

Sommaire :

Sommaire :	1
Problématique :	3
Architecture du projet :	4
Détails sur la partie feeder (Bronze Layer) :	6
1. splitter.py	6
2. save_hdfs.py	6
3. main.py	8
Hadoop UI – YARN :	9
HDFS UI – Lakehouse/Bronze/	9
Spark UI :	11
4. verif_jour.py	12
5. config.py	13
Détails sur la partie preprocessor (Silver Layer) :	14
pre_traitement.py - Nettoyage et enrichissement du Bronze	14
Fonctions réutilisables :	14
Hadoop UI – YARN :	17
Spark UI :	18
HDFS UI – Table Hive :	20
PgADMIN4 – Metastore Hive:	20
Détails sur la partie ML (Silver Layer) :	23
severity_prediction.py - Machine Learning	23
Spark UI :	25
Hadoop UI – YARN :	26
HDFS UI – Table Hive :	27
PgADMIN4 - Metastore Hive :	28
join_ml.py - Jointure finale pour enrichissement	29
Hadoop UI – YARN :	30
HDFS UI – Table Hive :	30
PgADMIN4 – Metastore_hive :	31

Détails sur la partie Datamart (Gold Layer) :	32
datamarts.py - Export analytique vers PostgreSQL (Data Marts).....	32
PgADMIN4 – Datamart :	33
Spark UI :.....	34
Hadoop UI – YARN :	36
Détails sur la partie API :	37
main_api.py - API FastAPI pour exposer les données du datamart.....	37
FastAPI UI :	37
Business values :	39
Pour les agences de sécurité routière / collectivités locales :	41
❏ Pour les assureurs / réassureurs :	41
Recommandation stratégique développée	42
Récapulatif de l'ordre d'exécution des différents scripts :	43

Problématique :

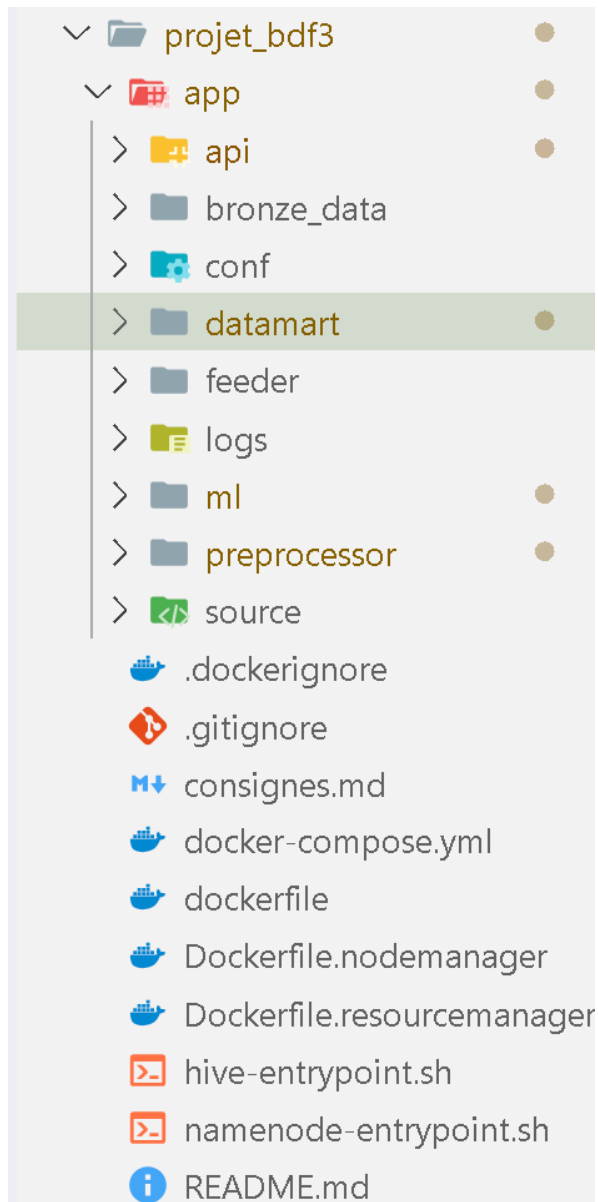
Comment les conditions environnementales, géographiques et temporelles influencent-elles la gravité des accidents routiers aux États-Unis, et comment peut-on anticiper ou réduire leur impact à l'aide de modèles prédictifs ?

Les accidents de la route sont un enjeu majeur de santé publique, de sécurité et de coûts logistiques. Grâce à un historique de **plus de 7,7 millions d'accidents** géolocalisés et horodatés, enrichis par des variables comme :

- la météo (Weather_Condition, Temperature(C), Humidity(%), Visibility_km)
- les infrastructures (Traffic_Signal, Junction, etc.)
- la temporalité (Sunrise_Sunset, Twilight, etc.)

...il devient possible d'identifier les conditions précises qui contribuent à des accidents plus graves (gravité **Severity**), d'entraîner des modèles prédictifs (**prediction_classe**), et de mieux orienter les politiques de prévention à travers le pays.

Cette problématique vise à exploiter ce gisement de données pour mieux **anticiper les situations à haut risque**, et mettre en place des actions ciblées et géolocalisées avant qu'un accident grave ne survienne.



Architecture du projet :

```
projet_bdf3/
├── app/
│   ├── api/          <-- API REST FastAPI pour exposer les données finales
│   ├── bronze_data/  <-- Données locales simulées (parquet/csv par date)
│   ├── conf/         <-- Paramètres de configuration éventuels
│   ├── datamart/     <-- Scripts Spark pour construire et exporter les datamarts
│   ├── feeder/       <-- Pipeline d'ingestion (local -> HDFS -> Bronze)
│   ├── logs/         <-- Fichiers logger horodatés pour chaque étape
│   ├── ml/           <-- Entraînement et prédiction via modèles ML
│   ├── preprocessor/ <-- Nettoyage, conversion (HDFS Bronze -> table Hive)
│   └── source/       <-- Fichier source brut (`US_Accidents_March23.csv`)
```

Racine du projet :

```
|— .dockerignore    <-- Ignore les fichiers inutiles pour Docker
|— .gitignore       <-- Fichiers/dossiers exclus du versionnement Git
|— consignes.md     <-- énoncé du projet
|— docker-compose.yml <-- Orchestration Docker : Spark, Hive, Hadoop, YARN etc.
|— dockerfile       <-- Image principale (Spark / PySpark)
|— Dockerfile.nodemanager <-- Dockerfile dédié au NodeManager YARN
|— Dockerfile.resourcemanager <-- Dockerfile pour le ResourceManager YARN
|— hive-entrypoint.sh <-- Script de démarrage personnalisé pour HiveServer2
|— namenode-entrypoint.sh <-- Script d'initialisation du NameNode HDFS
|— README.md        <-- Documentation du projet
```

Détails sur la partie feeder (Bronze Layer) :

1. splitter.py

But : Découper le gros fichier qui se trouve dans app/source/US Accidents March23.csv (+3Go) en **trois partitions par date sous format parquet**, y ajouter une colonne dates de simulation, puis sauvegarder chaque partie au format Parquet (et CSV pour vérifier les données) dans un répertoire local.

Fonctionnement :

- Lecture du fichier source CSV : US_Accidents_March23.csv
- Split aléatoire en 3 DataFrames (parquet)
- Ajout d'une colonne situation_date (3 dates simulées)
- Sauvegarde dans : `./bronze_data/temp/{date}/parquet/` et `./bronze_data/temp/{date}/csv/`

Lancer ce script en premier pour simuler une ingestion locale avant d'envoyer les fichiers dans HDFS avec :

```
Python feeder/splitter.py
```

Résultats :



2. save_hdfs.py

But : Lire les fichiers Parquet locaux générés ci-dessus et les écrire dans HDFS sous le dossier `source/raw/{date}/parquet/` (logique d'ingestion brute).

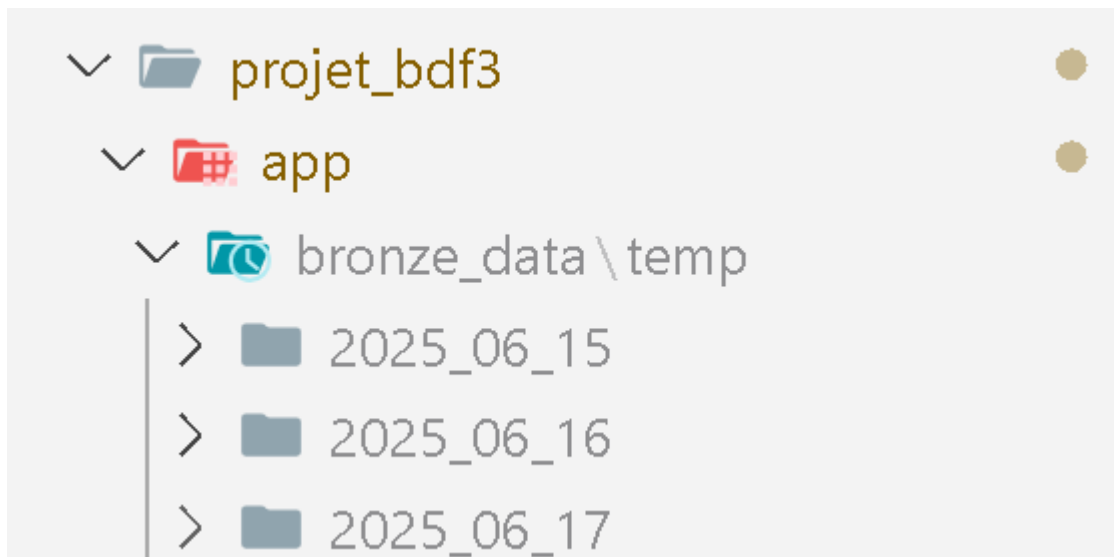
Fonctionnement :

- Liste les sous-dossiers locaux disponibles (ex: 2025_06_15, 2025_06_16, etc.)
- Pour chaque date :
 - Lecture Parquet en local
 - Suppression du dossier HDFS cible si existant
 - Écriture en mode overwrite sur HDFS

À exécuter après spliter.py pour peupler HDFS.

Python feeder/save_hdfs.py

Ici on exécute le script save_hdfs.py sur la machine hôte et pas dans le container spark projet_bdf3 pour envoyer le dossier bronze en local qui contient les 3 fichiers split



Dans

HDFS

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot

Startup Progress

Utilities

Browse Directory

/source/raw

Go!

Show

25

entries

Search:

<input type="checkbox"/>	<div></div> Permission	<div></div> Owner	<div></div> Group	<div></div> Size	<div></div> Last Modified	<div></div> Replication	<div></div> Block Size	<div></div> Name	<div></div>
<input type="checkbox"/>	drwxr-xr-x	root	supergroup	0 B	Jun 26 23:28	0	0 B	2025_06_15	<div></div>
<input type="checkbox"/>	drwxr-xr-x	root	supergroup	0 B	Jun 26 23:29	0	0 B	2025_06_16	<div></div>
<input type="checkbox"/>	drwxr-xr-x	root	supergroup	0 B	Jun 26 23:29	0	0 B	2025_06_17	<div></div>

Pour que dans YARN, le script *main.py* puisse le lire car si on ne le fait pas, le script va main.py va nous lever une erreur disant que le fichier n'existe pas si on décide de l'exécuter avec YARN

Environnement YARN != Environnement Spark

3. main.py

But : Lecture incrémentale des données depuis HDFS (source/raw) vers un dossier lakehouse/bronze, en fusionnant avec les données précédentes si elles existent (merge). C'est l'étape principale du feeder.

Fonctionnement :

- Recherche de la **dernière date** existante dans le bronze local !!!!
- Compare avec les dates disponibles dans raw pour lire la plus récente !!!
- Pour chaque nouvelle date :
 - Lecture des nouvelles données (pre_bronze)
 - Union avec le bronze existant si présent
 - Sauvegarde du résultat cumulé dans bronze/{new_date}/parquet/
 - Log via fichier + console

Ce script est à exécuter régulièrement pour enrichir le bronze depuis les fichiers HDFS.

```
spark-submit \  
  --master local \  
  --deploy-mode client \  
  --num-executors 2 \  
  --executor-cores 2 \  
  --executor-memory 2G \  
  --driver-memory 2G \  
  --conf spark.yarn.appMasterEnv.PYSPARK_PYTHON=python3 \  
  --conf spark.executorEnv.PYSPARK_PYTHON=python3 \  
  --conf spark.hadoop.yarn.resourcemanager.hostname=resourcemanager \  
  Feeder/main.py
```

Le script feeder/main.py a été lancé sur le cluster YARN en mode client, ce qui nous permet de visualiser les logs directement en local, tout en exécutant les tâches distribuées sur le cluster.

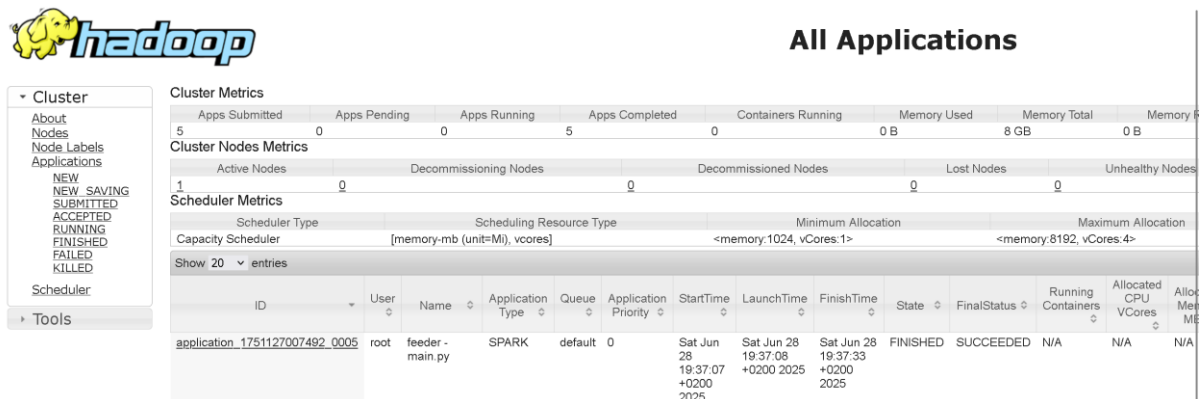
Nous avons configuré Spark pour utiliser **2 exécutors avec 2 cœurs et 2 Go de mémoire chacun**, ce qui permet de **paralléliser les opérations de lecture, filtrage ou découpage** des données sans surcharger le cluster.

Les options --conf utilisées dans la commande spark-submit permettent de configurer finement l'environnement d'exécution de Spark, en particulier dans un cluster YARN.

Par exemple, `spark.yarn.appMasterEnv.PYSPARK_PYTHON=python3` et `spark.executorEnv.PYSPARK_PYTHON=python3` garantissent que Spark utilisera Python 3 à la fois pour le driver (ApplicationMaster) et pour les executors, ce qui évite les incompatibilités entre versions de Python.

De plus, `spark.hadoop.yarn.resourcemanager.hostname=resourcemanager` permet de spécifier explicitement l'adresse du ResourceManager YARN lorsque Spark ne peut pas la détecter automatiquement, notamment dans un environnement Docker

Hadoop UI – YARN :



The screenshot shows the Hadoop UI for YARN. On the left is a sidebar with navigation links: Cluster, About Nodes, Node Labels, Applications (selected), NEW, NEW_SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED, Scheduler, and Tools. The main area is titled 'All Applications' and displays various metrics and a table of applications.

Cluster Metrics

Apps Submitted	0	Apps Pending	0	Apps Running	5	Apps Completed	0	Containers Running	0 B	Memory Used	8 GB	Memory Total	0 B
----------------	---	--------------	---	--------------	---	----------------	---	--------------------	-----	-------------	------	--------------	-----

Cluster Nodes Metrics

Active Nodes	0	Decommissioning Nodes	0	Decommissioned Nodes	0	Lost Nodes	0	Unhealthy Nodes	0
--------------	---	-----------------------	---	----------------------	---	------------	---	-----------------	---

Scheduler Metrics

Scheduler Type	Capacity Scheduler	Scheduling Resource Type	[memory-mb (unit=Mi), vcores]	Minimum Allocation	<memory:1024, vCores:1>	Maximum Allocation	<memory:8192, vCores:4>
----------------	--------------------	--------------------------	-------------------------------	--------------------	-------------------------	--------------------	-------------------------

Applications Table

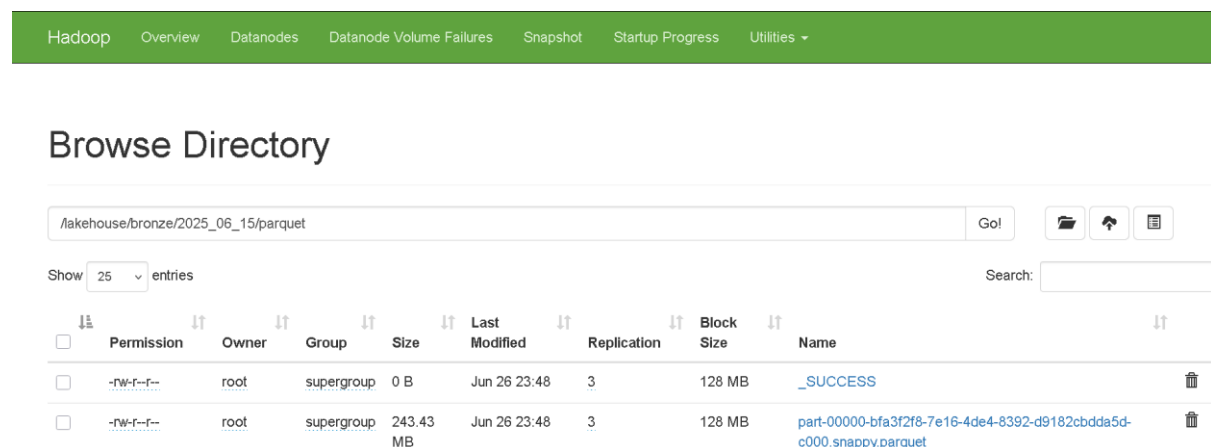
ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCoers	Allocated Memory
application_1751127007492_0005	root	feeder - main.py	SPARK	default	0	Sat Jun 28 19:37:07 +0200 2025	Sat Jun 28 19:37:08 +0200 2025	Sat Jun 28 19:37:33 +0200 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A

HDFS UI – Lakehouse/Bronze/

Ci-dessous est représenté processus du feeder:

Le premier parquet (correspondant à la date du 15-06-5025).

On voit que la taille du fichier est de 243.43 mb.



The screenshot shows the HDFS UI for Lakehouse/Bronze/. The top navigation bar includes links: Hadoop, Overview, Datanodes, Datanode Volume Failures, Snapshot, Startup Progress, and Utilities. The main area is titled 'Browse Directory' and shows a directory listing for '/lakehouse/bronze/2025_06_15/parquet'.

Directory Listing




Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	0 B	Jun 26 23:48	3	128 MB	_SUCCESS
-rw-r--r--	root	supergroup	243.43 MB	Jun 26 23:48	3	128 MB	part-00000-bfa3f2f8-7e16-4de4-8392-d9182cbdda5d-c000.snappy.parquet

Ci-dessous on voit la seconde date (16-06-2025) avec une taille de fichier de 486.91 mb




soit le double de la journée précédente (ce qui est normale puisque on ajoute la journée du 15-06 faisant 243.43 à la journée du 16-06 soit 243.43+

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities ▾

Browse Directory

/lakehouse/bronze/2025_06_16/parquet Go!   




Show 25 ▾ entries Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	0 B	Jun 26 23:50	3	128 MB	_SUCCESS	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	486.91 MB	Jun 26 23:50	3	128 MB	part-00000-62ad121b-ae89-45b4-9f6f-c3001a00ad16-c000.snappy.parquet	




Enfin ci-dessous est représenté le dernier jour (17-06-2025) qui reprend le dernier jour (ici 16-06-2025) est ajoute la journée du 17-06. Et on voit que cela a bien marché puisque on a 734.53 mb.

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities ▾

Browse Directory

/lakehouse/bronze/2025_06_17/parquet Go!   

Show 25 ▾ entries Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	0 B	Jun 26 23:52	3	128 MB	_SUCCESS	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	734.53 MB	Jun 26 23:52	3	128 MB	part-00000-b5c51c18-0ef0-4492-91a2-50e2e41e66a2-c000.snappy.parquet	

Spark UI

spark 3.5.6

JobsStagesStorageEnvironmentExecutorsSQL / DataFrame

feeder - main.py application UI

Spark Jobs (?)

User: root
Total Uptime: 9 s
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 6

Event Timeline

☐ Enable zooming

Executors

Added

Removed

Jobs

Succeeded

Failed

Running

Active Jobs (1)

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
6	parquet at NativeMethodAccessorImpl.java:0 parquet at NativeMethodAccessorImpl.java:0 (kill)	2025/06/28 23:21:20	2 s	0/1	0/7

spark 3.5.6

JobsStagesStorageEnvironmentExecutorsSQL / DataFrame

feeder - main.py application UI

Stages for All Jobs

Active Stages: 1
Completed Stages: 6
Skipped Stages: 2

Active Stages (1)

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go


Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
8	parquet at NativeMethodAccessorImpl.java:0 +details (kill)	2025/06/28 23:21:20	43 s	4/7 (1 running)	514.1 MiB			1294.7 MiB

Completed Stages (6)

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
7	count at NativeMethodAccessorImpl.java:0 +details	2025/06/28 23:21:19	16 ms	1/1			354.0 B	
5	count at NativeMethodAccessorImpl.java:0 +details	2025/06/28 23:21:19	0.3 s	6/6	221.3 KiB			354.0 B
4	parquet at NativeMethodAccessorImpl.java:0 +details	2025/06/28 23:21:18	62 ms	1/1				
3	count at NativeMethodAccessorImpl.java:0 +details	2025/06/28 23:21:18	0.1 s	1/1			59.0 B	
1	count at NativeMethodAccessorImpl.java:0 +details	2025/06/28 23:21:18	0.3 s	1/1	13.9 KiB			59.0 B
0	parquet at NativeMethodAccessorImpl.java:0 +details	2025/06/28 23:21:16	0.5 s	1/1				



[Jobs](#)
[Stages](#)
[Storage](#)
[Environment](#)
[Executors](#)
[SQL / DataFrame](#)

feeder - main.py application UI

SQL / DataFrame

Running Queries: 1
Completed Queries: 2

↳ Running Queries (1)

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go

ID ▾	Description	Submitted	Duration	Running Job IDs	Succeeded Job IDs	Failed Job IDs
2	parquet at NativeMethodAccessorImpl.java:0 <small>+ details</small>	2025/06/28 23:21:19	1.2 min	[7]	[6]	

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go

↳ Completed Queries (2)

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go

ID ▾	Description	Submitted	Duration	Job IDs
1	count at NativeMethodAccessorImpl.java:0 <small>+ details</small>	2025/06/28 23:21:19	0.4 s	[4][5]
0	count at NativeMethodAccessorImpl.java:0 <small>+ details</small>	2025/06/28 23:21:17	1 s	[1][2]

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go

4. verif_jour.py

But : Simulation pour vérifier que l'incrémentation, epartition t fonctionne bien
Il dupliquer un fichier Parquet existant et y ajouter une nouvelle valeur de date dans une colonne situation_date (utile pour simuler des données futures).

Fonctionnement :

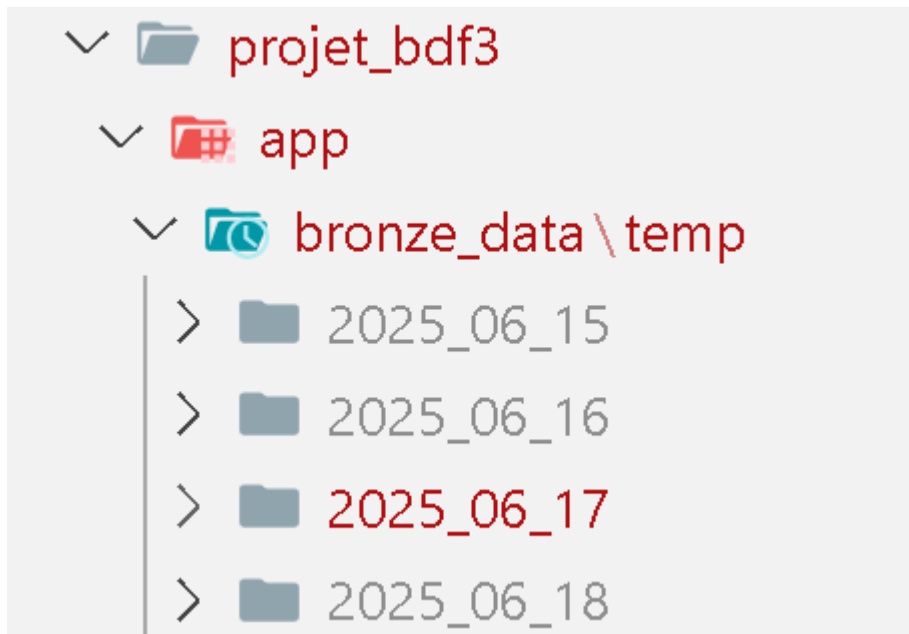
- Lecture d'un fichier .parquet
- Ajout colonne situation_date
- Sauvegarde dans un nouveau répertoire temp

Utilitaire, utile pour générer des données simulées.

```
Python feeder/verif_jour.py
```

Ici, le script est aussi à lancer dans la machine hôte, car il lit le dossier suivant

Résultats :



Et va créer une nouvelle partition avec une nouvelle date pour générer un nouveau parquet.

Ensuite nous relançons main.py et nous voyons dans les loggers qu'il reconnait bien qu'il y a un nouveau fichier, est qu'il va l'ajouter aux données du dernier fichier sur hdfs dans le lakehouse (dans notre cas 17-06-2025) etc (cf explication page 8-9-10).

5. config.py

But : Centraliser tous les chemins et constantes de configuration pour les autres scripts.

Fonctionnement :

- Contient les chemins HDFS (comme TEMP_PATH, BRONZE_ROOT...)
- Offre des méthodes statiques pour :
 - Récupérer les dates disponibles dans un dossier HDFS ou local
 - Construire dynamiquement des chemins vers les répertoires Parquet

Utilisé dans presque tous les scripts. Ce fichier ne s'exécute pas seul mais est importé.

Détails sur la partie preprocessor (Silver Layer) :

pre_traitement.py - Nettoyage et enrichissement du Bronze

But :

Ce script exécute une transformation approfondie des données stockées dans le dossier Bronze (HDFS), afin de produire deux DataFrames nettoyés :

- Un pour les analyses ML
- Un autre pour les jointures (description, localisation, condition météo conservées)

Les deux sont écrits dans des **tables Hive** pour exploitation ultérieure.

Fonctions principales :

- Conversion des unités (Fahrenheit → Celsius, Miles → KM, Inches → CM...)
- Nettoyage direction du vent + transformation en vecteurs sin/cos
- Indexation des catégories : Timezone, State, Weather_Condition
- Transformation des colonnes binaires (bool vers 0/1)
- Traitement des colonnes twilight (jour/nuit)
- Suppression de colonnes inutiles ou très bruitées
- Gestion des valeurs nulles : moyenne, zéro ou médiane
- Sauvegarde dans Hive

Fonctions réutilisables :

- **setup_logger()** : initialise des logs horodatés dans logs/ pour le debug et la traçabilité.
- **get_last_bronze_date()** : détecte automatiquement le dernier dossier disponible dans le Bronze.
- **apply_unit_conversions()** : convertit les unités impériales → métriques (mi → km, F → C...) et calcule des durées (duration_minutes_*).
- **transform_categorical_features()** : indexe les variables catégorielles (State, Timezone, Weather_Condition) pour ML.
- **transform_binary_features()** : convertit les colonnes booléennes (Stop, Signal, Railway...) en colonnes numériques binaires.
- **clean_wind_direction()** : transforme la direction du vent en angle (°), puis crée ses coordonnées polaires sin/cos.
- **manage_null_values()** : gère les valeurs manquantes avec différentes stratégies (moyenne, médiane, zéro).

- **drop_columns()** : supprime proprement les colonnes inutiles selon les cas d'usage (ML ou analyse).
- **save_to_hive()** : sauvegarde un DataFrame Spark vers une table Hive parquet.
- **log_df_show()** : affiche un échantillon du DataFrame dans les logs.

Ces fonctions sont **appelées à plusieurs reprises** dans le **même** script, souvent sur différentes copies du DataFrame (df, df2, df2_cleaned) pour répondre à plusieurs besoins analytiques

Lancement

```
spark-submit \  
  --master yarn \  
  --deploy-mode client \  
  --num-executors 1 \  
  --executor-cores 1 \  
  --executor-memory 1G \  
  --driver-memory 1G \  
  --conf spark.yarn.appMasterEnv.PYSPARK_PYTHON=python3 \  
  --conf spark.executorEnv.PYSPARK_PYTHON=python3 \  
  --conf spark.hadoop.yarn.resourcemanager.hostname=resourcemanager \  
  preprocessor/pre_traitement.py
```

Ici aussi le script preprocessor/pre_traitement.py a été lancé sur le cluster YARN en mode client, pour visualiser les logs directement en local, tout en exécutant les tâches distribuées sur le cluster. Mais ici, on a configuré Spark pour utiliser **1 exécutors avec 1 cœurs et 1 Go de mémoire chacun**

Les options --conf utilisées dans la commande spark-submit permettent de configurer finement l'environnement d'exécution de Spark, en particulier dans un cluster YARN.

Par exemple, spark.yarn.appMasterEnv.PYSPARK_PYTHON=python3 et spark.executorEnv.PYSPARK_PYTHON=python3 garantissent que Spark utilisera Python 3 à la fois pour le driver (ApplicationMaster) et pour les executors, ce qui évite les incompatibilités entre versions de Python.

De plus, spark.hadoop.yarn.resourcemanager.hostname=resourcemanager permet de spécifier explicitement l'adresse du ResourceManager YARN lorsque Spark ne peut pas la détecter automatiquement, notamment dans un environnement Docker

Exemple de colonnes produites (df pour ML)

- Temperature(C)
- Wind_Speed_kmh
- Humidity(%)
- Weather_Condition_index
- Timezone_index, State_index
- wind_dir_sin, wind_dir_cos
- duration_minutes_accident, duration_minutes_record_weather
- Amenity_num, Traffic_Signal_num...

Exemple de colonnes conservées (df2 pour jointure)

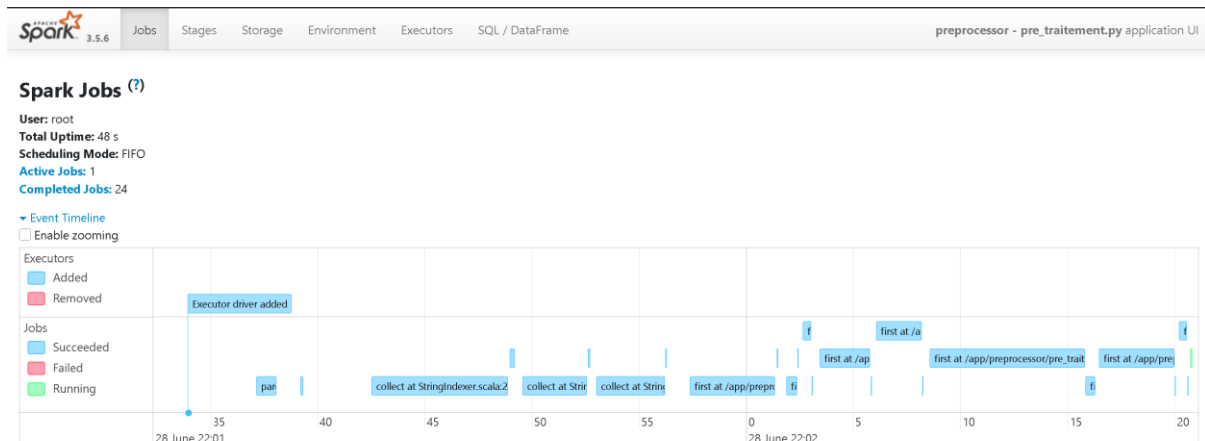
- Description, City, Zipcode, Street, Weather_Condition
- Sunrise_Sunset, Civil_Twilight, etc.

Hadoop UI – YARN :

Cluster Metrics																																	
4	Apps Submitted			0	Apps Pending		0	Apps Running		4	Apps Completed		0	Containers Running		0 B	Memory Used		8 GB	Memory Total		0 B	Memory Reserved		0	VCores Used		8	VCores Total		0	VCores Reserved	
Cluster Nodes Metrics																																	
1	Active Nodes			Decommissioning Nodes			0	Decommissioned Nodes			0	Lost Nodes			0	Unhealthy Nodes			0	Rebooted Nodes			0	Shutdown Nodes									
Scheduler Metrics																																	
Scheduler Type				Scheduling Resource Type				Minimum Allocation				Maximum Allocation				Maximum Cluster Application Priority																	
Capacity Scheduler				[memory-mb (unit=M), vcores]				<memory 1024, vCores 1>				<memory 8192, vCores 4>				0																	
Show 20 entries																																	
Search																																	
ID	+	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU Vcores	Allocated Memory MB	Reserved CPU Vcores	Reserved Memory MB	% of Queue	% of Cluster	Progress	Tracking UI	Blacklisted Nodes												
application.1751413250509_0007		root	preprocessor - pre_treatment.py	SPARK	default	0	Sat Jun 28 23:45:51 +0000 2025	Sat Jun 28 23:45:52 +0000 2025	Sat Jun 28 23:46:11 +0000 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0		History	0												

Accepted -> Running -> Finished

Spark UI :



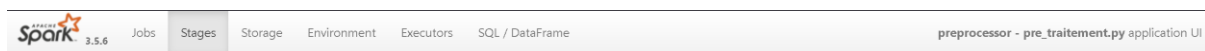
Active Jobs (1)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
24	first at /app/preprocessor/pre_traitement.py:205 first at /app/preprocessor/pre_traitement.py:205 (kill)	2025/06/28 22:02:20	46 ms	0/1	0/6

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Completed Jobs (24)



Stages for All Jobs

Active Stages: 1
Completed Stages: 37
Skipped Stages: 16

Active Stages (1)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go


Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
53	showString at NativeMethodAccessorImpl.java:0 +details (kill)	2025/06/28 22:02:38	Unknown	0/6				

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Completed Stages (37)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
52	count at NativeMethodAccessorImpl.java:0 +details	2025/06/28 22:02:37	20 ms	1/1			354.0 B	
50	count at NativeMethodAccessorImpl.java:0 +details	2025/06/28 22:02:37	0.2 s	6/6	221.3 KiB			354.0 B
49	showString at NativeMethodAccessorImpl.java:0 +details	2025/06/28 22:02:36	0.4 s	1/1	128.4 MiB			
48	showString at NativeMethodAccessorImpl.java:0 +details	2025/06/28 22:02:35	0.6 s	1/1	128.4 MiB			
47	showString at NativeMethodAccessorImpl.java:0 +details	2025/06/28 22:02:34	0.7 s	1/1	48.0 MiB			
46	first at /app/preprocessor/pre_traitement.py:222 +details	2025/06/28 22:02:33	0.3 s	1/1			226.8 KiB	
44	first at /app/preprocessor/pre_traitement.py:222 +details	2025/06/28 22:02:26	7 s	6/6	7.3 MiB			226.8 KiB
43	first at /app/preprocessor/pre_traitement.py:205 +details	2025/06/28 22:02:25	18 ms	1/1			408.0 B	
41	first at /app/preprocessor/pre_traitement.py:205 +details	2025/06/28 22:02:23	2 s	6/6	3.3 MiB			408.0 B
40	first at /app/preprocessor/pre_traitement.py:205 +details	2025/06/28 22:02:23	44 ms	6/6			408.0 B	


3.5.6

[Jobs](#)
[Stages](#)
[Storage](#)
[Environment](#)
[Executors](#)
[SQL / DataFrame](#)

preprocessor - pre_traitement.py application UI

SQL / DataFrame

Running Queries: 1
Completed Queries: 19

▼ Running Queries (1)

Page:
1 Pages. Jump to . Show items in a page. Go

ID ▾	Description	Submitted	Duration	Running Job IDs	Succeeded Job IDs	Failed Job IDs
19	showString at NativeMethodAccessorImpl.java:0 +details	2025/06/28 22:02:38	1.0 min	[37]		

Page:
1 Pages. Jump to . Show items in a page. Go

▼ Completed Queries (19)

Page:
1 Pages. Jump to . Show items in a page. Go

ID ▾	Description	Submitted	Duration	Job IDs
18	count at NativeMethodAccessorImpl.java:0 +details	2025/06/28 22:02:37	0.3 s	[35][36]
17	showString at NativeMethodAccessorImpl.java:0 +details	2025/06/28 22:02:36	0.6 s	[34]
16	showString at NativeMethodAccessorImpl.java:0 +details	2025/06/28 22:02:35	0.8 s	[33]
15	showString at NativeMethodAccessorImpl.java:0 +details	2025/06/28 22:02:33	1 s	[32]
14	first at /app/preprocessor/pre_traitement.py:222 +details	2025/06/28 22:02:26	7 s	[30][31]
13	first at /app/preprocessor/pre_traitement.py:205	2025/06/28 22:02:23	2 s	[28][29]

HDFS UI – Table Hive :

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities										
Browse Directory										
<input type="text" value="/user/hive/warehouse/preprocessed_accidents_for_join"/> <input type="button" value="Go!"/>										
Show 25 entries Search: <input type="text"/>										
	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name		
<input type="checkbox"/>	-rw-r--r--	root	supergroup	0 B	Jun 28 18:56	3	128 MB	_SUCCESS		
<input type="checkbox"/>	-rw-r--r--	root	supergroup	602.1 MB	Jun 28 18:56	3	128 MB	part-00000-5c019dbc-b53c-4335-a12d-4d2115125d80-c000.snappy.parquet		

Les données de la couche silver ont bien été stockées dans une table Hive nommée : preprocessed accidents for join

PgADMIN4 – Metastore Hive:

BDD Hive présent dans le chemin d'accès HDFS : /user/hive/warehouse/

public.DBS/metastore_hive/postgres@PostgreSQL 15						
100 rows						
Query Query History						
1 SELECT * FROM public."DBS"						
2 ORDER BY "DB_ID" ASC LIMIT 100						
3						
Data Output Messages Notifications						
DB_ID	DESC	DB_LOCATION_URI	NAME	OWNER_NAME	OWNER_TYPE	
[PK] bigint	character varying (4000)	character varying (4000)	character varying (128)	character varying (128)	character varying (10)	
1	Default Hive database	hdfs://namenode:8020/user/hive/warehouse	default	public	ROLE	

public.TBLS/metastore_hive/postgres@PostgreSQL 15

100 rows

Query Query History

```
1 SELECT * FROM public."TBLS"
2 ORDER BY "TBL_ID" ASC LIMIT 100
3
```

Scratch Pad

Data Output Messages Notifications

	TIME	OWNER character varying (767)	RETENTION bigint	SD_ID bigint	TBL_NAME character varying (256)	TBL_TYPE character varying (128)	VIEW_EXPANDED_TEXT text	VIEW_ORIGINAL_TEXT text
1	0	root	0	1	test	MANAGED_TABLE	[null]	[null]
2	0	root	0	4	test_ctypes_resolu	MANAGED_TABLE	[null]	[null]
3	0	root	0	6	preprocessed_accidents	MANAGED_TABLE	[null]	[null]
4	0	root	0	7	preprocessed_accidents_clean...	MANAGED_TABLE	[null]	[null]
5	0	root	0	11	preprocessed_accidents_silver	MANAGED_TABLE	[null]	[null]
6	0	root	0	16	preprocessed_accidents_for_ml	MANAGED_TABLE	[null]	[null]
7	0	root	0	21	predicted_severity	MANAGED_TABLE	[null]	[null]
8	0	root	0	23	predicted_severity_full	MANAGED_TABLE	[null]	[null]
9	0	root	0	31	preprocessed_accidents_for_join	MANAGED_TABLE	[null]	[null]

Et on voit bien ici dans le metastore_hive sur Postgres que preprocessed_accidents_for_join est bien **reconnu comme une table Hive**

public.SDS/metastore_hive/postgres@PostgreSQL 15

100 rows

Query Query History

```
1 SELECT * FROM public."SDS"
2 ORDER BY "SD_ID" ASC LIMIT 100
3
```

Scratch Pad

Data Output Messages Notifications

	SD_ID [PK] bigint	INPUT_FORMAT character varying (4000)	IS_COMPRESSED boolean	LOCATION character varying (4000)
1	1	org.apache.hadoop.mapred.TextInputFormat	false	hdfs://namenode:8020/user/hive/warehouse/test
2	4	org.apache.hadoop.hive ql.io.parquet.MapredParquetInputFormat	false	hdfs://namenode:8020/user/hive/warehouse/test_hive.db/test_ctypes_resolu
3	5	org.apache.hadoop.hive ql.io.parquet.MapredParquetInputFormat	false	hdfs://namenode:8020/user/hive/warehouse/test_hive.db/test_ctypes_resolu/date
4	6	org.apache.hadoop.hive ql.io.parquet.MapredParquetInputFormat	false	hdfs://namenode:8020/user/hive/warehouse/preprocessed_accidents
5	7	org.apache.hadoop.hive ql.io.parquet.MapredParquetInputFormat	false	hdfs://namenode:8020/user/hive/warehouse/preprocessed_accidents_cleaned
6	11	org.apache.hadoop.hive ql.io.parquet.MapredParquetInputFormat	false	hdfs://namenode:8020/user/hive/warehouse/preprocessed_accidents_silver
7	16	org.apache.hadoop.hive ql.io.parquet.MapredParquetInputFormat	false	hdfs://namenode:8020/user/hive/warehouse/preprocessed_accidents_for_ml
8	21	org.apache.hadoop.hive ql.io.parquet.MapredParquetInputFormat	false	hdfs://namenode:8020/user/hive/warehouse/predicted_severity
9	23	org.apache.hadoop.hive ql.io.parquet.MapredParquetInputFormat	false	hdfs://namenode:8020/user/hive/warehouse/predicted_severity_full
10	31	org.apache.hadoop.hive ql.io.parquet.MapredParquetInputFormat	false	hdfs://namenode:8020/user/hive/warehouse/preprocessed_accidents_for_join

Là, on peut voir le format (le parquet) du contenu de la table Hive, et aussi son chemin d'accès dans HDFS

Détails sur la partie ML (Silver Layer) :

severity_prediction.py - Machine Learning

But

Ce script entraîne un modèle de classification supervisée (Random Forest) sur les données nettoyées issues du preprocessing, dans le but de prédire le niveau de gravité des accidents (Severity) en fonction de plusieurs paramètres.

Fonctionnement

1. Initialisation de la session Spark avec Hive support
2. Lecture de la table prétraîtée pour ML
3. Construction de la table de features + nettoyage
4. Split train/test et entraînement
5. Prédiction + ajout d'une colonne prediction_classe
6. Évaluation de l'accuracy
7. Sauvegarde finale dans Hive

Lancement

```
spark-submit \  
  --master yarn \  
  --deploy-mode client \  
  --num-executors 2 \  
  --executor-cores 2 \  
  --executor-memory 2G \  
  --driver-memory 2G \  
  --conf spark.yarn.appMasterEnv.PYSPARK_PYTHON=python3 \  
  --conf spark.executorEnv.PYSPARK_PYTHON=python3 \  
  --conf spark.hadoop.yarn.resourcemanager.hostname=resourcemanager \  
  ml/severity_prediction.py
```

toujours pareil, on lance sur le cluster YARN en mode client. Mais ici, on a configuré Spark pour utiliser 2 **exécuteurs avec 2 cœurs et 2 Go de mémoire** parce que le modèle s'entraîne sur un petit dataset (car notre machine n'est pas assez puissante pour le faire tourner)

Les options `--conf` utilisées dans la commande `spark-submit` permettent de configurer finement l'environnement d'exécution de Spark, en particulier dans un cluster YARN.

Par exemple, `spark.yarn.appMasterEnv.PYSPARK_PYTHON=python3` et `spark.executorEnv.PYSPARK_PYTHON=python3` garantissent que Spark utilisera Python 3 à la fois pour le driver (ApplicationMaster) et pour les executors, ce qui évite les incompatibilités entre versions de Python.

De plus, `spark.hadoop.yarn.resourcemanager.hostname=resourcemanager` permet de spécifier explicitement l'adresse du ResourceManager YARN lorsque Spark ne peut pas la détecter automatiquement, notamment dans un environnement Docker

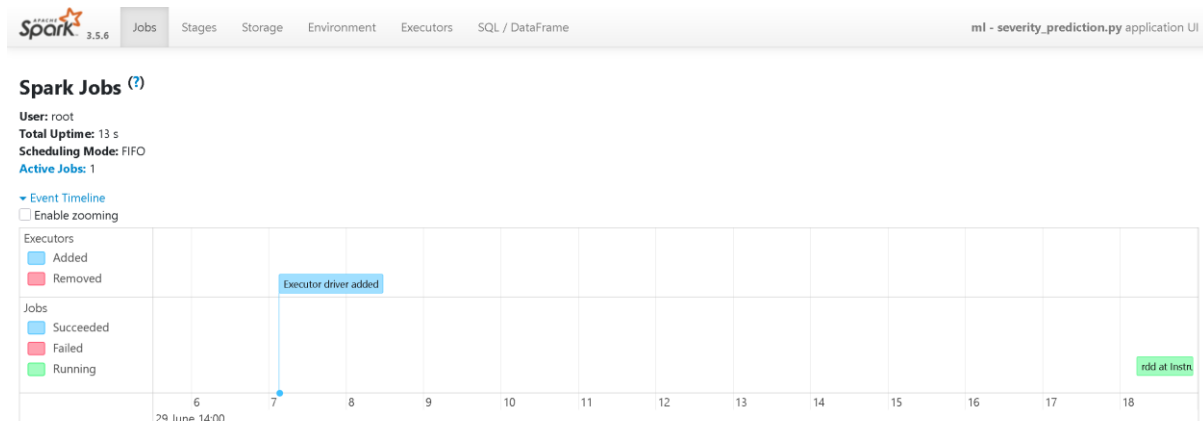
Exemple de colonnes utilisées pour les features

- Temperature(C), Wind_Speed_kmh, Humidity(%)
- Weather_Condition_index, Timezone_index, State_index
- duration_minutes_accident, wind_dir_sin, etc.

Sortie finale

- Une table Hive `default.predicted_severity_full` contenant :
 - Toutes les colonnes originales
 - Une colonne `prediction_classe` (cast de prediction en int)

Spark UI :

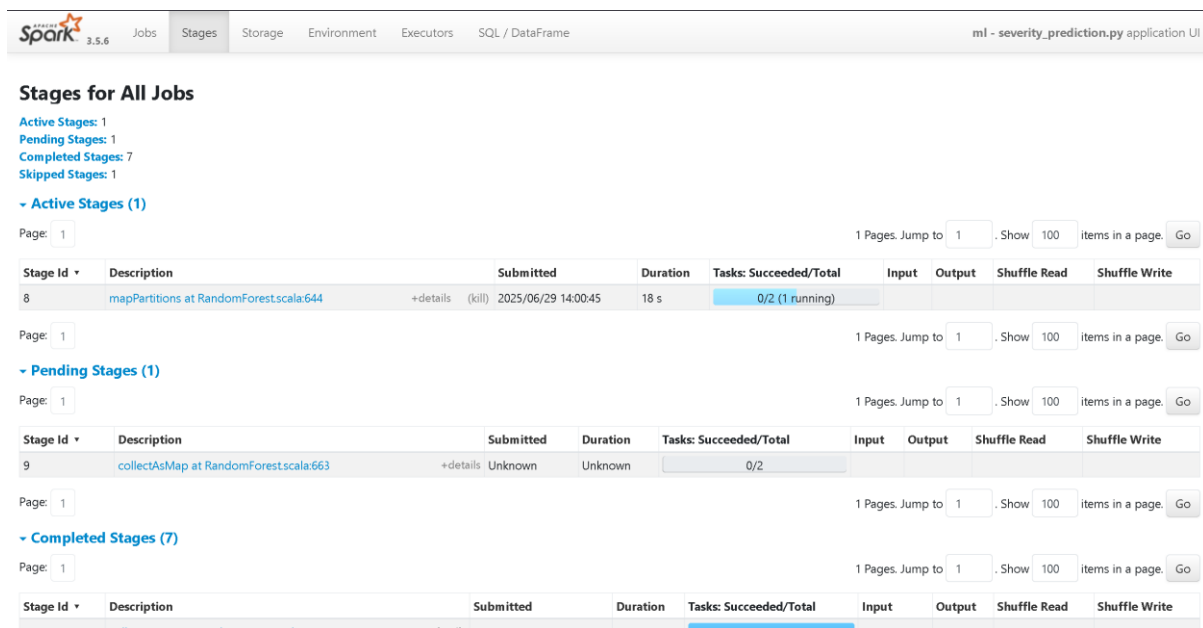


Active Jobs (1)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	rdd at Instrumentation.scala:62 rdd at Instrumentation.scala:62	2025/06/29 14:00:18 (kill)	0.7 s	0/1	0/2 (1 running)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go



Storage

→ RDDs

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
4	* (1) Project [ID#0, Severity#1, Humidity(C)#2, Pressure(in)#3, Distance_km#4, Temperature(C)#5, Wind_Chill(C)#6, Visibility_km#7, Wind_Speed_kmh#8, Precipitation(cm)#9, duration_minutes_accident#10, duration_minutes_record_weather#11, wind_dir_sin#12, wind_dir_cos#13, Timezone_index#14, State_index#15, Weather_Condition_index#16, Amenity_num#17, Bump_num#18, Crossing_num#19, Give_Way_num#20, Junction_num#21, No_Exit_num#22, Railway_num#23, ... 11 more fields] +- *(1) Filter atleastnonnulls(33, Humidity(C)#2, Pressure(in)#3, Distance_km#4, Temperature(C)#5, Wind_Chill(C)#6, Visibility_km#7, Wind_Speed_kmh#8, Precipitation(cm)#9, duration_minutes_accident#10, duration_minutes_record_weather#11, wind_dir_sin#12, wind_dir_cos#13, Timezone_index#14, State_index#15, Weather_Condition_index#16, Amenity_num#17, Bump_num#18, Crossing_num#19, Give_Way_num#20, Junction_num#21, No_Exit_num#22, Railway_num#23, Roundabout_num#24, ... 10 more fields) +- *(1) Sample 0.0, 0.1, false, 42 +- *(1) ColumnarToRow ...	Disk Memory Deserialized 1x Replicated	2	100.00%	264.6 MiB	8.9 MiB
36	bagged tree points	Disk Memory Deserialized 1x Replicated	2	100.00%	252.8 MiB	0.0 B

SQL / DataFrame

Completed Queries: 1

→ Completed Queries (1)

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go

ID	Description	Submitted	Duration	Job IDs
0	take at DatasetUtils.scala:193	2025/06/29 14:00:33	3 s	[1][2]

Page: 1

1 Pages. Jump to 1. Show 100 items in a page. Go

Hadoop UI – YARN :

YDP

All Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used
2	0	1	1	1	1 GB	8 GB	0 B	8

Cluster Nodes Metrics


Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
1	0	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximur
Capacity Scheduler	[memory-mb (unit-M), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores	Allocated Memory MB	Reserved CPU VCores	Reserved Memory MB	% of Queue	% of Cluster
application_1751205420657_0002	root	ml - severity_prediction.py	SPARK	default	0	Sun Jun 29 16:13:42 +0200 2025	Sun Jun 29 16:13:42 +0200 2025	N/A	ACCEPTED	UNDEFINED	1	1	1024	0	0	12.5	12.5



All Applications

Cluster

About

Nodes

Node Labels

Applications

NEW

NEW, SAVING

SUBMITTED

ACCEPTED

RUNNING

FINISHED

FAILED

KILLED

Scheduler

Tools

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	Virtual Memory
2	0	0	2	0	0 B	8 GB	0 B	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes
1	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[memory-mb (unit=M), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU Vcores	Allocated Memory MB	Reserved CPU Vcores
application_1751205420657_0002	root	ml-severity_prediction.py	SPARK	default	0	Sun Jun 29 16:13:42 +0200 2025	Sun Jun 29 16:13:42 +0200 2025	Sun Jun 29 16:13:59 +0200 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A

HDFS UI – Table Hive :

Hadoop

Overview

Datanodes

Datanode Volume Failures

Snapshot




Startup Progress

Utilities

Browse Directory




/user/hive/warehouse/predicted_severity_full

Go!



Show 25 entries

Search:

	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	0 B	Jun 27 22:35	3	128 MB	_SUCCESS	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	13.43 MB	Jun 27 22:35	3	128 MB	part-00000-9bcbe645-3a33-4d9b-bcba-ab4a576a64a2-c000.snappy.parquet	
<input type="checkbox"/>	-rw-r--r--	root	supergroup	1.81 MB	Jun 27 22:35	3	128 MB	part-00001-9bcbe645-3a33-4d9b-bcba-ab4a576a64a2-c000.snappy.parquet	

PgADMIN4 - Metastore Hive :

public.TBLS/metastore_hive/postgres@PostgreSQL 15

100 rows

Query Query History Scratch Pad

```
1 SELECT * FROM public."TBLS"  
2 ORDER BY "TBL_ID" ASC LIMIT 100  
3
```

Data Output Messages Notifications

	OWNER character varying (767)	RETENTION bigint	SD_ID bigint	TBL_NAME character varying (256)	TBL_TYPE character varying (128)	VIEW_EXPANDED_TEXT text	VIEW_ORIGINAL_TEXT text	IS_REWI boolean
1	root	0	1	test	MANAGED_TABLE	[null]	[null]	false
2	root	0	4	test_ctypes_resolu	MANAGED_TABLE	[null]	[null]	false
3	root	0	6	preprocessed_accidents	MANAGED_TABLE	[null]	[null]	false
4	root	0	7	preprocessed_accidents_clean...	MANAGED_TABLE	[null]	[null]	false
5	root	0	11	preprocessed_accidents_silver	MANAGED_TABLE	[null]	[null]	false
6	root	0	16	preprocessed_accidents_for_ml	MANAGED_TABLE	[null]	[null]	false
7	root	0	21	predicted_severity	MANAGED_TABLE	[null]	[null]	false
8	root	0	23	predicted_severity_full	MANAGED_TABLE	[null]	[null]	false

join_ml.py - Jointure finale pour enrichissement

Quand l'exécuter

Ce script doit être lancé **après** l'exécution de :

1. pre_traitement.py – qui génère la table preprocessed_accidents_for_join
2. severity_prediction.py – qui produit les prédictions dans predicted_severity_full

But

Fusionner les résultats de prédiction (prediction_classe) avec les colonnes descriptives conservées dans le DataFrame preprocessed_accidents_for_join, afin de produire une table finale enrichie et exploitable.

Fonctions

- Lecture de default.predicted_severity_full
- Lecture de default.preprocessed_accidents_for_join
- Jointure sur la colonne ID
- Sélection des colonnes utiles
- Sauvegarde de la table finale enrichie dans Hive : default.final_joined_accidents

Lancement

```
spark-submit \  
--master yarn \  
--deploy-mode client \  
--num-executors 1 \  
--executor-cores 1 \  
--executor-memory 1G \  
--driver-memory 1G \  
--conf spark.yarn.appMasterEnv.PYSPARK_PYTHON=python3 \  
--conf spark.executorEnv.PYSPARK_PYTHON=python3 \  
--conf spark.hadoop.yarn.resourcemanager.hostname=resourcemanager \  
Preprocessor/join_ml.py
```

Sortie finale :

- Table Hive : default.final_joined_accidents
- Contient :
 - Données descriptives enrichies (ville, météo, description...)

- Colonne prediction_classe

Hadoop UI – YARN :

Cluster Metrics

Apps Submitted	0	Apps Pending	1	Apps Running	0	Apps Completed	1	Containers Running	1 GB	Memory Used	8 GB	Memory Total	0 B	Memory Reserved	1	VCo	VC	
Cluster Nodes Metrics																		
Active Nodes		Decommissioning Nodes				Decommissioned Nodes				Lost Nodes		Unhealthy Nodes		Rebooted Nodes				
1		0				0				0		0				0		
Scheduler Metrics																		
Scheduler Type			Scheduling Resource Type					Minimum Allocation				Maximum Allocation				Maximum		
Capacity Scheduler			[memory-mb (unit=Mi), vcores]					<memory:1024, vCores:1>				<memory:8192, vCores:4>				0		
Show 20 ▾ entries																		
ID	▾	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCo	Allocated Memory MB	Reserved CPU VCo	Reserved Memory MB	% of Queue	% of Cluster
application_1751214151276_0001																		
		root	preprocessor-join_ml.py	SPARK	default	0	Sun Jun 29 18:38:16 +0200 2025	Sun Jun 29 18:38:19 +0200 2025	N/A	ACCEPTED	UNDEFINED	1	1	1024	0	0	12.5	12.5

Cluster Metrics

Apps Submitted	0	Apps Pending	0	Apps Running	1	Apps Completed	0	Containers Running	0 B	Memory Used	8 GB	Memory Total	0 B	Memory Reserved	0	VCo	VCores Total	
Cluster Nodes Metrics																		
Active Nodes		Decommissioning Nodes				Decommissioned Nodes				Lost Nodes		Unhealthy Nodes		Rebooted Nodes				
1		0				0				0		0				0		
Scheduler Metrics																		
Scheduler Type			Scheduling Resource Type					Minimum Allocation				Maximum Allocation				Maximum Cluster Appl		
Capacity Scheduler			[memory-mb (unit=Mi), vcores]					<memory:1024, vCores:1>				<memory:8192, vCores:4>				0		
Show 20 ▾ entries																		
ID	▾	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCo	Allocated Memory MB	Reserved CPU VCo	Reserved Memory MB	% of Queue	% of Cluster
application_1751214151276_0001																		
		root	preprocessor-join_ml.py	SPARK	default	0	Sun Jun 29 18:38:16 +0200 2025	Sun Jun 29 18:38:19 +0200 2025	Sun Jun 29 18:38:37 +0200 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	0.0	0.0

HDFS UI – Table Hive :

Hadoop	Overview	Datanodes	Datanode Volume Failures	Snapshot	Startup Progress	Utilities
--------	----------	-----------	--------------------------	----------	------------------	-----------

Browse Directory

/user/hive/warehouse/final_joined_accidents

Go!

Show 25 entries

Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	root	supergroup	0 B	Jun 29 17:25	3	128 MB	__SUCCESS
-rw-r--r--	root	supergroup	63.24 MB	Jun 29 17:25	3	128 MB	part-00000-2c7c6d50-2a3b-4cfe-8f4f-d6ae2c76d6e9-c000.snappy.parquet

PgADMIN4 – Metastore_hive :

The screenshot shows the PgAdmin4 web interface. The top bar indicates the connection to 'public.TBLS/metastore_hive/postgres@PostgreSQL 15'. Below the toolbar, the 'Query' tab is active, displaying a SQL query:

```
1 SELECT * FROM public."TBLS"  
2 ORDER BY "TBL_ID" ASC LIMIT 100  
3
```

The 'Data Output' tab is also visible, showing the results of the query in a table with 10 rows and 10 columns. The columns are: TBL_ID, CREATE_TIME, DB_ID, LAST_ACCESS_TIME, OWNER, RETENTION, SD_ID, TBL_NAME, and TBL_TYPE. The data shows various tables managed by 'root'.

	TBL_ID [PK] bigint	CREATE_TIME bigint	DB_ID bigint	LAST_ACCESS_TIME bigint	OWNER character varying (767)	RETENTION bigint	SD_ID bigint	TBL_NAME character varying (256)	TBL_TYPE character varying (128)
1	1	1750975139	1	0	root	0	1	test	MANAGED_TABLE
2	3	1750978137	7	0	root	0	4	test_ctypes_resolu	MANAGED_TABLE
3	6	1751016375	1	0	root	0	6	preprocessed_accidents	MANAGED_TABLE
4	7	1751018439	1	0	root	0	7	preprocessed_accidents_clean...	MANAGED_TABLE
5	11	1751031182	1	0	root	0	11	preprocessed_accidents_silver	MANAGED_TABLE
6	16	1751036592	1	0	root	0	16	preprocessed_accidents_for_ml	MANAGED_TABLE
7	21	1751050480	1	0	root	0	21	predicted_severity	MANAGED_TABLE
8	23	1751056556	1	0	root	0	23	predicted_severity_full	MANAGED_TABLE
9	31	1751129774	1	0	root	0	31	preprocessed_accidents_for_join	MANAGED_TABLE
10	36	1751210745	1	0	root	0	36	finaljoined_accidents	MANAGED_TABLE

Détails sur la partie Datamart (Gold Layer) :

datamarts.py - Export analytique vers PostgreSQL (Data Marts)

Quand l'exécuter

Ce script doit être lancé **en dernier**, une fois que la table default.final_joined_accidents a été générée via join_ml.py. Il produit plusieurs **marts analytiques** sur PostgreSQL à partir des données finales enrichies.

But

Créer plusieurs tables agrégées (Data Marts) pour analyse rapide :

- Par ville
- Par condition météo
- Par type d'infrastructure
- Accidents graves selon la météo
- Export complet de la table finale vers PostgreSQL

Fonctions

- Lecture de la table Hive final_joined_accidents
- Agrégations multiples avec groupBy
- Export via .write.jdbc() vers PostgreSQL

Détail des marts produits

1. dm_accidents_by_city : nb d'accidents, distance, durée, sévérité par ville et État
2. dm_weather_accidents : accidents selon conditions météo + température, précipitations
3. dm_accidents_by_infra : effet des infrastructures (stop, railway...)
4. dm_grave_accidents_weather : accidents de sévérité 4 par météo
5. gold_final_accidents : dump complet de la table finale enrichie

Lancement

```
spark-submit \  
--master yarn \  
--deploy-mode client \  
--num-executors 5 \  

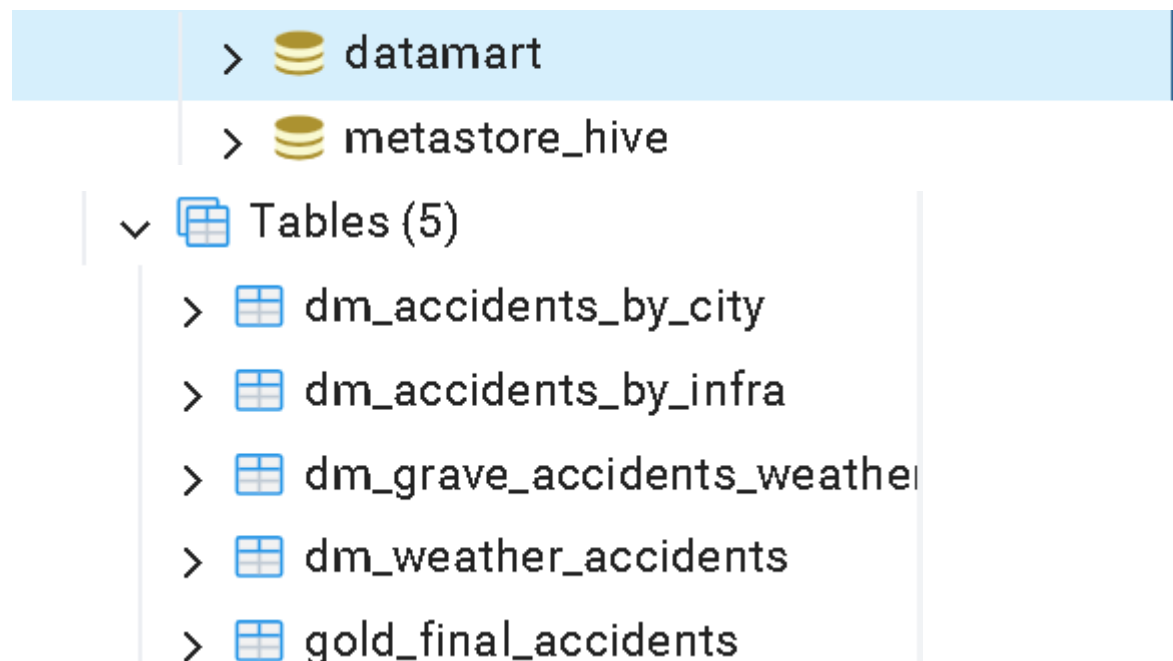
```



```
--executor-cores 2 \  
--executor-memory 1G \  
--driver-memory 1G \  
--conf spark.yarn.appMasterEnv.PYSPARK_PYTHON=python3 \  
--conf spark.executorEnv.PYSPARK_PYTHON=python3 \  
--conf spark.hadoop.yarn.resourcemanager.hostname=resourcemanager \  
datamart/datamarts.py
```

Là on veut avoir 5 executors car il y'a 5 datamarts à générer ce qui peut etre long avec qu'un seul executor. La mémoire on la met à 1GB car les datamarts sont des agrégats donc plus léger que le dataset d'origine

PgADMIN4 – Datamart :



public.gold_final_accidents/datamart/postgres@PostgreSQL 15

100 rows

Query Query History

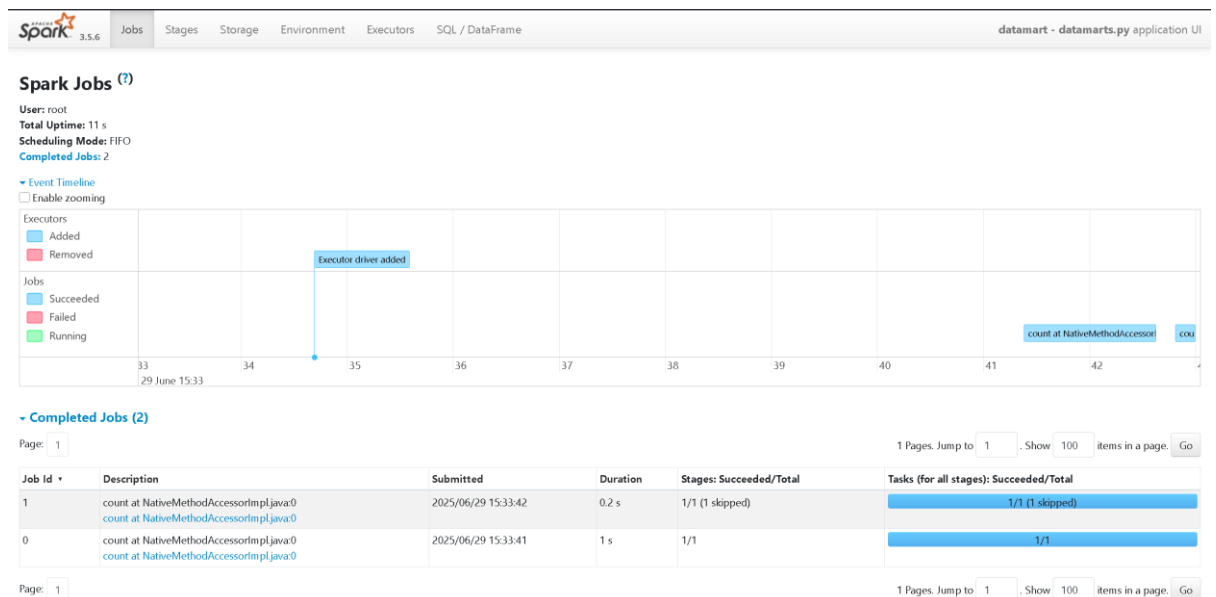
```
1 SELECT * FROM public.gold_final_accidents
2 LIMIT 100
3
```

Scratch Pad

Data Output Messages Notifications

	ID text	Severity integer	Start_Lat double precision	Start_Lng double precision	End_Lat double precision	End_Lng double precision	Description text
1	A-6481366	4	35.346149	-106.597106	35.342631	-106.590983	Incident on US-550 EB near MM 4 Road closed. Take a
2	A-6481526	2	40.72150111179008	-73.9745199676801	40.72809111179008	-73.9716599676801	Crash on Franklin D. Roosevelt Drive northbound at 9th
3	A-6481585	2	35.797746999999994	-78.5566	35.797459	-78.555717	Incident on NEW BERN AVE near HEDINGHAM BLVD D
4	A-6482083	2	43.151421	-77.634507	43.150263	-77.636096	Incident on BROWN ST near KENSINGTON ST Drive wi
5	A-6482223	2	38.756514787489145	-90.01391529994005	38.75635478748913	-90.00375529994005	Incident on I-270 EB near IL-157 Road closed. Take alt
6	A-6482235	2	42.869929	-77.885846	42.869974	-77.881867	Incident on N CHESTNUT ST near E MAIN ST Drive wit
7	A-6482296	2	43.018	-73.796061	42.96752100000001	-73.801911	Crash on I-87 Northway southbound between Exit 13S
8	A-6482394	2	44.975895	-93.281223	44.975915	-93.281587	Accident from N 12th St / Hawthorne Ave to I-394 E.
9	A-6482475	2	40.772534	-73.674201	40.781578	-73.649528	Slow traffic on Northern State Pkwy E from Shelter Roc

Spark UI :



APACHE
Spark
3.5.6

Jobs

Stages

Storage

Environment

Executors

SQL / DataFrame

datamart - datamarts.py application UI

Stages for All Jobs

Active Stages: 1
Completed Stages: 15
Skipped Stages: 10

Active Stages (1)

Page: 11 Pages. Jump to 1. Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
25	jdbc at NativeMethodAccessorImpl.java:0	+details (kill) 2025/06/29 15:33:56	24 s	0/1 (1 running)				

Page: 11 Pages. Jump to 1. Show 100 items in a page. Go

Completed Stages (15)

Page: 11 Pages. Jump to 1. Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
24	jdbc at NativeMethodAccessorImpl.java:0	+details 2025/06/29 15:33:55	0.2 s	1/1			6.6 KB	
22	jdbc at NativeMethodAccessorImpl.java:0	+details 2025/06/29 15:33:55	0.3 s	1/1	2.9 MiB			6.6 KB
21	jdbc at NativeMethodAccessorImpl.java:0	+details 2025/06/29 15:33:53	1.0 s	6/6			425.0 B	
14	jdbc at NativeMethodAccessorImpl.java:0	+details 2025/06/29 15:33:53	0.1 s	1/1	228.4 KB			71.0 B
13	jdbc at NativeMethodAccessorImpl.java:0	+details 2025/06/29 15:33:53	0.1 s	1/1	212.4 KB			72.0 B
12	jdbc at NativeMethodAccessorImpl.java:0	+details 2025/06/29 15:33:53	93 ms	1/1	186.3 KB			70.0 B
11	jdbc at NativeMethodAccessorImpl.java:0	+details 2025/06/29 15:33:53	93 ms	1/1	163.8 KB			70.0 B
10	jdbc at NativeMethodAccessorImpl.java:0	+details 2025/06/29 15:33:52	0.2 s	1/1	145.2 KB			70.0 B
9	jdbc at NativeMethodAccessorImpl.java:0	+details 2025/06/29 15:33:52	0.3 s	1/1	168.8 KB			72.0 B
8	jdbc at NativeMethodAccessorImpl.java:0	+details 2025/06/29 15:33:51	0.2 s	1/1			11.8 KB	
6	jdbc at NativeMethodAccessorImpl.java:0	+details 2025/06/29 15:33:50	0.7 s	1/1	2.1 MiB			11.8 KB

APACHE
Spark
3.5.6

Jobs

Stages

Storage

Environment

Executors

SQL / DataFrame

datamart - datamarts.py application UI

SQL / DataFrame

Running Queries: 1
Completed Queries: 5

Running Queries (1)

Page: 11 Pages. Jump to 1. Show 100 items in a page. Go

ID	Description	Submitted	Duration	Running Job IDs	Succeeded Job IDs	Failed Job IDs	Sub Execution IDs
9	jdbc at NativeMethodAccessorImpl.java:0	2025/06/29 15:33:56	40 s				[10]

Page: 11 Pages. Jump to 1. Show 100 items in a page. Go

Completed Queries (5)

Page: 11 Pages. Jump to 1. Show 100 items in a page. Go

ID	Description	Submitted	Duration	Job IDs	Sub Execution IDs
7	jdbc at NativeMethodAccessorImpl.java:0	2025/06/29 15:33:55	0.9 s		[8]
5	jdbc at NativeMethodAccessorImpl.java:0	2025/06/29 15:33:52	3 s		[6]
3	jdbc at NativeMethodAccessorImpl.java:0	2025/06/29 15:33:50	2 s		[4]
1	jdbc at NativeMethodAccessorImpl.java:0	2025/06/29 15:33:43	7 s		[2]
0	count at NativeMethodAccessorImpl.java:0	2025/06/29 15:33:40	3 s	[0][1]	

Page: 11 Pages. Jump to 1. Show 100 items in a page. Go

Hadoop UI – YARN :

Cluster Metrics

2	Apps Submitted	0	Apps Pending	1	Apps Running	1	Apps Completed	1	Containers Running	1 GB	Memory Used	8 GB	Memory Total	0 B	Memory Reserved	1	Vcores Used	8	VCore
---	----------------	---	--------------	---	--------------	---	----------------	---	--------------------	------	-------------	------	--------------	-----	-----------------	---	-------------	---	-------

Cluster Nodes Metrics

1	Active Nodes	0	Decommissioning Nodes	0	Decommissioned Nodes	0	Lost Nodes	0	Unhealthy Nodes	0	Rebooted Nodes
---	--------------	---	-----------------------	---	----------------------	---	------------	---	-----------------	---	----------------

Scheduler Metrics

Scheduler Type		Scheduling Resource Type		Minimum Allocation		Maximum Allocation		Maximum Clu	
Capacity Scheduler		[memory-mb (unit=Mi), vcores]		<memory:1024, vCores:1>		<memory:8192, vCores:4>		0	

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU Vcores	Allocated Memory MB	Reserved CPU Vcores	Reserved Memory MB	% of Queue	% of Cluster	Pro
application_1751214151276_0002	root	datamart-datamarts.py	SPARK	default	0	Sun Jun 29 19:04:01 +0200 2025	N/A	N/A	ACCEPTED	UNDEFINED	1	1	1024	0	0	12.5	12.5	

Cluster Metrics

2	Apps Submitted	0	Apps Pending	0	Apps Running	2	Apps Completed	0	Containers Running	0 B	Memory Used	8 GB	Memory Total	0 B	Memory Reserved	0	Vcores Used	0	
---	----------------	---	--------------	---	--------------	---	----------------	---	--------------------	-----	-------------	------	--------------	-----	-----------------	---	-------------	---	--

Cluster Nodes Metrics

1	Active Nodes	0	Decommissioning Nodes	0	Decommissioned Nodes	0	Lost Nodes	0	Unhealthy Nodes	0	Rebooted Nodes
---	--------------	---	-----------------------	---	----------------------	---	------------	---	-----------------	---	----------------

Scheduler Metrics

Scheduler Type		Scheduling Resource Type		Minimum Allocation		Maximum Allocation		Ma	
Capacity Scheduler		[memory-mb (unit=Mi), vcores]		<memory:1024, vCores:1>		<memory:8192, vCores:4>		0	

Show 20 entries

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU Vcores	Allocated Memory MB	Reserved CPU Vcores	Reserved Memory MB	% of Queue	% of Cluster	Pro
application_1751214151276_0002	root	datamart-datamarts.py	SPARK	default	0	Sun Jun 29 19:04:01 +0200 2025	Sun Jun 29 19:04:02 +0200 2025	Sun Jun 29 19:04:23 +0200 2025	FINISHED	SUCCEEDED	N/A	N/A	N/A	N/A	N/A	N/A	0.0	

Détails sur la partie API :

main_api.py - API FastAPI pour exposer les données du datamart

Quand l'exécuter

Ce script peut être exécuté **une fois que** la table gold_final_accidents a été générée et exportée dans PostgreSQL par datamarts.py. Il permet d'interroger les données via une API HTTP REST.

But

Offrir un point d'accès API pour explorer les données depuis le datamart PostgreSQL.

Fonctionnalités

- Connexion à la base datamart dans PostgreSQL via SQLAlchemy
- Lecture automatique de la table gold_final_accidents
- Endpoint /table/ avec pagination (page et limit)
- Résultat JSON formaté ligne par ligne

Démarrage de l'API

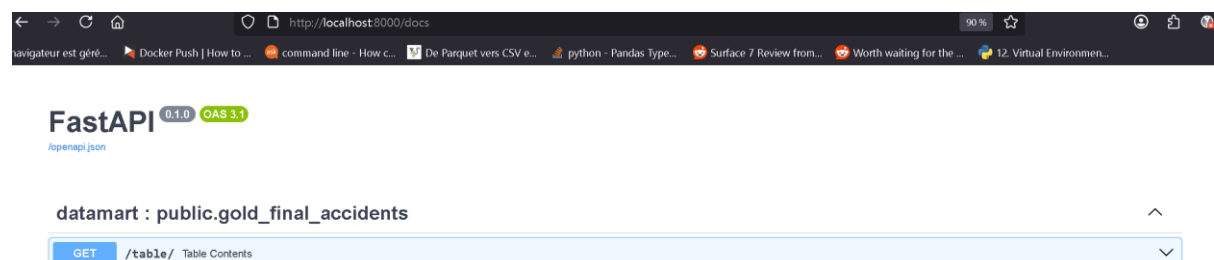
Il faut d'abord entrer dans le container : spark

```
docker exec -it projet_bdf3 bash
```

Puis lancer le serveur uvicorn

```
uvicorn api.main:app --host 0.0.0.0 --port 8000 --reload
```

FastAPI UI :



datamart : public.gold_final_accidents

GET /table/ Table Contents

Parameters

Name	Description
page integer (query)	4
limit integer (query)	10

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8000/table/?page=4&limit=10' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:8000/table/?page=4&limit=10
```

Server response

Code Details

200

Response body

```
{
  "ID": "A-6482223",
  "Severity": 2,
  "Start_Lat": 38.756514787489145,
  "Start_Lng": -90.61391529994005,
  "End_Lat": 38.75635478748913,
  "End_Lng": -90.60375519994005,
  "Description": "Incident on I-270 EB near IL-157 Road closed. Take alternate route.",
  "Street": "I-270",
  "City": "Glen Carbon",
  "State": "IL",
  "Zipcode": "62034",
  "Timezone": "US/Central",
  "Airport_Code": "KALN",
  "Handicap": 0,
  "Pressure(in)": 29.83,
  "Weather_Condition": "Partly Cloudy",
  "Assemt": false,
  "Bump": false,
  "Crossing": false,
  "Give_Way": false,
  "Junction": false,
  "No_Exit": false,
  "Railway": false,
  "Roundabout": false,
  "Station": false,
  "Stop": false,
}
```

Response headers

```
content-length: 9974
content-type: application/json
date: Sun, 29 Jun 2025 17:21:08 GMT
server: unicorn
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

Business values :

Pour exploiter les données enrichies dans **Tableau**, nous avons établi une connexion à une base de données **PostgreSQL** locale. En renseignant les informations nécessaires (serveur localhost, port 5432, base datamart, identifiants d'accès), nous avons pu accéder aux différentes tables exportées depuis Spark. Cela permet une **visualisation directe des datamarts** depuis Tableau, avec des mises à jour dynamiques si les données évoluent côté base.

PostgreSQL

Général

SQL initial

Serveur

localhost

Port

5432

Base de données

datamart

Authentification

Nom d'utilisateur et mot de passe

Nom d'utilisateur

postgres

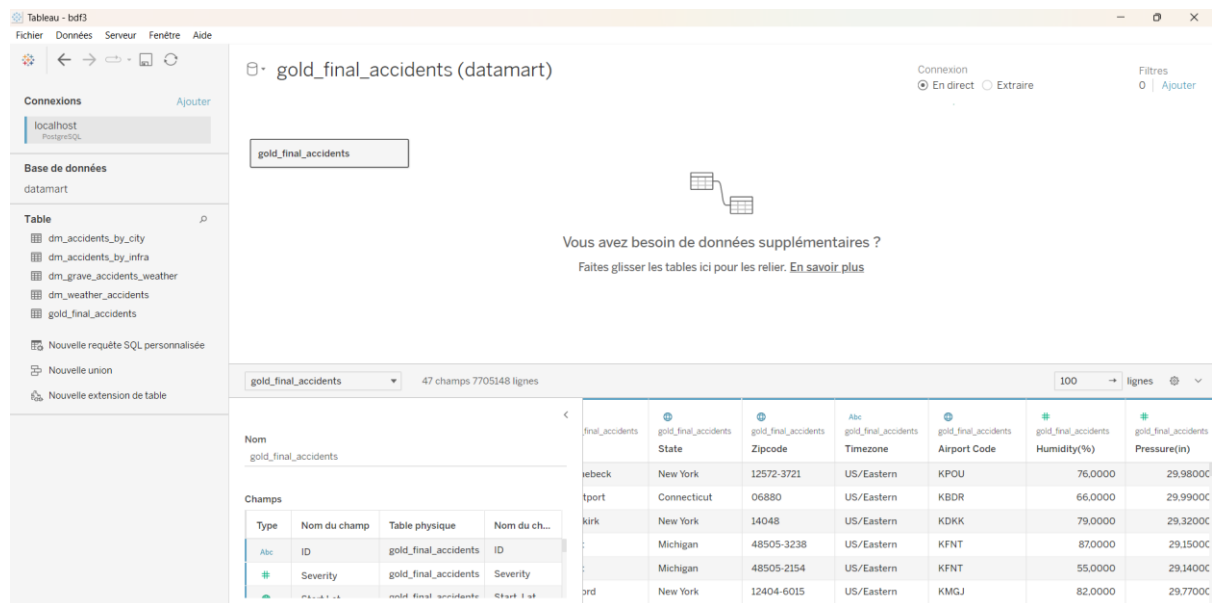
Mot de passe

.....

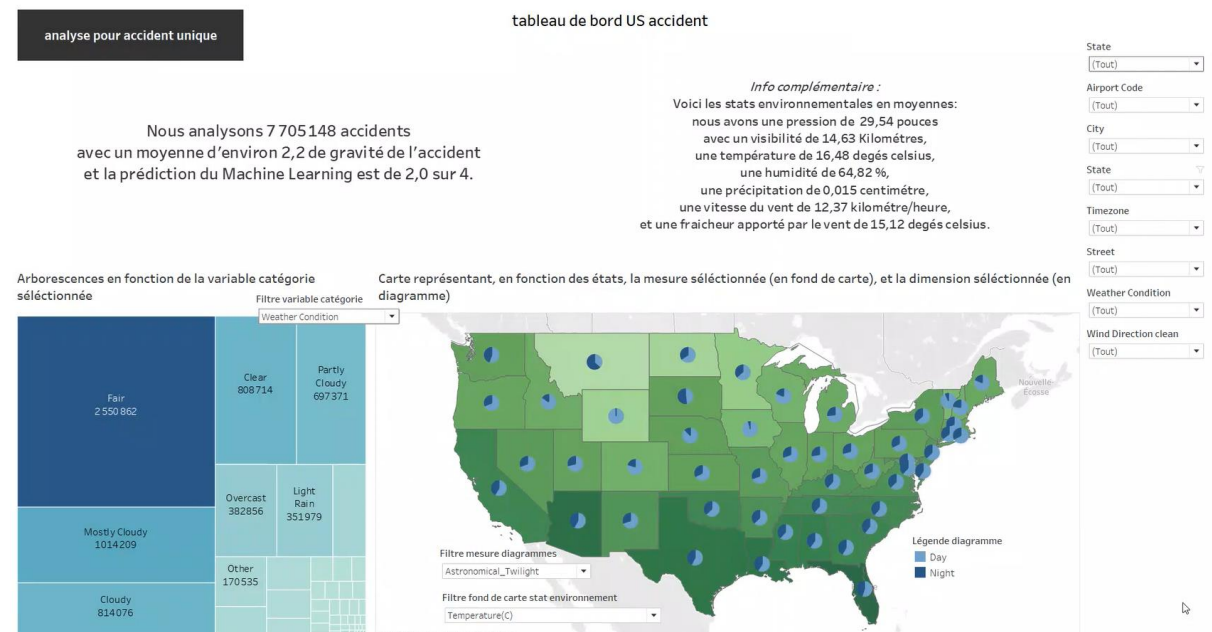
☐ Nécessite SSL

Connexion

Une fois la connexion établie, nous utilisons la datamart `gold_final_accidents` comme source principale. Elle regroupe plus de **7,7 millions d'accidents** enrichis avec des informations géographiques, environnementales et une **prédiction de la gravité** issue du modèle de Machine Learning. Cette table, connectée en direct à PostgreSQL, permet de construire des visualisations dynamiques (cartes, indicateurs, arborescences) et d'explorer tous les axes d'analyse disponibles dans les datamarts thématiques à gauche.



Dashboards :



Utilités :

Pour les **agences de sécurité routière / collectivités locales** :

- **Ciblage intelligent** des zones à risque selon les données météo/horaire → ex : déploiement de signalisation intelligente, limitation de vitesse dynamique dans les zones de brouillard, campagnes de sensibilisation météo-dépendantes.
- **Investissements plus efficaces** : grâce aux datamarts construits (accidents par ville, par infrastructure...), les budgets de sécurité peuvent être **priorisés selon la gravité moyenne**.
- **Appui à la planification urbaine** : mise en évidence des situations aggravantes (ex : intersections mal signalées) via les colonnes booléennes (Stop, Railway, Junction...).

☒ Pour les **assureurs / réassureurs** :

- Utiliser les **modèles prédictifs (ML)** pour mieux **segmenter le risque par région, météo et comportement**.
- Affiner les **modèles actuariels** avec des variables climatiques réelles → au lieu de se baser uniquement sur des historiques sinistres.

Recommandation stratégique

Mettre en œuvre un système d'alerte préventive et adaptative basé sur les prédictions de gravité d'accident issues de l'analyse de données

Objectif :

Réduire la survenue d'accidents graves (gravité 3 ou 4) grâce à des **interventions ciblées et contextualisées** déclenchées par des conditions à risque identifiées automatiquement via le modèle.

1. Cartographie dynamique des zones à risque

- Exploiter les résultats du **modèle prédictif** pour localiser, par État ou ville, les **zones les plus susceptibles de générer des accidents graves** en fonction de :
 - l'heure de la journée,
 - les conditions météo (visibilité, pluie, vent...),
 - l'absence de signalisation ou d'infrastructure,
 - la présence de carrefours dangereux (Junction, Stop, etc.).

2. Déclenchement d'alertes intelligentes

- En couplant ces prédictions à des flux **temps réel** (API météo, capteurs routiers, caméras, données IoT), on peut :
 - **Envoyer une alerte aux conducteurs** via Waze, Google Maps, applications de covoiturage.
 - **Activer une signalisation lumineuse temporaire** (panneaux à messages variables, flashes LED dans les zones à visibilité réduite).
 - **Réduire dynamiquement la vitesse maximale** autorisée dans une zone donnée.

3. Actions correctives d'infrastructure

- Utiliser les datamarts exportés pour prioriser les budgets d'investissement :
 - Ajouter des feux aux intersections non protégées les plus dangereuses.
 - Repenser la visibilité ou l'éclairage sur les routes rurales fréquemment touchées.

Mettre en place des ralentisseurs, des ronds-points ou des zones tampons là où la prédiction moyenne est élevée.

Récapulatif de l'ordre d'exécution des différents scripts

:

```
projet_bdf3/
└─ app/
    └─ feeder/
        └─ splitter.py
        └─ save_hdfs.py
        └─ main.py
        └─ verif_jour.py
    └─ processor/
        └─ pre_traitement.py
        └─ join_ml.py
    └─ ml/
        └─ severity_prediction.py
    └─ datamart/
        └─ datamarts.py
    └─ api/
        └─ main.py
    └─ logs/
    └─ bronze_data/

<-- Étape 1 : Ingestion brute vers hdfs
# 1. Découpe du CSV brut en fichiers par date
# 2. Enregistrement des fichiers Parquet dans Le HDFS (Bronze)
# 3. Point d'entrée : appelle splitter + save_hdfs
# 4. Vérifie Les dates déjà présentes dans Bronze

<-- Étapes 2 et 4 : Nettoyage puis transfert vers Table hive
# 5. Nettoyage et enrichissement → Hive :
#   - preprocessed_accidents_for_ml
#   - preprocessed_accidents_for_join
# 7. Jointure finale avec Les prédictions ML → Hive :

final_joined_accidents

<-- Étape 3 : Modélisation
# 6. Entraîne un modèle Random Forest et prédit La gravité → Hive :

predicted_severity_full

<-- Étape 5 : Construction des marts dans PostgreSQL
# 8. Génère Les data marts et exporte dans PostgreSQL :
#   - dm_accidents_by_city
#   - dm_weather_accidents
#   - dm_accidents_by_infra
#   - dm_grave_accidents_weather
#   - gold_final_accidents

<-- Étape 6 : API d'exposition des données finales
# 9. Expose gold_final_accidents via FastAPI (route `/table/`)
```