiment qu’au dernier moment, lorsqu’on recherche à afficher ou utiliser un résultat, que le calcul est effectué.

**L’API DataFrame** de pyspark découle du RDD. Il est la **structure la plus optimisée en Machine Learning**. Elle utilise de façon sous-jacente les bases d’un RDD mais a été structurée en colonnes autant qu’en lignes dans une structure SQL. Sa forme est inspirée des DataFrame du module pandas.

**Spark SQL** est un module de PySpark qui permet de travailler sur de la **donnée structurée**. C’est donc au sein de ce module qu’a été développé le Spark DataFrame.



Ci-dessous un exemple d’un RDD de type Data Frame regroupant les 10 premières dates de notre BDD de crimes.



Il est cependant tout à fait possible de passer du format “data frame Spark” au format plus traditionnel “Pandas” à l’aide de la fonction “.toPandas()”.

De manière générale, le framework PySpark traite de grandes quantités de données beaucoup plus rapidement que les autres frameworks conventionnels tels que Python de la distribution Anaconda.

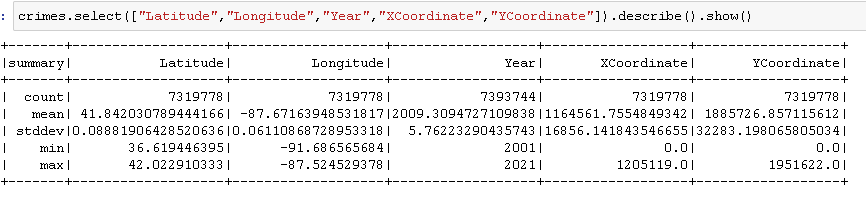
**Exploration et création de nos indicateurs avec PySpark**

Lors des deux précédents sprint, nous avons résumé la complexité des crimes de Chicago à la fois pour les aspects temporels (évolution du nombre de crimes), géographiques (nombre de crimes par area) et structurels à travers différents indicateurs.

L’objectif de ce sprint est de réaliser la même chose en utilisant la librairie SparkSQL de Pyspark et de tester l’ensemble des fonctions de la librairie pour aller encore plus loin en termes d’indicateurs..

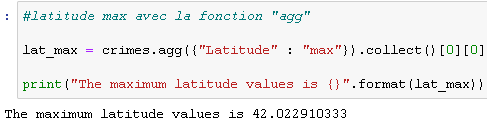
Dans un premier temps, nous avons ainsi exploré les données au moyen de plusieurs fonctions de base de la librairie SparkSQL et plusieurs statistiques :

* select (data frame en entier), filter (avec conditions), where (avec conditions)

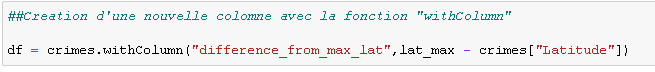




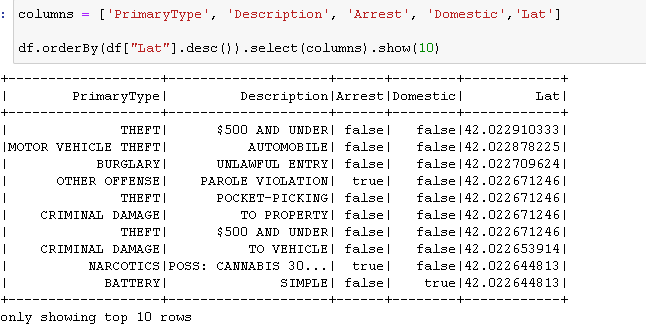
* Aggregation max/min avec la fonction “agg”



* Ajout d’une colonne avec la fonction “withcolumn”



* Group By, Order by



Par ordre de latitude décroissant, liste des 10 premiers crimes, de leur Primary Type et des attributs Description, Arrest et Domestic.

D’autres statistiques (max, min, corrélation …) ont également été réalisées sur les données à titre indicatif.

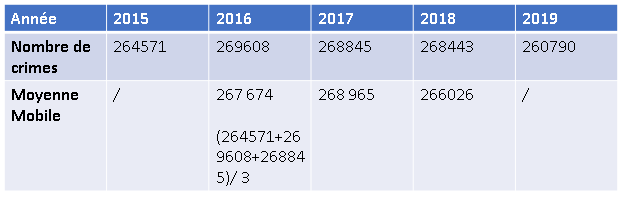
Parmis ces statistiques, nous avons construit une requête calculant **la moyenne mobile du nombre de crimes par année.**

Une moyenne mobile permet de lisser une série de valeurs exprimées en fonction du temps (série chronologique) **en prenant en compte la valeur de la mesure n-1 et n+1**. Elle permet donc d'éliminer les fluctuations les moins significatives et mettre en avant des tendances sur le long terme.

Dans le cadre de notre étude, nous avons ainsi appliqué la fonction “avg” de Pyspark pour le calcul de moyenne mais en ajoutant à la suite une fonction over avec comme paramètre “Partition By” = “id” afin de partitionner notre moyenne par année. La combinaison de ces 2 fonctions nous permet de calculer la moyenne mobile du nombre de crimes par année.

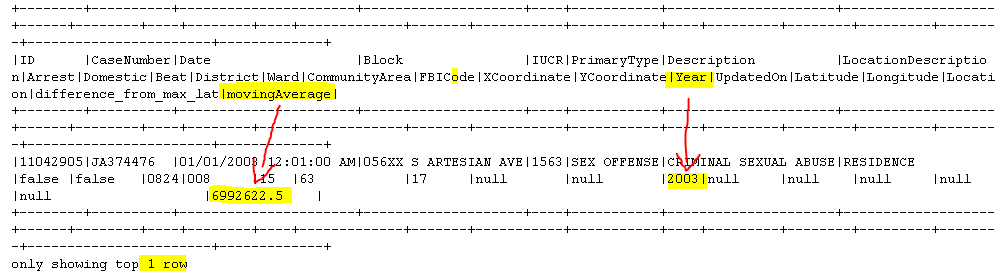


En retour, nous récupérons un data frame avec l’ensemble des moyennes par année, une moyenne lissée comme ci-dessous de 2015 à 2019.



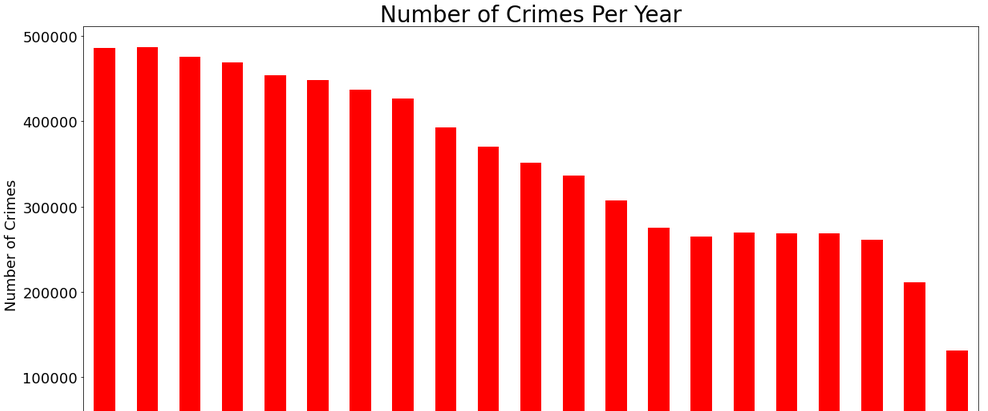
D’un point de vue algorithmique, nous avons appelé le code précédent pour la première année de notre dataset (2003) et nous récupérons en retour en format RDD le “moving average” correspondant à la moyenne mobile et l’année correspondant. Pour avoir plusieurs années affichées, on peut paramétrer la fonction à notre guise :





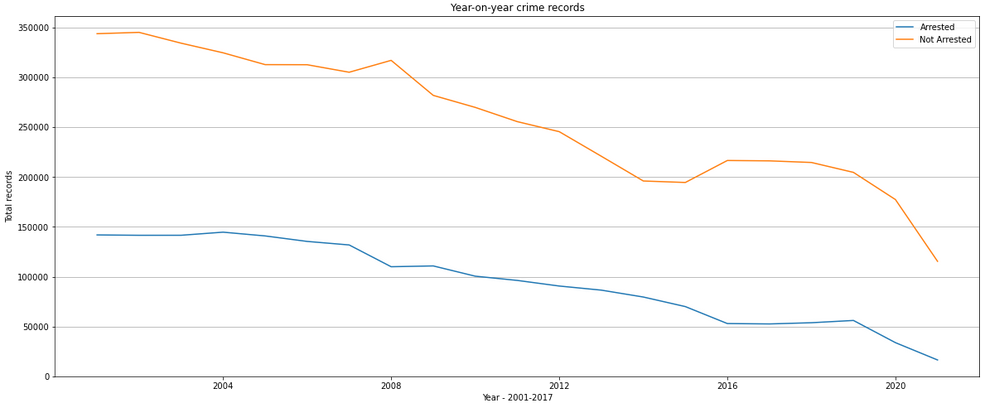
Enfin sur la partie analytics, création de :

* **l’indicateur du nombre de crimes par an** (déjà calculé à l’aide de la librairie PyMongo lors du Sprint 0/1) par le graphique en **histogramme**.

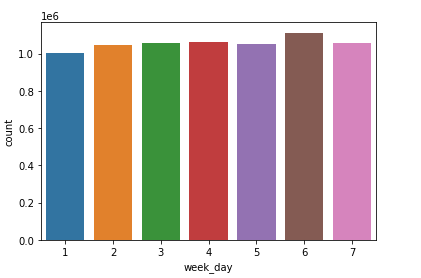
****

On observe une baisse du nombre de crimes par an entre 2001 et 2021. Mais ce nombre de crimes reste élevé et la baisse est ralentie depuis 2015.

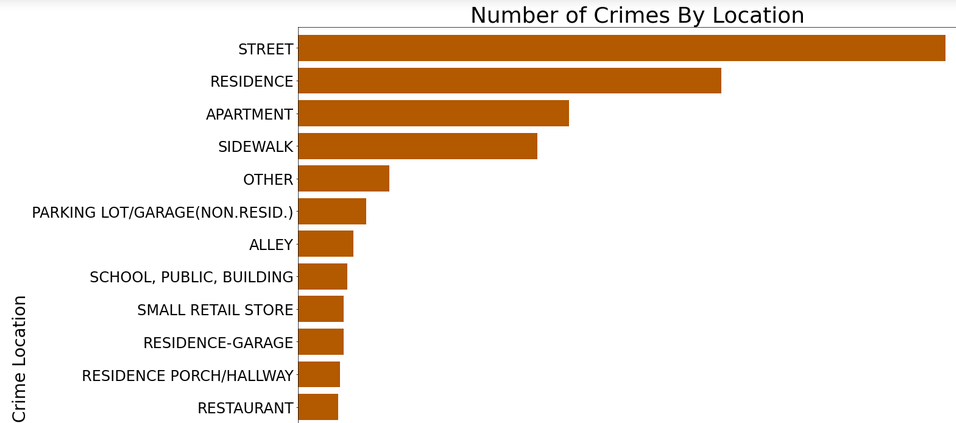
* **l’indicateur du nombre de crimes par an** par un graphique linéaire qui différencie deux types de crimes : crimes étant réalisé avant un arrestation et ceux sans arrestation par la suite. On remarque que le nombre de crimes n’ayant pas abouti à une arrestation sont plus nombreux.



* **Indicateur** du nombre de crimes par mois. Nombre de crimes plus élevé le samedi. Moins de crimes en début de semaine (surtout le lundi).



* **l’indicateur du nombre de crimes par “Location”** par un histogramme (déjà calculé à l’aide de la librairie PyMongo lors du Sprint 0/1)

****

Dans le cadre de ce sprint 2, cette phase d’exploration a été primordiale afin de maîtriser l’API Pyspark et ses fonctions et d’explorer plus en profondeur nos données de criminalité.

**Prédiction du type de crimes - SparkMlib**

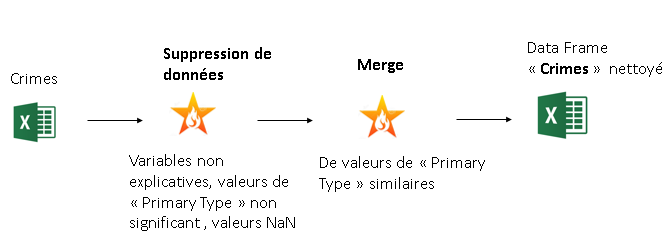
Le second objectif final du projet concerne la prédiction du type de crime “Primary Type” (vol, agression, etc..) en fonction d’un certain nombre de variables explicatives à déterminer. Cette seconde partie de ce sprint a impliqué plusieurs phases importantes :

Data Preparation et Features Engineering

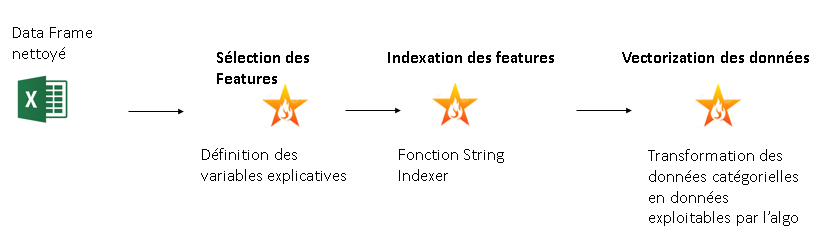
Avant de procéder à la modélisation des données, il est nécessaire de passer par un nombre d’étapes préliminaires qui sont essentielles à un projet de **machine Learning**. Bien que fastidieuses, il ne faut surtout pas les mettre de côté car elles sont garantes de la qualité de notre modélisation. Un modèle dont la qualité des données n’est pas optimale, ne pourra pas réaliser de bonnes prédictions.

Les étapes préliminaires, aussi appelées étapes de **data pre-processing**, recouvrent ainsi les étapes de vérification de l’état, du nettoyage – **data cleaning** – et de la préparation de données.

Cette étape s’est réalisée en 2 étapes : suppression des données inutiles (variables non explicatives, données NaN manquantes) et une gestion des données “Primary Type”, variable cible plus précise : suppression des valeurs de la variable “Primary Type” non significant et merge des données Primary Type similaires en 1 seul valeur.



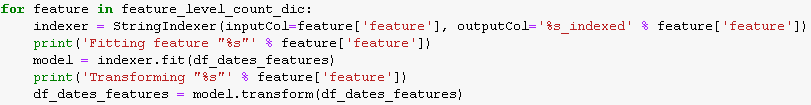
La deuxième phase consistait à faire du feature Engineering pour préparer nos variables à la construction de modèles.



Dans un premier temps, nous avons sélectionné manuellement l’ensemble de nos variables explicatives (à la main) parmi nos variables du data set nettoyé.



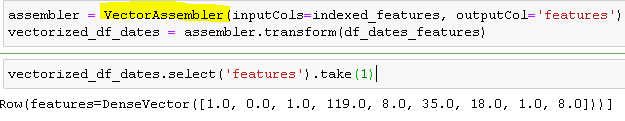
Puis nous avons appliqué, à travers une boucle “for” la fonction PySpark “StringIndexer” sur les features sélectionnées.



StringIndexer encode une colonne d'étiquettes en une colonne d'indices d'étiquettes. StringIndexer peut encoder plusieurs colonnes et les indexer.

On lui associe un label à un index pour l'ordonnancer.

Enfin, nous vectorizons l’ensemble de nos features indexés à l’aide de la fonction “VectorAssembler”.



Elle nous permet de convertir des caractéristiques numériques en un seul vecteur utilisé par les modèles d'apprentissage automatique. Elle prend en entrée notre liste de colonnes indexed\_features et la combine en une seule colonne vectorielle (vecteur de caractéristiques). Cette numérisation de colonne sera notre entrée dans les modèles d'apprentissage automatique dans Spark ML.

Ainsi à l’issue de cette phase de features Engineering, notre data set de crimes est prêt et bien organisé pour pouvoir définir notre stratégie d’apprentissage.

Stratégie Apprentissage

Comme nous l’avons vu en cours, l’entraînement d’un modèle de prédiction revient à mesurer l’erreur de la sortie de l’algorithme avec les données d’exemple et chercher à la minimiser.

Un premier piège à éviter est d'évaluer la qualité de votre modèle final à l'aide des mêmes données qui ont servi pour l'entraînement. En effet, le modèle est complètement optimisé pour les données à l'aide desquelles il a été créé. L'erreur sera précisément minimum sur ces données. Alors que l'erreur sera toujours plus élevée sur des données que le modèle n'aura jamais vues.

Pour minimiser ce problème de sur-apprentissage, la meilleure approche est de séparer **dès le départ** notre jeu de données en deux parties distinctes :

* Le **training set**, qui va nous permettre d’entraîner notre modèle et sera utilisé par l’algorithme d’apprentissage. C'est celui dont on a parlé depuis le début.

* Le **testing set**, qui permet de mesurer l’erreur du modèle final sur des données qu’il n’a jamais vues. On va simplement passer ces données comme s'il s'agissait de données que l’on n'a encore jamais rencontrées (comme cela va se passer ensuite en pratique pour prédire de nouvelles données) et mesurer la performance de notre modèle sur ces données.

C'est à nous de définir la proportion du dataset que vous souhaitez allouer à chaque partie.

En général, les données sont séparées avec les proportions suivantes : **60 % pour le training set et 40 % pour le testing set.** à l’aide de la librairie **“randomSPlit”** de la librairie **SparkMlib.**



Par la suite nous avons défini 4 modèles d’apprentissage à appliquer et entraîner sur nos données :

* regression logistique
* Random Forest
* Bayes
* MLP (Réseaux de neurones classiques)

Ces 3 modèles ont été choisis car ils nous sont familiers et ont été testés au cours de nos 3 ans d’étude au CNAM à travers d’autres projets.

Résultats :

La métrique d’évaluation de nos modèles est l’**accuracy/précision**.

Comme avec les librairies classiques de Python “keras” ou “sckit learn”, on appelle des fonctions spécifiques à la librairie SparkLib :

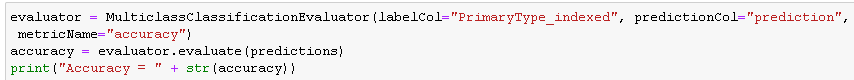
* **la fonction SparKlib “fit”** pour faire apprendre notre modèle sur les données de crimes nettoyées d’apprentisssage



* **la fonction Sparklib “transform”** afin de prédire notre “Primary Type” sur notre ensemble test. On précise ainsi les variables que nous souhaitons récupérer : notre “Primary\_type”, la prédiction du modèle et enfin la probabilité de bonne prédiction ou non (cas du classifieur Bayes);



* la fonction Sparklib “MulticlassClassificationEvaluator” fonction classique globale de la librairie qui prédit notre “Primary Type” sur notre ensemble test. Spécifique sur une feature à plusieurs classes :



En lançant l’ensemble de ces modèles d’apprentissage à l’aide de ces fonctions, nous obtenons les résultats suivants :



Ces modèles ont été lancés sans recherche de paramètres optimisés pour chacun des algorithmes. On remarque que les résultats de précision sont insuffisants, voire très insuffisants notamment pour le classifieur Bayes. Seul l’algorithme du MLP nous offre un résultat convenable.

Commentaires et axes d’amélioration

Ces résultats sont largement insuffisants et nous ont permis de réfléchir sur des axes d’améliorations possibles. Nous pouvons proposer quelques pistes :

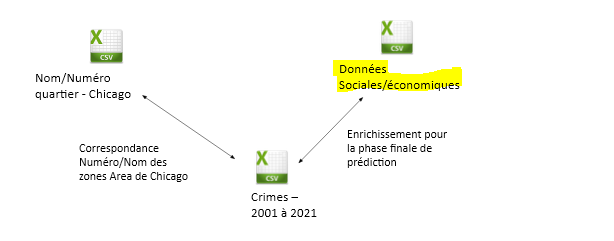
* **Meilleur choix des variables explicatives**

Les variables explicatives intégrées dans notre modèle ont été choisies à la main selon notre connaissance de la base de crimes. Le choix de ces variables aurait pu être optimisé et donc plus pertinent en utilisant des techniques de “selection features” pour chacun de nos modèles d’apprentissage. Par exemple, pour le modèle de régression logistique, une régression par LASSO aurait pu être pertinente pour un meilleur choix de nos variables. Pour les autres modèles, des paramètres existent pour définir automatiquement les variables les plus explicatives

* **Ajout d’autres variables (autres fichiers excel)**

Sur notre data set de base en début de projet, nous avons uniquement exploité le fichier excel “crimes” sans exploiter les deux autres fichiers que nous avons eu à notre disposition pour des questions de temps.

Il serait pertinent d’exploiter ces autres fichiers renseignant des variables descriptives sociales et économiques (taux de chômage,natalité, etc …) qui peuvent mieux expliquer, ont un meilleur pouvoir explicatif du type de crime ‘Primary Type”.



* **Hyper-paramétrage**

Les modèles de Machine Learning lancés dans le cadre de ce projet ont été définis avec des paramètres choisis à la volée, sans réelle réflexion sur leur pertinence. Par exemple pour le modèle MLP de réseaux de neurones, nous avons construit 3 couches, nous avons également défini un paramètre max depth = 4 pour notre modèle de Random Forest.

Un des axes importants d’amélioration consiste ainsi à automatiser le choix des meilleures hyper-paramètres de nos modèles en utilisant des techniques de “**tuning d’hyper-paramètres**” avec notamment les fonctions de grid-search de la librairie “SParkMlib”. Ces techniques passent en revue un ensemble de paramètres pour un modèle donné et sont capables de définir les meilleurs paramètres possibles pour une meilleure prédiction.

Très souvent, ces techniques permettent de faire augmenter l’accuracy sans être en sur-apprentissage.

**Conclusion Sprint 2**

A l’issue du Sprint T2, plusieurs étapes ont été réalisées, notamment :

* Familiarisation et compréhension de l’Outil Spark, de son API Pyspark et ses librairies
* La construction d’indicateurs complexes à l’aide de la librairie Spark SQL
* La prédiction du type de crime “primary Type” avec des résultats moyens mais avec des pistes et axes d'amélioration pour avoir des meilleurs résultats.

Nos objectifs fixés en Sprint 0 ont été atteints comme le montre la BackLog ci-dessous. Nous avons dans un premier temps conçu des indicateurs au moyen de méthodes Big data différentes (PyMongo, Map Reduce, Spark) et avons réussi à modéliser notre base de données pour aboutir à une prédiction du type de crime.

Les axes d’amélioration possible du projet concernent la prédiction du type du crime avec des améliorations possibles dans la préparation de nos données et dans l’optimisation de nos algorithmes de Machine Learning pour avoir un meilleur score de prédiction.

Ci-dessous, la backlog mise à jour à l’issue de ce Sprint T2 :

