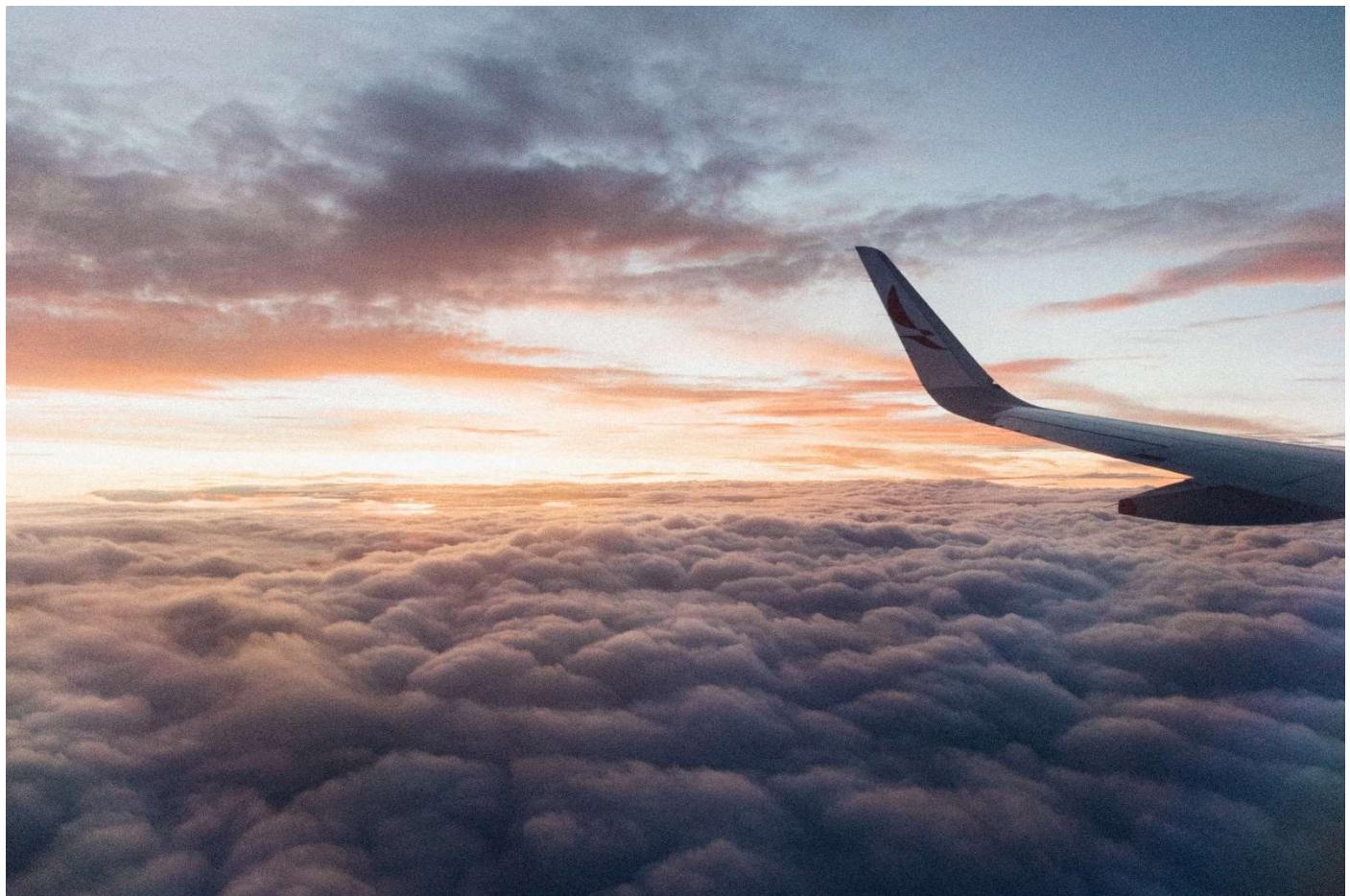


PROJET FLIGHT



[Photo de Caroline Cagnin sur Pexels](#)

Table des matières

Introduction	3
1. Architecture et plateformes utilisées	4
1.1. Tables bronze, silver & gold	4
1.2. Cluster Hadoop / Spark du Lamsade.....	4
1.3. Databricks	5
1.4. Google Cloud Platform.....	5
1.5. Environnement local	8
2. Preprocessing : tables silver.....	14
2.1. Premier aperçu des datasets	14
2.2. Prétraitements des données de vols	15
2.3. Prétraitements des données météo	16
2.4. Prétraitements des données wban	16
3. Analyses exploratoires	17
3.1. Statistiques et analyse des données de vols.....	17
3.2. Statistiques et analyse des données de météo.....	20
3.3. Analyse combinée	21
4. Feature engineering	25
4.1. Préparation des données de vols pour les modèles	25
4.2. Préparation des données de météo pour les modèles.....	27
4.3. Jointure et sélection de features	28
5. Modèles de Machine Learning.....	30
5.1. Metrics & proof of concept.....	30
5.2. Model final, mise en production.....	35
5.3. Améliorations potentielles.....	35
Conclusion.....	37
Annexes.....	38

Introduction

Afin de valider et consolider un certain nombre de connaissances acquises durant cette formation, il nous a été proposé de travailler sur le projet Flight.

L'idée générale est de construire un modèle prédictif capable de prévoir les retards d'avions dus à une mauvaise météo. De manière générale, de nombreux vols sont impactés par des retards engendrant un cout annuel de plusieurs milliards de pertes à la fois pour les compagnies comme pour les voyageurs. Une partie non négligeable de ces retards peuvent être imputée aux mauvaises conditions météorologiques. Être en mesure de les anticiper est donc un enjeu important.

Les données sont fournies et portent sur le marché US. Elles couvrent de nombreux vols étalés sur plusieurs mois et ; par conséquent ; sont très volumineuses. Ce qui nécessite une approche « big data » qui distribue et parallélise autant le stockage que les calculs : nous utiliserons donc le framework Spark.

Le document de référence "*Using Scalable Data Mining for Predicting Flight Delays*" (L. Belcastro, Fabrizio Marozzo, Domenico Talia, Paolo Trunfio) traite de ce sujet et nous permettra de nous en inspirer, afin d'en reproduire l'analyse, ainsi que de mieux comprendre les données (Annexe 1)

L'ensemble des développements sont menés dans le language Scala et demeurent accessible dans le repository Github suivant : <https://github.com/YoloCall/FlightFinal> .

Dans un premier temps, nous réaliserons l'exploration, le nettoyage et la transformation des données. Ensuite il faudra sélectionner et joindre ces données de manière pertinente. Par la suite, la construction de différents modèles prédictifs fera l'objet d'un paragraphe spécifique ainsi que leurs évaluations, interprétations et comparaison. Enfin nous serons amenés à conclure en évoquant les difficultés rencontrées et les pistes non explorées.

1. Architecture et plateformes utilisées

1.1. Tables bronze, silver & gold

Pour ce projet, nous avons suivi une architecture data avec des tables « bronze », « silver » et « gold ».



La couche **Bronze** accueille toutes les données en provenance des systèmes externes : les data brutes stockées soit dans HDFS (pour le cluster du Lamsade), soit dans Google Cloud Storage (pour GCP) soit dans le système de fichiers présent dans Databricks (cf. ci-dessous).

Dans la couche **Silver**, les données de la couche Bronze sont identifiées, fusionnées, mises en conformité et nettoyées (selon le principe du « strict nécessaire ») de façon à délivrer une « vue entreprise ».

Elle permet de générer des rapports ad-hoc et de réaliser des opérations d'analytique et sert de source aux analystes, aux data engineers ainsi qu'aux data scientists.

Les données de la couche **Gold** sont généralement organisées en bases de données dédiées à des projets spécifiques et prêtes à consommer. La couche Gold est destinée à la création de rapports. C'est là que sont appliquées les dernières transformations et règles de qualité.

Les données sont affinées à chaque passage d'une couche à l'autre.

1.2. Cluster Hadoop / Spark du Lamsade

Les différents scripts scala fournis en annexes qui ont permis la réalisation des différentes couches de données présentées précédemment ont été lancés sur le cluster du Lamsade : il s'agit d'un cluster Hadoop comprenant entre autres HDFS, Hive et Spark.

Après avoir ajouté notre clé SSH privée, on se connecte au master node en ligne de commande. Les fichiers bruts sont déjà présents dans HDFS. Ce cluster vient avec la version 3.2.0 de Spark et son shell dédié :

```
obrunet@vmhadoopmaster:~$ spark-shell --version
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/cephfs/shared/spark-3.2.0-bin-hadoop2.7/jars/slf4j-log4j12-
\StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/cephfs/shared/hadoop-2.10.1/share/hadoop/common/lib/slf4j-
f4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Welcome to

          _/ _/ 
         / \ \ \ / -\ / / / \ / 
         / \ \ .- \ / / / / / \ \ 
          /_ /             version 3.2.0

Using Scala version 2.12.15, OpenJDK 64-Bit Server VM, 1.8.0_212
Branch HEAD
Compiled by user ubuntu on 2021-10-06T13:20:32Z
Revision 5d45a415f3a29898d92380380cf82bfc7f579ea
Url https://github.com/apache/spark
Type --help for more information.
obrunet@vmhadoopmaster:~$
```

1.3. Databricks

Nous avons également utilisé la version community édition de Databricks en créant un cluster avec les différents paramètres de configuration suivants:

1 Driver: 15.3 GB Memory, 2 Cores, 1 DBU (A Databricks Unit is a normalized unit of processing power on the Databricks Lakehouse Platform used for measurement and pricing purposes. The number of DBUs a workload consumes is driven by processing metrics, which may include the compute resources used and the amount of data processed)

1.4. Google Cloud Platform

Pour terminer nous avons également utilisé GCP et les différents services suivants :

- Google Cloud Storage (GCS) : afin de déposer les données bronze dans un bucket spécifique.
- BigQuery : pour créer les tables silver et gold, et mener des requêtes SQL à des fins d'analytics.
- Dataproc (cluster Hadoop / Spark) : en guise d'ETL et pour passer d'une couche à l'autre, également à des fins d'analytics pour appréhender les datasets ainsi que pour la partie machine learning.

Différentes façons de lancer des jobs :

- notebook (for the developpement / analysis part)
- via la web UI

The screenshot shows the Google Cloud DataProc interface. On the left, there's a sidebar with sections like 'Jobs on clusters', 'Clusters', 'Jobs', 'Workflows', 'Auto-scaling policies', 'Serverless', 'Batches', 'Metastore services', 'Metastore', 'Federation', and 'Utilities'. The main area shows 'Cluster details' for 'node-2'. It includes fields for 'Name' (node-2), 'Cluster UUID' (12453932-b4dc-4cc6-a466-bd67d313c20e), 'Type' (Dataproc cluster), and 'Status' (Running). Below this is a monitoring dashboard with tabs for 'MONITORING', 'JOBS', 'VM INSTANCES', 'CONFIGURATION', and 'WEB INTERFACES'. The 'MONITORING' tab displays YARN memory usage with a chart showing 12GB used and 10GB available. On the right, there's a 'Submit a job' form. It has fields for 'Job ID' (node-2), 'Job type' (PySpark), 'Main python file' (gs://iasd-input-data/compute_CCF_with_RDD_and_DF.py), 'Additional python files', 'Jar files', and 'Archive files'. A note at the top of the form says: 'For PD-Standard without local SSDs, we strongly recommend provisioning 1TB or larger to ensure consistent performance. For more information, see https://cloud.google.com/compute/docs/disks/performance'.

On peut ajouter des propriétés pour spécifier la quantité de mémoire de l' « exécuteur », le nombre de cœur:

Properties ?

Key 1 * spark.executor.memory	Value 1 3g
Key 2 * spark.driver.memory	Value 2 4g
Key 3 * spark.executor.cores	Value 3 1

- la commande équivalente à spark submit dans GCP est la suivante (les variables shell commençant par \$ devant être initialisées au préalable ainsi que le chemin path_main.scala changed par l'URL du script dans le bucket GCS : bucket: gs://flight-project/random_forest.scala

```
gcloud dataproc jobs submit spark path_main.scala \
--cluster=$CLUSTER_NAME \
--region=$REGION \
--properties="spark.submit.deployMode=cluster", \
"spark.dynamicAllocation.enabled=true", \
"spark.shuffle.service.enabled=true", \
"spark.executor.memory=15g", \
"spark.driver.memory=16g", \
"spark.executor.cores=5"
```

Une fois le « job » soumis nous pouvons voir son status dans l'interface Yarn de l'application manager :

iasd4-364813 > node-2 Sign out

 All Applications

Cluster Metrics														
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Used Resources	Total Resources	<memory:12 GB, vCores:4>							
1	0	1	0	3	<memory:6.75 GB, vCores:2>	<memory:12 GB, vCores:4>	<memory:12 GB, vCores:4>							
Cluster Nodes Metrics														
Active Nodes	Decommissioning Nodes			Decommissioned Nodes			Lost Nodes		Unhealthy !					
2	0			0			0		0					
Scheduler Metrics														
Scheduler Type	Scheduling Resource Type			Minimum Allocation			Maximum Allocation							
Capacity Scheduler	[memory-mb (unit=Mi), vcores]			<memory:1, vCores:1>			<memory:6144, vCores:2>			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	Allocated GPUs
application_1666449416260_0001	root	PySpark App by Olivier & Jean-Louis	SPARK	default	0	Sat Oct 22 16:54:02 +0200 2022	N/A	N/A	RUNNING	UNDEFINED	2	2	6912	-1

Et obtenir plus de détails et d'informations relatives à Spark (stages, jobs...)

iasd4-364813 > node-2 Sign out

 Jobs Stages Storage Environment Executors SQL PySpark App by Olivier & Jean-Louis application UI

Spark Jobs (?)

User: root
Total Uptime: 7.7 min
Scheduling Mode: FAIR
Active Jobs: 1
Completed Jobs: 100

Event Timeline

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
100	sortByKey at /tmp/test-bis/compute_CCF_with_RDD_and_DF.py:104 sortByKey at /tmp/test-bis/compute_CCF_with_RDD_and_DF.py:104 (kill)	2022/10/22 15:01:02	38 s	1/5	7/28 (1 running)

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

Completed Jobs (100)

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
99	sortByKey at /tmp/test-bis/compute_CCF_with_RDD_and_DF.py:104 sortByKey at /tmp/test-bis/compute_CCF_with_RDD_and_DF.py:104	2022/10/22 15:00:53	9 s	1/1 (1 skipped)	4/4 (4 skipped)
98	sortByKey at /tmp/test-bis/compute_CCF_with_RDD_and_DF.py:104 sortByKey at /tmp/test-bis/compute_CCF_with_RDD_and_DF.py:104	2022/10/22 15:00:15	38 s	2/2	8/8
97	count at NativeMethodAccessorImpl.java:0	2022/10/22 15:00:15	23 ms	1/1 (16 skipped)	1/1 (65 skipped)

1.5. Environnement local

Nous avons monté un environnement de développement en local afin de pallier les limites de ressource du cloud et avoir plus de liberté. La machine en local, sous OS X, dispose de 32 Go de RAM et 16 cores qui peuvent être répartis et parallélisés, nous le verrons plus bas sur la configuration de spark et de son spark-submit.

L'environnement utilisé est le suivant :

- JAVA :

Langage de programmation largement utilisé dans le contexte de Spark et Scala, notamment grâce à sa JVM (Java Virtual Machine) qui est le moteur d'exécution sous-jacent. Spark est construit sur Scala, qui est compatible avec la JVM, et qui s'exécute sur la JVM. Cela signifie que lorsqu'on écrit du code Spark en Scala, ce code est exécuté dans la JVM, profitant ainsi de ses avantages en termes de performances, de gestion de la mémoire et de portabilité.

Il faut faire attention par rapport à la version utilisée car elle aura un impact sur les autres technologies, dans notre cas la version 8 convient avec le reste.

- Scala :

Langage de programmation conçu pour être à la fois fonctionnel et orienté objet. Il est compatible avec la machine virtuelle Java (JVM), ce qui permet de bénéficier d'une vaste gamme de bibliothèques et d'outils disponibles dans l'écosystème Java. Il est intégré à Spark pour distribuer les calculs et donc à Hadoop pour distribuer les données.

Afin d'être iso avec le cluster du Lamsade, la version installée est en 2.12.15

L'installation se fait assez facilement sur OS X depuis un 'brew install scala@<version>' et en ajoutant le bin dans les variables d'environnement.

- Spark :

Framework de traitement de données distribués, il traite de grands ensembles de données à l'aide de clusters. Il est conçu pour offrir une performance élevée et une grande évolutivité pour le traitement de données à grande échelle en effectuant des opérations sur les données en parallèle.

Spark-submit est une commande qui permet de soumettre une application Spark divisée et exécutée en plusieurs tâches sur des nœuds du cluster en parallèle. L'application Spark est compilée puis packagée grâce à Maven et exécutée par un spark-submit disposant de paramètres tels que : le nombre d'instances, le nombre de cores, le nombre d'executeurs etc.

D'après la doc Spark, la version fonctionnant avec Scala 2.12 et utilisant Java 8 est la 3.3.x.

L'installation se fait en suivant les documentations :

- <https://spark.apache.org/docs/3.3.2/index.html>
- <https://spark.apache.org/docs/3.3.2/spark-standalone.html>

Une fois installé nous pouvons disposer de la technologie soit depuis un Shell :

```
[mathisperez@macbook-pro-de-mathis ~ % spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/11/04 17:23:27 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Spark context Web UI available at http://macbook-pro-de-mathis.home:4040
Spark context available as 'sc' (master = local[*], app id = local-1699115007744).
Spark session available as 'spark'.
Welcome to
    __|__/\_/\_\_/\_\_/\_\_
   / \ / .--/\_\_/\_\_/\_\_/\_\_
   /_/
version 3.3.2

Using Scala version 2.12.15 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_371)
Type in expressions to have them evaluated.
Type :help for more information.
```

Soit en mode Standalone permettant d'avoir un environnement d'exécution configurable suivant la machine et un suivi des jobs :

Le master :

The screenshot shows the Apache Spark 3.3.2 Master UI. It includes sections for Workers (1), Running Applications (0), Running Drivers (0), and Completed Applications (2). The Completed Applications section lists two entries: 'app-20231104165547-0001' and 'app-20231104163913-0000', both of which are FINISHED with a duration of 6,8 min. The Completed Drivers section lists one entry: 'driver-20231104165544-0001', which is FINISHED with a duration of 6,8 min.

Worker ID	Address	State	Cores	Memory	Resources
worker-20231104163223-[REDACTED]	[REDACTED]	ALIVE	16 (0 Used)	31.0 GiB (0.0 B Used)	

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20231104165547-0001	Flight	15	10.0 GiB		2023/11/04 16:55:47	mathisperez	FINISHED	6,8 min
app-20231104163913-0000	Flight	15	10.0 GiB		2023/11/04 16:39:13	mathisperez	FINISHED	6,8 min

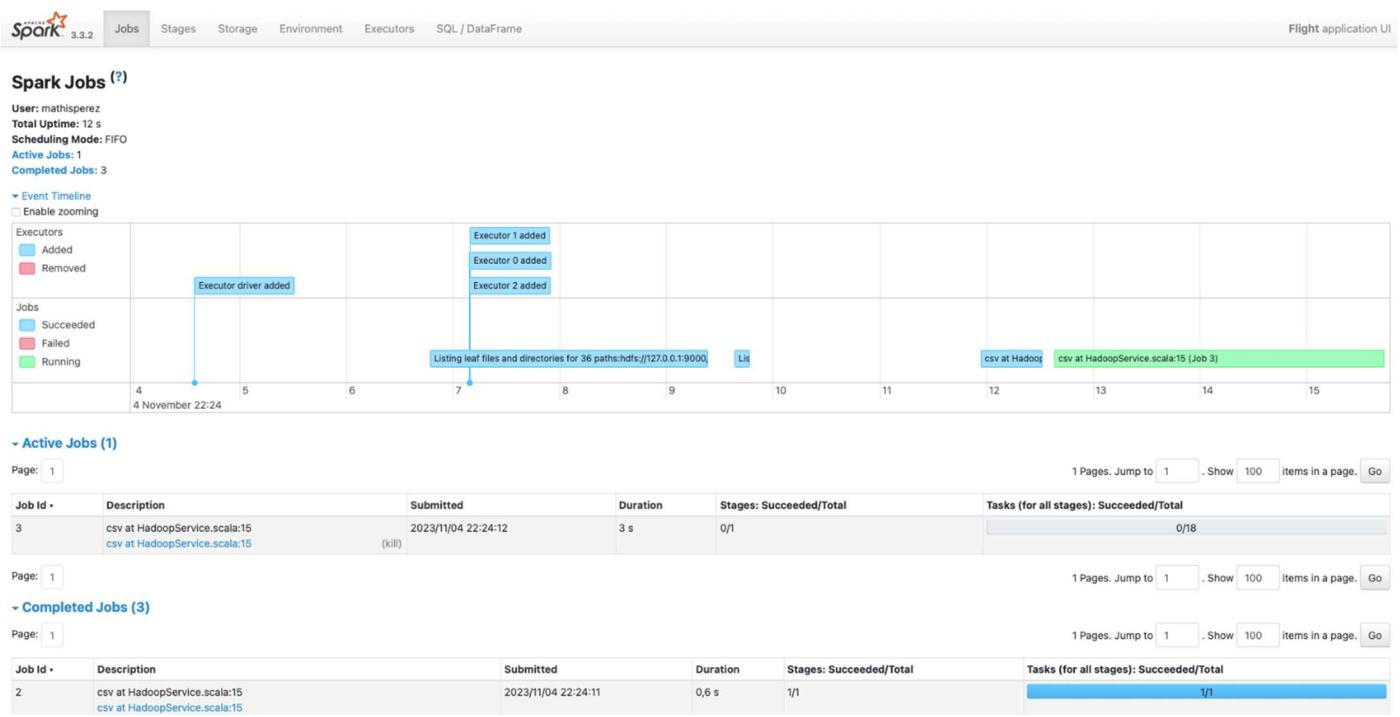
Submission ID	Submitted Time	Worker	State	Cores	Memory	Resources	Main Class	Duration
driver-20231104165544-0001	2023/11/04 16:55:44	worker-20231104163223-[REDACTED]	FINISHED	1	1024.0 MiB		com.dauphine.flight.Application	
driver-20231104163910-0000	2023/11/04 16:39:10	worker-20231104163223-[REDACTED]	FINISHED	1	1024.0 MiB		com.dauphine.flight.Application	

Le Worker :

The screenshot shows the Apache Spark 3.3.2 Worker UI. It includes sections for Running Executors (0) and Finished Executors (6). The Finished Executors section lists three executors (ID 2, 1, 0) that were KILLED. Each executor has its job details listed: ID: app-20231104163913-0000, Name: Flight, User: mathisperez. The logs for each executor are listed as 'stdout stderr'.

ExecutorID	State	Cores	Memory	Resources	Job Details	Logs
2	KILLED	5	10.0 GiB		ID: app-20231104163913-0000 Name: Flight User: mathisperez	stdout stderr
1	KILLED	5	10.0 GiB		ID: app-20231104163913-0000 Name: Flight User: mathisperez	stdout stderr
0	KILLED	5	10.0 GiB		ID: app-20231104163913-0000 Name: Flight User: mathisperez	stdout stderr

Une application en cours d'exécution :



Une fois le code compilé dans un .jar (voir le point sur Maven) il va pouvoir être exécuté sur la machine depuis un spark-submit et son plan d'exécution sera visible comme ci-dessus. Exemple :

```
spark-submit \
--master spark://macbook-pro-de-mathis.home:7077 \
--deploy-mode cluster \
--conf spark.driver.cores=1 \
--conf spark.driver.memory=1G \
--conf spark.executor.instances=3 \
--conf spark.executor.cores=5 \
--conf spark.executor.memory=10G \
--class com.dauphine.flight.Application \
/Users/mathisperez/DevApp/ProjetFlight/target/ProjetFlight-1.0.0-SNAPSHOT.jar "remote"
```

Dans cet exemple nous exécutons un .jar du nom de ProjetFlight-1.0.0-SNAPSHOT.jar avec un argument « remote » pris par l'application. Cette exécution se fera sur le master en mode cluster.

La machine dispose de 32 Go de RAM et 16 cores, nous avons instancié le calcul sur 1 Driver avec 1 core et 1 Go de RAM. Il est conseillé d'en mettre minimum 2 Go de RAM mais 1 Go est pris par la version Standalone et nous voulions utiliser toutes les ressources. Étant donné que l'application print des calculs basiques 1 Go suffit, auquel cas l'application tomberait en erreur : OutOfMemory.

Le Driver va ensuite distribuer les tâches à 3 Exécuteurs, chacun avec 5 cores et 10 Go de RAM. Cette configuration n'est pas tuné compte tenu du temps que met l'application à être traitée (inférieur à 10 minutes) sur l'ensemble des données.

- Hadoop

Hadoop est un écosystème de logiciels open-source pour le traitement de données distribués. Le cœur de Hadoop est constitué de deux composants principaux : Hadoop Distributed File System (HDFS) et MapReduce.

HDFS est un système de fichiers distribués permettant de stocker de grandes quantités de données sur des clusters. MapReduce est un modèle de programmation qui permet de traiter des données en parallèle sur des clusters.

Les données sont stockées dans HDFS et traitées à l'aide de Spark en mémoire.

La version utilisée est la 3.3.5.

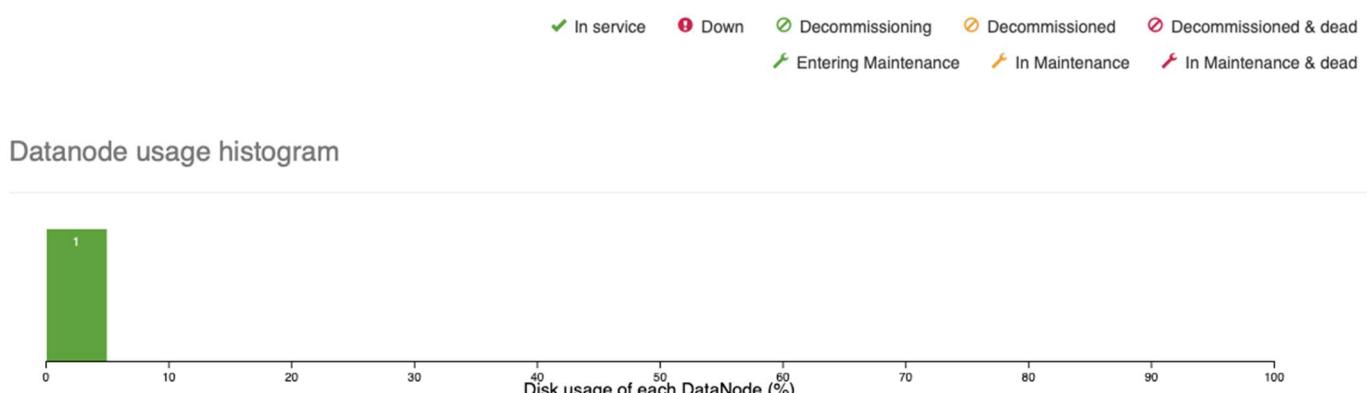
L'installation se fait principalement en configurant des fichiers de conf :

- <https://phoenixnap.com/kb/install-hadoop-ubuntu> (même process sur OS X)

Une fois l'installation faite, le Datanode et le Namenode démarré, nous pouvons déposer les données et les calculs intermédiaires sur HDFS :



Datanode Information



In operation

DataNode State	All	Show 25 entries	Search:						
Node	Http Address	Last contact	Last Block Report	Used	Non DFS Used	Capacity	Blocks	Block pool used	Version
✓ /default-rack/macbook-pro-de-	http://macbook-pro-de-	1s	171m	6.24 GB	357.19 GB	931.55 GB	178	6.24 GB (0.67%)	3.3.5

Showing 1 to 1 of 1 entries

Previous 1 Next

```
[mathisperez@macbook-pro-de-mathis ~ % hdfs dfs -ls /data
2023-11-04 17:22:57,003 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 4 items
drwxr-xr-x  - mathisperez supergroup          0 2023-10-26 20:23 /data/weather
drwxr-xr-x  - mathisperez supergroup          0 2023-11-04 17:02 /data/output
drwxr-xr-x  - mathisperez supergroup          0 2023-10-21 16:16 /data/wban
drwxr-xr-x  - mathisperez supergroup          0 2023-10-26 20:24 /data/flight
[mathisperez@macbook-pro-de-mathis ~ % hdfs dfs -ls /data/output
2023-11-04 17:23:03,338 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 6 items
drwxr-xr-x  - mathisperez supergroup          0 2023-11-04 16:59 /data/output/weather
drwxr-xr-x  - mathisperez supergroup          0 2023-11-04 16:56 /data/output/flight
drwxr-xr-x  - mathisperez supergroup          0 2023-11-04 17:00 /data/output/flightSilver
drwxr-xr-x  - mathisperez supergroup          0 2023-11-04 17:01 /data/output/gold
drwxr-xr-x  - mathisperez supergroup          0 2023-11-04 17:02 /data/output/predict
drwxr-xr-x  - mathisperez supergroup          0 2023-11-04 16:59 /data/output/airport
mathisperez@macbook-pro-de-mathis ~ % ]
```

Ainsi que récupérer les données stockées sur HDFS via spark-shell et les analyser, à défaut d'utiliser Hive dont l'installation est en erreur sur la machine :

```
[mathisperez@macbook-pro-de-mathis ~ % spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/11/04 17:23:27 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Spark context Web UI available at http://macbook-pro-de-mathis.home:4040
Spark context available as 'sc' (master = local[*], app id = local-1699115007744).
Spark session available as 'spark'.
Welcome to

    /---/-
   / \ \ - \ \ - \ \ - \ \ - \ \ - \
  / \ \ / . \ \ / \ \ / \ \ / \ \ / \ \ / \
 / \ \ / \ \ / \ \ / \ \ / \ \ / \ \ / \ \ / \
version 3.3.2

Using Scala version 2.12.15 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_371)
Type in expressions to have them evaluated.
Type :help for more information.

[scala] > val df_predict = spark.read.orc("/data/output/predict")
df_predict: org.apache.spark.sql.DataFrame = [DEP_AIRPORT_ID: int, ARR_AIRPORT_ID: int ... 121 more fields]
```

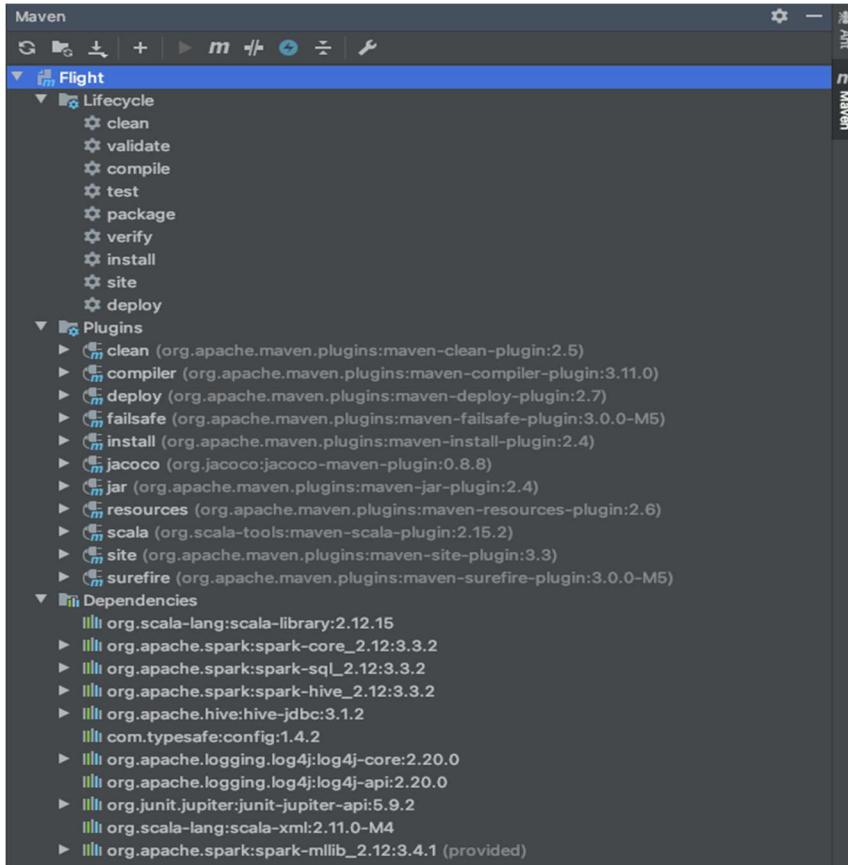
- Maven

Maven est un outil de gestion de projets open source largement utilisé pour la construction, la gestion et la distribution de projets Java.

Le fonctionnement de Maven est basé sur la notion de projets et de fichiers de configuration. Le fichier de configuration principal est le fichier pom.xml (Project Object Model) qui décrit les informations du projet, les dépendances, les plugins et les paramètres de construction. Maven utilise également un système de référentiel qui permet de stocker les artefacts générés et les dépendances du projet, facilitant ainsi leur gestion et leur distribution.

L'installation s'effectue, en récupérant le Binary et en le plaçant dans les variables d'environnement.

Notre projet dispose d'un pom.xml comprenant l'ensemble des dépendances et plugins utiles au fonctionnement de l'application, notamment les technologies citées plus haut et va permettre de build et packager dans un .jar le tout :



```

[INFO] --- maven-compiler-plugin:3.11.0:compile (default-compile) @ ProjetFlight ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-compiler-plugin:3.11.0:testCompile (scala-test-compile) @ ProjetFlight ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-scala-plugin:2.15.2:testCompile (scala-test-compile) @ ProjetFlight ---
[INFO] Checking for multiple versions of scala
[WARNING] Expected all dependencies to require Scala version: 2.12.15
[WARNING] com.dauphine.flight:ProjetFlight:1.0.0-SNAPSHOT requires scala version: 2.12.15
[WARNING] com.twitter.chill_2.12:0.10.0 requires scala version: 2.12.14
[WARNING] Multiple versions of scala libraries detected!
[INFO] includes = [**/*.java,**/*.scala,]
[INFO] excludes = []
[INFO] /Users/mathisperez/FlightFinal/src/test/scala/-1: info: compiling
[INFO] Compiling 7 source files to /Users/mathisperez/FlightFinal/target/test-classes at 1699137483907
[INFO] prepare-compile in 0 s
[INFO] compile in 3 s
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ ProjetFlight ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 832 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.11.0:testCompile (default-testCompile) @ ProjetFlight ---
[INFO] Changes detected - recompiling the module! :dependency
[INFO]
[INFO] --- maven-surefire-plugin:3.0.0-M5:test (default-test) @ ProjetFlight ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ ProjetFlight ---
[INFO] Building jar: /Users/mathisperez/FlightFinal/target/ProjetFlight-1.0.0-SNAPSHOT.jar
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.8:report (jacoco-site) @ ProjetFlight ---
[INFO] Skipping JaCoCo execution due to missing execution data file.
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 18.716 s
[INFO] Finished at: 2023-11-04T23:38:08+01:00
[INFO]

```

- Git

Afin de mutualiser notre code et travailler sans corrompre celui d'un collègue nous avons utilisé Git.

Nous avons instancié notre projet dans un dépôt commun sur GitHub avec chacun sa propre branche démarrant depuis la branche par défaut, la main :

Author	Commit Message	Date
feat/mathis	YoloCall	04/11/2023 06:36
feat/mathis	YoloCall	04/11/2023 06:31
main	YoloCall	04/11/2023 06:04
	YoloCall	04/11/2023 00:18
	YoloCall	01/11/2023 13:49
	YoloCall	31/10/2023 16:48
	YoloCall	04/11/2023 05:32
	YoloCall	04/11/2023 04:44
	YoloCall	04/11/2023 00:18
origin/feat/olivier	Olivier Brunet*	03/11/2023 14:59
origin/feat/olivier	Olivier Brunet*	03/11/2023 14:57
origin & feat/test	YoloCall*	03/11/2023 13:51
	Olivier Brunet*	01/11/2023 17:37
	YoloCall	01/11/2023 13:49
	YoloCall	31/10/2023 16:48
	Olivier Brunet*	30/10/2023 20:38
origin/feat/benjamin	Benlef92*	30/10/2023 12:30
	Benlef92*	30/10/2023 10:10
	Benlef92*	28/10/2023 18:56
	YoloCall	27/10/2023 20:12
	YoloCall	26/10/2023 20:43
	YoloCall	26/10/2023 13:57
	Olivier Brunet*	25/10/2023 20:19
	Olivier Brunet*	22/10/2023 11:29
	Olivier Brunet*	22/10/2023 11:29
	Olivier Brunet*	22/10/2023 11:29
	Olivier Brunet*	22/10/2023 11:28
	Olivier Brunet*	22/10/2023 11:28
	Olivier Brunet*	22/10/2023 11:28
	Olivier Brunet*	19/10/2023 22:07

2. Preprocessing : tables silver

2.1. Premier aperçu des datasets

- Flights

24 Fichiers fichiers csv donnant, mois par mois, les informations principales, ainsi que les différents types de retards, des vols aux USA sur les années 2012&2013.

Nombre total de lignes = 12 466 244

Voici les différentes colonnes :

FL_DATE:date	→ Date du vol au format 'YYYY-MM-DD'
OP_CARRIER_AIRLINE_ID:integer	→ Identifiant de la compagnie aérienne
OP_CARRIER_FL_NUM:integer	→ Numéro d'avion de la compagnie
ORIGIN_AIRPORT_ID:integer	→ Aéroport de départ du vol
DEST_AIRPORT_ID:integer	→ Aéroport d'arrivée du vol
CRS_DEP_TIME:integer	→ Heure de départ initiale du vol au format 'HH:mm'
ARR_DELAY_NEW:double	→ Retard total sur le vol à l'arrivée
CANCELLED:double	→ vol annulé = 1
DIVERTED:double	→ vol détourné = 1
CRS_ELAPSED_TIME:double	→ durée normale du vol
WEATHER_DELAY:double	→ retard en minutes sur le vol lié aux météos extrêmes
NAS_DELAY:double	→ retard en minutes sur le vol lié au National Aviation System
_c12:string	→ Dummy column

- Weather

Afin de comprendre le contenu de chaque colonne nous nous sommes appuyés sur le document « Local Climatological Data (LCD) » trouvé sur le site « Quality Controlled Local Climato logical Data (QCLCD) (annexe 2).

Nombre total de lignes = 16 446 741

Toutes les colonnes suffixées par 'Flag' sont dummy ; toujours vides. Nous ne les présentons donc pas.

Voici les différentes colonnes :

WBAN:integer	→ Identifiant de la station météo
Date:integer	→ Date du relevé météo au format 'YYYY-MM-DD'
Time:integer	→ Heure du relevé au format 'HHmm'
StationType:integer	→ dummy column for us
SkyCondition:string (Annexe 2 pour plus de détails)	→ couches de nuages (3 max) avec leur densité et leur hauteur.
Visibility:string	→ visibilité horizontale entre 0 et 10. 10 étant la meilleure
WeatherType:string plus de détails)	→ type de météo : neige, pluie, brouillard, ... (Annexe 2 pour
DryBulbFarenheit:integer	→ température 'sèche' en degrés Fahrenheit
DryBulbCelsius:float selon l'annexe 2)	→ température 'sèche' en degrés Celsius (mesure de référence
WetBulbFarenheit:integer	→ température 'humide' (ressentie ?) en degrés Fahrenheit
WetBulbCelsius:float	→ température 'humide' (ressentie ?) en degrés Celsius
DewPointFarenheit:integer	→ température 'autre' en degrés Fahrenheit
DewPointCelsius:float	→ température 'autre' en degrés Celsius
RelativeHumidity:string	→ taux d'humidité de l'air entre 0 et 100%

<code>WindSpeed</code> :integer	→ Vitesse du vent (en nœuds ?)
<code>WindDirection</code> :integer	→ Direction du vent (180=vent de sud, 360=vent de Nord)
<code>ValueForWindCharacter</code> :integer	→ donnée relative au vent
<code>StationPressure</code> :float	→ pression atmosphérique mesurée à la station
<code>PressureTendency</code> :integer	→ tendance de la pression atmosphérique
<code>PressureChange</code> :integer	→ changement de pression atmosphérique
<code>SeaLevelPressure</code> :float	→ pression atmosphérique au niveau de la mer
<code>RecordType</code> :string	→ type de relevé : principalement AA et SP
<code>HourlyPrecip</code> :float	→ precipitations dans l'heure
<code>Altimeter</code> :float	→ Pression atmosphérique réduite au niveau de la mer (Annexe2)

- **Wban**

Fichier=wban_airport_timezone.csv

Nombre total de lignes = 305

Ce fichier permet de faire la jointure entre les « `AirportID` » des fichiers `Flight` et les « `WBAN` » des fichiers « `weather` ».

Il contient également une colonne « `TimeZone` » donnant le décalage horaire par rapport au méridien Greenwich.

Ces trois colonnes seront des entiers.

```
AirportID:integer
WBAN:integer
TimeZone:integer
```

2.2. Prétraitements des données de vols

Certaines données de la table `Flights` s'avèrent inutiles pour ce projet ou inconsistantes, nous les avons donc traitées comme suit :

- Drop de la colonne `_c12` : dummy, Always empty.
- Colonne `CANCELLED` et `DIVERTED` : suppression des lignes =1 et suppression des colonnes : Dans le papier fourni il est écrit en 4.1 que ces vols ne sont pas conservés.
- On supprime les quelques vols en doublons (mêmes `"FL_DATE"`, `"OP_CARRIER_AIRLINE_ID"`, `"OP_CARRIER_FL_NUM"`, `"ORIGIN_AIRPORT_ID"`).
- Colonnes `NAS_DELAY` et `WEATHER_DELAY` : on remplace les null par 0.
- Colonne `ARR_DELAY_NEW` : on supprime les outliers ; vol plus 45' en avance et plus 5.3 heures en retard.

2.3. Prétraitements des données météo

Certaines données de la table Flights s'avèrent inutiles pour ce projet ou inconsistantes, nous les avons donc traitées comme suit :

- Colonnes suffixées « Flag » : droppées, elles sont vides
- Colonnes Visibility, WindSpeed et RelativeHumidity : suppression des espaces(trim) puis cast en entier (sinon valeurs mises à null).

2.4. Prétraitements des données wban

Les données de wban sont déjà propres, aucun prétraitements à opérer dessus.

3. Analyses exploratoires

Les statistiques ci-dessous sont données pour l'ensemble des années 2012+2013.

3.1. Statistiques et analyse des données de vols

- Complétude des données de vols

Feature	Count	% null	Traitement
OP_CARRIER_AIRLINE_ID	12 466 244	0.00	
OP_CARRIER_FL_NUM	12 466 244	0.00	
ORIGIN_AIRPORT_ID	12 466 244	0.00	
DEST_AIRPORT_ID	12 466 244	0.00	
CRS_DEP_TIME	12 466 244	0.00	
CRS_ELAPSED_TIME	12 466 238	0.00	
ARR_DELAY_NEW	12 264 691	1.60	On ne sait pas si cela signifie, pas de retard ou bien si la donnée est manquante → on va effacer ces lignes (seulement 1.6% des vols)
WEATHER_DELAY	2 284 435	81.68	Il semble que null signifie pas de retard → on met la valeur 0
NAS_DELAY	2 284 435	81.68	

- Distribution des données

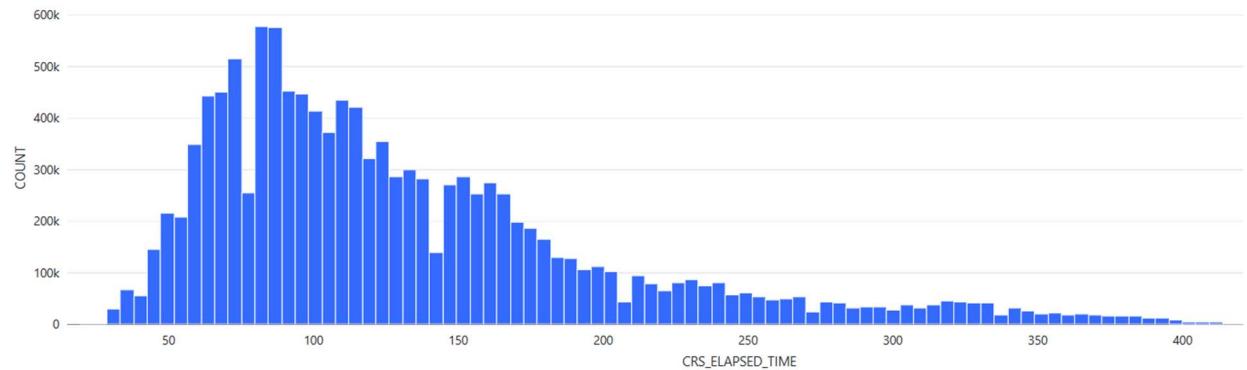
- CRS_ELAPSED_TIME

```
1 display(df_flights_ini.select("CRS_ELAPSED_TIME").where("CRS_ELAPSED_TIME < 480 and CRS_ELAPSED_TIME > 0"))
```

▶ (2) Spark Jobs

Table Visualization 1 +

▶ (4) Spark Jobs

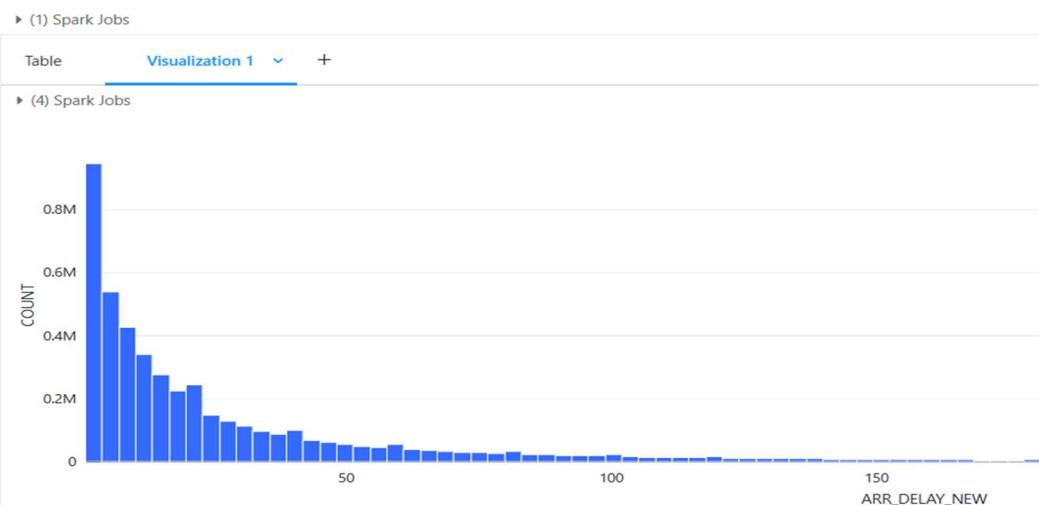


Durée des vols entre 30 et plus 400 minutes. Ceci paraît cohérent car un vol direct Honolulu - New York met près de 10h.

La plus grande partie des vols prend entre 60 et 180 minutes ce qui paraît assez cohérent sur le territoire des Etats-Unis.

- **ARR_DELAY_NEW** : 16.2% des vols sont en retard en 2012 (Seuil à 15 minutes). Répartis comme suit.

```
1 display(df_flights_ini.select("ARR_DELAY_NEW").where("ARR_DELAY_NEW < 480 and ARR_DELAY_NEW > 0"))
```



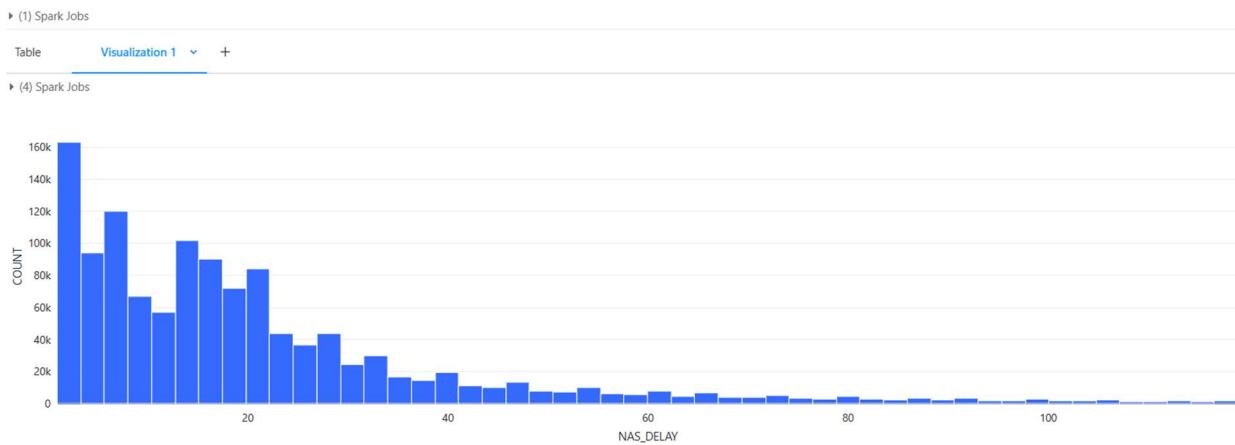
Ceci correspond bien à ce qui est donné dans le papier :

Table III. Analysis of flight on-time performance by year.

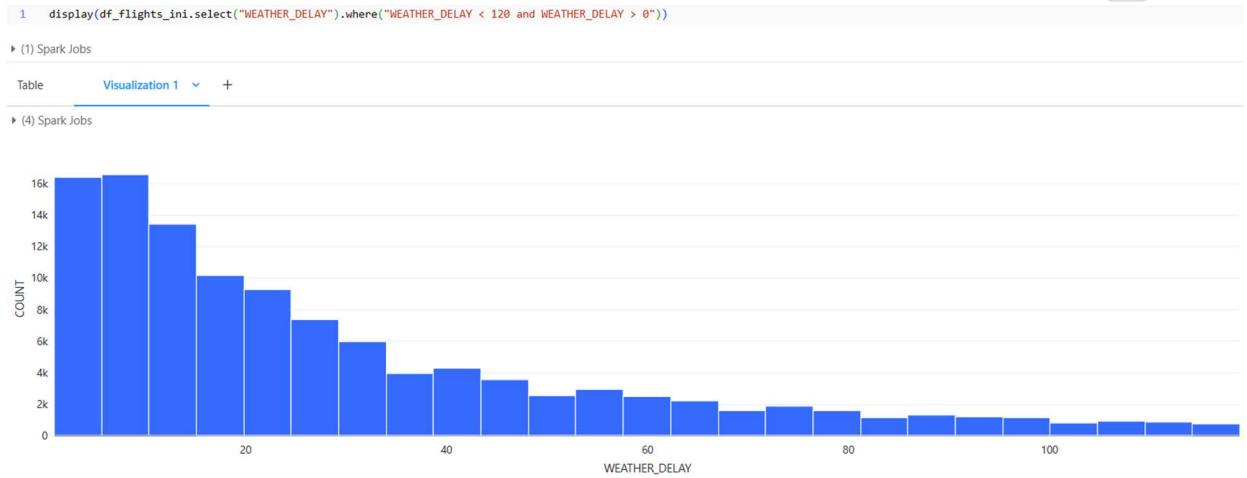
Year	Flights	Ontime	Delayed	Cancelled	Diverted
2009	6,450,285	79.5%	18.9%	1.4%	0.2%
2010	6,450,117	79.8%	18.2%	1.8%	0.2%
2011	6,085,281	79.6%	18.2%	1.9%	0.2%
2012	6,096,762	81.9%	16.7%	1.3%	0.2%
2013	6,369,482	78.3%	19.9%	1.5%	0.2%

- **NAS_DELAY** : 28.6% des vols en retard en 2012 sont dus à la NAS ; non exclusivement. Répartis comme suit :

```
1 display(df_flights_ini.select("NAS_DELAY").where("NAS_DELAY < 120 and NAS_DELAY > 0"))
```



- **WEATHER_DELAY** : 3.4% des vols en retard en 2012 sont dus à des conditions extrêmes ; non exclusivement. Répartis comme suit :



Ces deux chiffres correspondent bien à ce qui est communiqué dans le papier :

Table IV. Analysis of flight delay causes by year.

Year	Air carrier	Late-arriving aircraft	NAS	Extreme weather	Security
2009	26.6%	32.8%	37.0%	3.4%	0.2%
2010	28.9%	35.8%	32.1%	3.1%	0.3%
2011	28.2%	37.0%	31.8%	2.8%	0.2%
2012	29.8%	37.6%	29.6% 2.8%	2.8%	0.2%
2013	27.8%	38.8%	30.3%	2.9%	0.2%

- **ARR_DELAY_NEW Only** : 44.3% des vols en retard en 2012 ne sont ni dus à la NAS (0) ni à des conditions extrêmes (0) (ils sont dus aux Air Carrier et Late-arriving aircraft décrits dans le papier mais non donnés ici).

En effet, dans le papier il est écrit que les retards liés à la météo ne représentent que 32.8% des retards.

➔ Ceci va constituer un problème pour l'apprentissage des modèles car si l'on se base sur ARR_DELAY_NEW beaucoup de retards ne sont pas liés à la météo il va donc être compliqué pour le modèle d'apprendre correctement.

Et si l'on se base sur WEATHER_DELAY, là le modèle devrait pouvoir apprendre correctement, car seule la météo est responsable de ce type de retard, par contre nous ne couvriront qu'une infime partie des retards liés à la météo (3.4% vs 32.8%).

Nous verrons dans un prochain chapitre comment mitiger ce dilemme.

3.2. Statistiques et analyse des données de météo

Voici un tableau de synthèse permettant une première sélection des features.

Les statistiques sont basées sur l'ensemble des données de 2012.

Feature	Null values %	Attribute type	Null values management	Example
SkyCondition	0	Complexe catégorielle >100		BKN013 BKN023 OVC030
Visibility	43%	15 numérique values 0 - 10	10 (médiane)	7.00
WeatherType	92%	Complexe catégorielle > 100		VCTS +RA BR
DryBulbFarenheit	1.8%			Idem DryBulbCelsius
DryBulbCelsius	0.5%	Numérique -53 →60	17(médiane)	-4
WetBulbFarenheit	45%			Idem DryBulbCelsius
WetBulbCelsius	45%			Idem DryBulbCelsius
DewPointFarenheit	44%			Idem DryBulbCelsius
DewPointCelsius	43%			Idem DryBulbCelsius
RelativeHumidity	45%	Numérique 0-100	72% (médiane)	93
WindSpeed	40%	Numérique 0-172	6 (médiane)	5
WindDirection	43%	Catégorielle 38 values		160
ValueForWindCharacter	92%	Numérique		18
StationPressure	45%	Numérique		29.91
PressureTendency	93%	catégorielle 10 values 0-8		8
PressureChange	93%	Numérique		007
SeaLevelPressure	80%	Numérique		30.11
RecordType	0%	Catégoriale 6 values	0	AA(0)
HourlyPrecip	98%	Numérique		0.02
Altimeter	43%	Numérique 28.17-32	30	29.99

Les features apparaissant en noir ne seront pas conservées soit :

- pour des taux de complétude trop faibles,
- car une autre feature est équivalente et meilleure (**DryBulbCelsius** sera la seule température conservée)
- idem pour les mesures de pression atmosphérique : **Altimeter** sera conservée car :
 - * StationPressure dépend de l'altitude de la station
 - * meilleure complétude que les autres features de ce type.

La feature **WindDirection** ne sera pas conservée malgré une complétude moyenne car elle est catégorielle avec 36 valeurs (lourd à traiter) et que son impact sur les retards n'est pas évident du tout.

Les features : **Visibility**, **RelativeHumidity**, **WindSpeed** et **Altimeter** sont conservées malgré un taux de complétude moyen car il s'agit de variables continues et qu'elles ont à priori un impact sur les retards, comme nous le verrons dans le chapitre suivant.

Nous avons également calculé les médianes de ces features dans le but de remplacer les valeurs nulles. En effet le RandomForest (que nous utiliserons comme dans le papier) ne tolère pas les valeurs nulles et il filtrerait la ligne entière pour une seule feature nulle.

3.3. Analyse combinée

Nombre de vols dans les fichiers bruts : **12 250 157**

Nombre de vols après jointure wban : **11 592 854**

Nombre de jours couverts par les vols : **731**

Nombre de relevés météo bruts : **32 631 312**

Nombre de relevés météo avec 1 par heure : **10 668 051**

Nombre de jours couverts par la météo : **243**

Nombre de vols après jointure weather final : **2 803 543**

→ Etant donnée que la météo est manquante pour de nombreux jours il est logique que le nombre de vols après jointure chute à 2.8M.

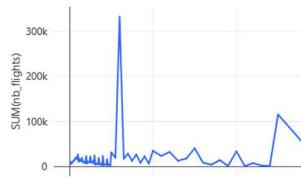
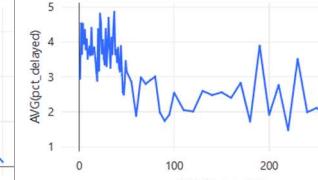
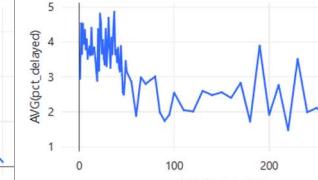
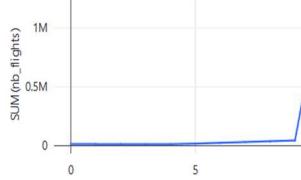
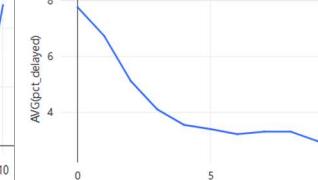
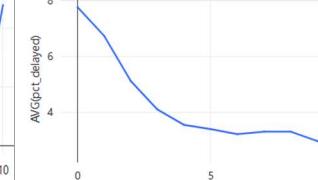
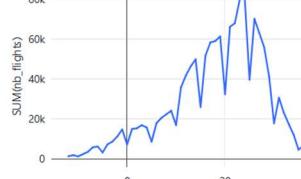
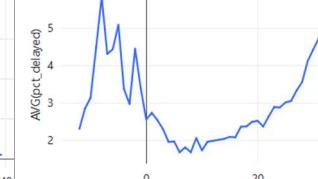
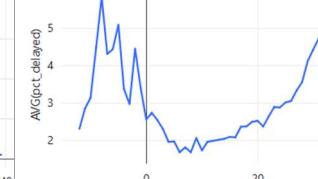
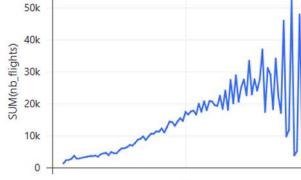
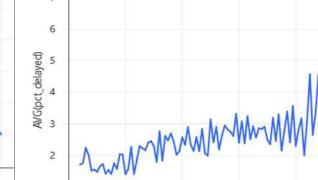
Ci-dessous un deuxième tableau de statistiques sur les features conservées en croisant les données de **Weather** et les données de **Flights**, notamment le retard.

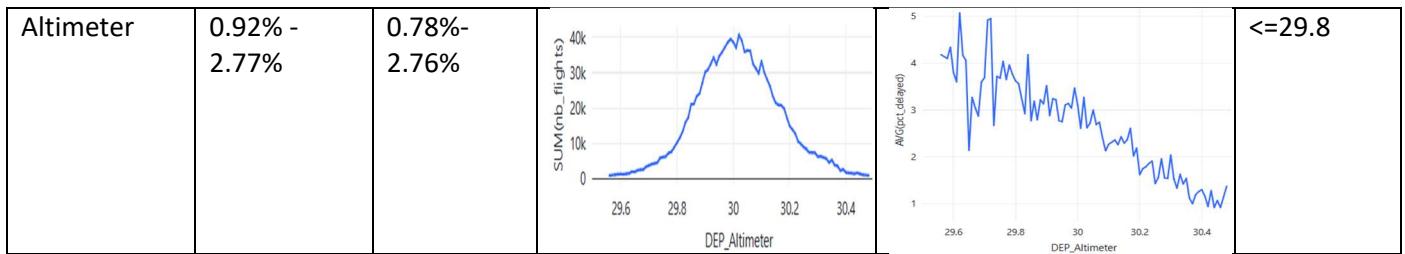
En voici la description :

- **Min-max P(DelayD2) (2012)** : Pourcentage min et max de retard selon la valeur de la feature, sur l'année 2012 entière
- **Min-max P(DelayD2) (January 2012)** : idem colonne précédente mais uniquement sur janvier 2012 (dans le but de mettre en lumière une éventuelle saisonnalité).
- **Distribution des données** : nombre d'occurrences pour chaque valeur de la feature concernée.
- **Taux de retard / valeur** : taux de retard pour pour chaque valeur de la feature concernée.

- **Bad weather conditions** : valeur, ou plage de valeurs, sur laquelle la feature obtient les plus hauts taux de retard. Cette feature sera testée pour essayer d'éliminer les retards non liés à la météo du dataset. D'avantage de détails seront données dans le chapitre feature engineering.

Si l'on prend l'exemple de **DryBulbCelsius**, on peut clairement voir que la grande majorité des vols avec une température comprise entre 10 et 30°C, et que nous avons deux pics de retards : entre -2 et -9°C et au-dessus des 30°C

Feature	Min-max P(DelayD2) (2012)	Min-max P(DelayD2) (January 2012)	Distribution des données	Taux de retard / valeur	Bad weather conditions
Sky1_type	1.77%- 8.51%	clearSky = 1.5% fewClouds = 2.15% scatterClou ds = 2.89% brokenClou ds = 3.2% overCast = 4.96% obscuredSk y = 11.76%	clearSky = 296 482 fewClouds = 591 847 scatterClouds = 228 825 brokenClouds = 177 998 overCast = 98 885 obscuredSky = 4 831	clearSky = 1.77% fewClouds = 2.65% scatterClouds = 3.41% brokenClouds = 3.41% overCast = 3.70% obscuredSky = 8.51%	obscuredSky
Sky1_height	1.14%- 4.87%	1.19%- 7.61%	 		<=25
Visibility	2.6%-7.76%	2.08%- 10.57%	 		<=4
DryBulb Celsius	1.68%- 5.83%	1.24%- 5.83%	 		-2 → -9 et >30
Relative Humidity	1.4% - 6.73%	0.97%- 6.94%	 		>=90
WindSpeed	2.29% - 5.42%	2.04%- 4.84%	 		>=25
RecordType	2.62% - 4.57%	2.38%- 5.45%	AA = 1 281 303 flights SP = 118 530 flights	AA = 2.62% delayed SP = 4.57% delayed	SP(1)



4. Feature engineering

4.1. Préparation des données de vols pour les modèles

- **FL_DATETIME** : concaténation FL_DATE et CRS_DEP_TIME pour obtenir la date de **départ** du vol au format « yyyy-MM-dd HHmm ».

Ex : 2013-07-01 et 1040 → 2013-07-01 10:40:00

- **LOCAL_DEP_DT_RND** : FL_DATETIME arrondi à l'heure pleine la plus proche.

Ex : 2013-07-01 10:40:00 → 2013-07-01 11:00:00 et 2013-07-01 10:10:00 → 2013-07-01 10:00:00

Cette feature va nous permettre de faire la jointure avec les données de weather aux quelles on va appliquer le même arrondi.

Cette méthode est simple et efficace. Son défaut c'est qu'elle produit plusieurs lignes avec le même **LOCAL_DEP_DT_RND** si l'on a plusieurs relevés météo durant la même heure.

- **LOCAL_ARR_DT** : calcul de l'heure locale d'arrivée du vol (car pas donnée)

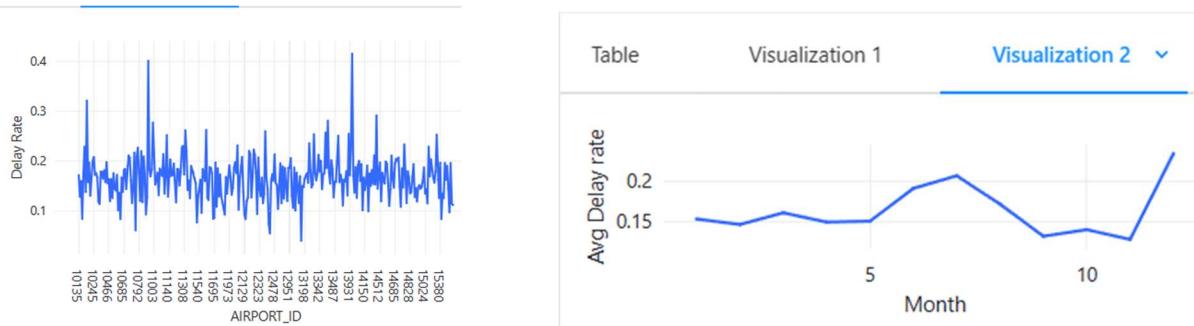
Date/heure de départ + durée du vol (CRS_ELAPSED_TIME) + retard (ARR_DELAY_NEW) + décalage horaire (TimeZone_ARR - TimeZone_Dep).

Ex : Date de départ = 2013-07-01 13:20:00, durée du vol = 140', TimeZone_ARR = -6, TimeZone_Dep = -7

$$\rightarrow \text{LOCAL_ARR_DT} = 13:20:00 + 02:20:00 + 01:00:00 = 2013-07-01 16:40:00$$

- **LOCAL_ARR_DT_RND** : idem **LOCAL_DEP_DT_RND** mais pour l'heure d'arrivée du vol.

- **TX_AIRPORT_MONTH_D** et **TX_AIRPORT_MONTH_A** : On a calculé le taux de retard par aéroport et par mois. On voit clairement que cela a une grosse incidence sur le taux de retard :



Cela nous écarte un peu du sujet, car il ne s'agit plus vraiment de retards liés à la météo, mais si l'on arrive également à prévoir les retards autres, pourquoi se priver ?

- **IS_DELAYED**: c'est la target. Le vol est-il considéré en retard ou non ?

Il y a plusieurs façons de la calculer. Voici les différentes méthodes imaginées.

- Méthode ARR_DELAY_NEW : On considère comme en retard tous les vols avec $\text{ARR_DELAY_NEW} \geq \text{delayThreshold}$.

Cette méthode à l'avantage de considérer tous les vols retardés à cause de la météo (puisque elle considère tous les vols retardés).

Mais elle a le gros inconvénient de considérer tous les vols retardés pour une autre raison que la météo. Ce qui représente un très gros bruit qui va nuire à l'apprentissage du modèle.

Elle représente 100% de vols en retard.

- Méthode WEATHER_DELAY : On considère comme en retard tous les vols avec $\text{WEATHER_DELAY} \geq \text{delayThreshold}$.

Cette méthode à l'avantage de considérer uniquement des vols retardés à cause de la météo.

Mais elle a le gros inconvénient de considérer uniquement une petite partie des retards liés à la météo.

- Méthode D2_DELAY : c'est la méthode utilisée dans le papier. En voici la formule :

$\text{ARR_DELAY_NEW} \geq \text{delayThreshold} \text{ and } (\text{WEATHER_DELAY} > 0 \text{ or } \text{NAS_DELAY} \geq \text{delayThreshold})$

Autrement dit : Tous les vols en retard affectés par une météo extrême plus ceux avec un retard NAS supérieur au seuil.

Cette méthode semble un assez bon compromis car elle va capturer les retards liés au WEATHER_DELAY mais également ceux liés à une mauvaise météo (mais non extrême) à travers le NAS_DELAY. Ceci sans trop prendre en compte les vols retardés pour d'autres causes que la météo ; En effet il est estimé que 58.3% des NAS_DELAY sont causés par la météo (cf papier).

- Méthode BAD_WEATHER : Comme détaillée dans le prochain chapitre, la feature CalmWeather a été créée pour marquer une ligne de weather lorsque les conditions météos y sont en tous points calmes.

A l'aide de cette feature, nous allons pouvoir déterminer les vols pour lesquels la météo a été très favorable et donc en déduire que si retard il y a sur ce vol il n'est pas lié à la météo. Et ainsi mettre le IS_DELAYED = 0.

Il en résulte que toutes les lignes avec IS_DELAYED = 1 ont au moins un élément de météo mauvais.

$\text{ARR_DELAY_NEW} \geq \text{delayThreshold} \text{ and } \text{CalmWeather} = 0$.

- MéthodeD2_BAD_DELAY : Idem à la méthode BAD_WEATHER mais appliquée sur la target D2_DELAY cette fois.

`ARR_DELAY_NEW >= delayThreshold and (WEATHER_DELAY > 0 or NAS_DELAY >= delayThreshold) and CalmWeather = 0.`

Voici ci-dessous la proportion de vols tagués IS_DELAYED par rapport aux nombre total de vols retardés pour chaque méthode.

Méthode	% de retards par rapport au total de retards
ARR_DELAY_NEW	100%
BAD_WEATHER	78%
D2_DELAY	40%
D2_BAD_DELAY	32%
WEATHER_DELAY	4.5%

Si l'on compare aux proportions données dans le papier (cf tableau V ci-dessous), on peut constater que D2_DELAY et encore plus D2_BAD_DELAY devrait être les deux méthodes se rapprochant le plus de la cible.

Table V. Analysis of delayed flights due to weather conditions by year.

Year	Extreme weather	NAS related to weather	Late-arriving aircraft related to weather	Total weather
2009	3.4%	24.3%	14.5%	42.3%
2010	3.1%	20.4%	14.0%	37.4%
2011	2.8%	20.1%	14.3%	37.2%
2012	2.8%	17.4%	12.6%	32.8%
2013	2.9%	17.7%	14.1%	34.6%

4.2. Préparation des données de météo pour les modèles

- **Sky1** : c'est une feature temporaire qui permet d'extraire la première couche de nuage de SkyCondition. Ex : BKN013 BKN023 OVC030 → BKN013. Etant donné que c'est surtout les nuages bas qui posent problème (cf feature Sky1_height) les 2 autres couches de nuages ne seront pas traitées
- **Sky1_type** : c'est les 3 premiers digits de Sky1. Ex : BKN013 → BKN (ils sont toujours renseignés).

Les différentes valeurs possibles sont : CLR, FEW, SCT, BKN, OVC, VV0 et VV1.

Pour chacune de ses valeurs on créera une feature booléenne : clearSky, fewClouds, scatterClouds, brokenClouds, overCast et obscuredSky.

- **Sky1_height** : c'est les 3 derniers digits de Sky1. Ex : BKN013 → 013. Lorsqu'ils n'existent pas on met la valeur médiane par défaut 60.
- **CalmWeather** : Comme évoqué plus haut nous avons crée une feature pour taguer les lignes météo avec des conditions calmes sur toutes les features. Concrètement Si :
 - `Sky1_height > 25`

- Visibility > 4
- DryBulbCelsius < -9 et > -2
- RelativeHumidity < 90
- WindSpeed < 25
- RecordType <> 1
- obscuredSky <> 1

Et ceci pour les différentes heures prises en compte, précédent chaque vol.

Alors le CalmWeather = 1 pour ce vol.

Les différentes valeurs ont été choisies en se basant sur les stats de retard pour chacune des features (cf tableau chapitre 3.3).

4.3. Jointure et sélection de features

• Weather auto-jointure

La première jointure à réaliser est celle permettant de mettre sur une même ligne un relevé météo et les n relevés précédents. Nous avons fixé n=3 pour ne pas avoir trop de features non plus et parce que dans le papier il est écrit que d'aller plus loin n'apporte pas grand-chose.

Pour réaliser cette jointure nous avons utilisé la fonction lag qui rajoute des colonnes en prenant la ligne du dessus. Cette fonction est assez efficace en Spark pour traiter ce grand nombre de données.

Ci-dessous un aperçu de ce que ça donne avec les relevés H0 et H-2 en fond vert et H-1 et H-3 en blanc.

On constate bien que les données H0 de 14:00 se retrouvent en H-1 de 15h, puis en H-2 de 16:00 et en H-3 de 17:00. (données en rouge).

On peut constater que cela fonctionne également quand on change de jour : entre le 02/05/2012 23:00 et le 03/05/2012 00:00

WBAN	DATETIME_RND	Sky1Visi	Dry1Rela	Wir1Reci	Altimet1	clea1few1	few1scal1	bro1o1	ove1obs1	Sky1Visi	Dry1Rela	Wir1Reci	Altimet1	clea1few1	few1scal1	bro1o1	ove1obs1	Sky1Visi	Dry1Rela	Wir1Reci	Altimet1	clea1few1	few1scal1	bro1o1	ove1obs1	Calm																			
4864	02/05/2012 00:00	60	10	18	56	17	0	29.85	1	0	0	0	0	0	95	10	17	58	20	0	29.89	0	0	1	0	0	95	10	16	63	0	29.9	0	0	0	0	1	0	1						
4864	02/05/2012 01:00	60	10	18	56	17	0	29.85	1	0	0	0	0	0	60	10	18	56	17	0	29.85	1	0	0	0	0	0	95	10	17	58	20	0	29.89	0	0	0	1	0	1					
4864	02/05/2012 02:00	60	10	18	52	20	0	29.84	1	0	0	0	0	0	60	10	18	56	17	0	29.85	1	0	0	0	0	0	95	10	17	58	20	0	29.89	0	0	1	0	0	1					
4864	02/05/2012 03:00	60	10	18	52	18	0	29.83	1	0	0	0	0	0	60	10	18	52	20	0	29.84	1	0	0	0	0	0	60	10	18	56	17	0	29.85	1	0	0	0	0	1					
4864	02/05/2012 04:00	60	10	17	52	23	0	29.81	1	0	0	0	0	0	60	10	18	52	18	0	29.83	1	0	0	0	0	0	60	10	18	56	20	0	29.85	1	0	0	0	0	1					
4864	02/05/2012 05:00	60	10	16	56	20	0	29.81	1	0	0	0	0	0	60	10	17	52	23	0	29.81	1	0	0	0	0	0	60	10	18	52	20	0	29.84	1	0	0	0	0	1					
4864	02/05/2012 06:00	60	10	15	62	13	0	29.79	1	0	0	0	0	0	60	10	16	56	20	0	29.81	1	0	0	0	0	0	60	10	18	52	20	0	29.83	1	0	0	0	0	1					
4864	02/05/2012 07:00	60	10	13	72	14	0	29.8	1	0	0	0	0	0	60	10	15	62	13	0	29.79	1	0	0	0	0	0	60	10	17	53	20	0	29.81	1	0	0	0	0	1					
4864	02/05/2012 08:00	60	10	13	69	11	0	29.8	1	0	0	0	0	0	60	10	13	72	14	0	29.8	1	0	0	0	0	0	60	10	16	20	56	0	29.81	1	0	0	0	0	1					
4864	02/05/2012 09:00	60	10	12	72	18	0	29.8	1	0	0	0	0	0	60	10	13	69	11	0	29.8	1	0	0	0	0	0	60	10	15	13	62	0	29.79	1	0	0	0	0	1					
4864	02/05/2012 10:00	60	10	12	72	18	0	29.8	1	0	0	0	0	0	60	10	12	72	13	0	29.8	1	0	0	0	0	0	60	10	13	14	72	0	29.8	1	0	0	0	0	1					
4864	02/05/2012 11:00	60	10	11	77	7	0	29.82	1	0	0	0	0	0	60	10	12	72	13	0	29.8	1	0	0	0	0	0	60	10	13	11	69	0	29.8	1	0	0	0	0	1					
4864	02/05/2012 12:00	60	10	11	77	10	0	29.81	1	0	0	0	0	0	60	10	11	77	10	0	29.81	1	0	0	0	0	0	60	10	12	72	13	0	29.8	1	0	0	0	0	1					
4864	02/05/2012 13:00	15	7	11	80	7	0	29.81	0	0	0	0	1	0	60	10	11	77	10	0	29.81	1	0	0	0	0	0	60	10	12	72	13	0	29.8	1	0	0	0	0	0					
4864	02/05/2012 14:00	13	7	11	80	7	0	29.83	0	0	0	1	0	0	15	7	11	80	11	0	29.81	0	0	1	0	0	0	60	10	11	77	10	0	29.81	1	0	0	0	0	0					
4864	02/05/2012 15:00	13	5	10	93	11	0	29.83	0	0	0	0	1	0	0	13	7	11	80	7	0	29.83	0	0	0	1	0	0	60	10	11	77	10	0	29.81	1	0	0	0	0	0				
4864	02/05/2012 16:00	3	4	13	100	7	0	29.75	0	0	0	0	1	0	0	13	5	10	93	11	0	29.83	0	0	0	1	0	0	13	5	10	11	80	0	29.81	0	0	1	0	0	0				
4864	02/05/2012 17:00	7	10	17	81	13	0	29.76	0	0	0	1	0	0	0	3	4	13	100	7	0	29.75	0	0	0	0	1	0	0	13	5	10	11	93	0	29.83	0	0	0	1	0	0			
4864	02/05/2012 18:00	18	10	17	75	14	0	29.77	0	0	0	0	0	1	0	7	10	17	81	13	0	29.76	0	0	1	0	0	0	3	4	13	7	100	0	29.75	0	0	0	0	1	0				
4864	02/05/2012 19:00	20	10	18	73	7	0	29.78	0	0	0	0	0	1	0	18	10	17	75	14	0	29.77	0	0	1	0	0	0	0	1	0	0	7	10	17	13	81	0	29.76	0	0	1	0	0	0
4864	02/05/2012 20:00	24	10	20	63	11	0	29.79	0	0	1	0	0	0	20	10	18	73	7	0	29.78	0	0	0	1	0	0	0	7	10	17	13	81	0	29.76	0	0	1	0	0	0				
4864	02/05/2012 21:00	26	10	20	63	7	0	29.79	0	0	1	0	0	0	24	10	20	63	11	0	29.79	0	0	1	0	0	0	0	20	10	18	73	7	0	29.78	0	0	0	0	1	0				
4864	02/05/2012 22:00	60	10	21	64	7	0	29.8	1	0	0	0	0	0	26	10	20	63	7	0	29.79	0	0	1	0	0	0	24	10	20	63	11	0	29.79	0	0	1	0	0	0					
4864	02/05/2012 23:00	60	10	22	55	11	0	29.81	1	0	0	0	0	0	60	10	21	64	7	0	29.8	1	0	0	0	0	0	24	10	20	63	11	0	29.79	0	0	1	0	0	0					
4864	03/05/2012 00:00	60	10	23	53	14	0	29.82	1	0	0	0	0	0	60	10	23	53	7	0	29.81	1	0	0	0	0	0	60	10	22	11	55	0	29.81	1	0	0	0	0	1					
4864	03/05/2012 01:00	60	10	23	53	7	0	29.81	1	0	0	0	0	0	60	10	23	53	7	0	29.81	1	0	0	0	0	0	60	10	22	11	55	0	29.81	1	0	0	0	0	1					
4864	03/05/2012 02:00	60	10	23	53	7	0	29.82	1	0	0	0	0	0	60	10	23	53	14	0	29.82	1	0	0	0	0	0	60	10	22	11	55	0	29.81	1	0	0	0	0	1					
4864	03/05/2012 03:00	50	10	23	52	7	0	29.83	0	0	1	0	0	0	60	10	23	53	7	0	29.81	1	0	0	0	0	0	60	10	23	53	7	0	29.82	1	0	0	0	0	1					
4864	03/05/2012 04:00	60	10	23	48	7	0	29.83	1	0	0	0	0	0	50	10	23	52	7	0	29.83	0	0	1	0	0	0	60	10	23	53	7	0	29.82	1	0	0	0	0	1					
4864	03/05/2012 05:00	60	10	23	52	7	0	29.84	1	0	0	0	0	0	60	10	23	48	7	0	29.83	1	0	0	0	0	0	60	10	23	7	53	0	29.82	1	0	0	0	0	1					

- **Jointure Flight-wban-Weather**

- Flight -wban :

Afin de permettre la jointure entre Flight et Weather, il faut d'abord faire la jointure entre Flight et wban pour récupérer le wban associé au AirportID, ceci pour l'aéroport de départ et celui d'arrivée :

```
wban("AirportID") = flights("ARR_AIRPORT_ID"), "inner"
```

```
wban("AirportID") = flights("DEP_AIRPORT_ID"), "inner"
```

Une jointure « inner » est choisie car il ne sert à rien de conserver des vols pour lesquels on ne peut pas déterminer le wban.

- Flight – Weather :

Nous pouvons maintenant faire les jointures entre flight et weather (avec les n relevés en ligne) sur le wban et sur la date/heure arrondie, ceci pour l'aéroport de départ et celui d'arrivée :

```
weather.WBAN = flights_wban.WBAN_DEP and weather.DATETIME_RND = flights_wban.LOCAL_DEP_DT_RND, "inner"
```

```
weather.WBAN = flights_wban.WBAN_ARR and weather.DATETIME_RND = flights_wban.LOCAL_ARR_DT_RND, "inner"
```

Ici aussi une jointure « inner » est choisie car nous ne souhaitons pas conserver les vols pour lesquels il manque les données météo au départ ou à l'arrivée.

Nous avons donc maintenant un dataset, de 2.8M lignes, avec un vol par ligne accompagné de ses relevés météos au départ et à l'arrivée, pour les 3 heures précédant le vol.

Nous allons donc pouvoir passer à la partie Machine Learning.

5. Modèles de Machine Learning

Dans ce dernier paragraphe nous allons expliquer comment nous avons développé différents modèles prédictifs, comment nous en avons évalué la performance afin de ne retenir que le plus pertinent d'entre eux.

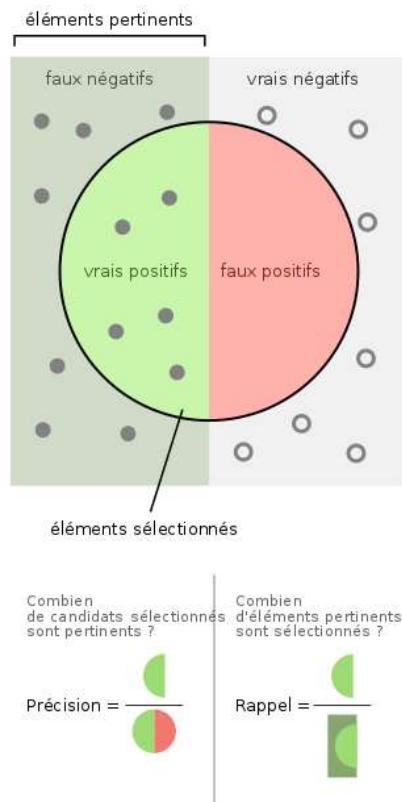
5.1. Metrics & proof of concept

La première étape consiste à développer un “proof of concept” afin de s’assurer de la faisabilité d’un modèle prédictif, de savoir si celui-ci répond correctement aux enjeux métier et si la performance requise peut être atteinte.

Dans notre cas de figure, il convient de déterminer avec le plus de précision possible si un vol sera en retard de plus de 30 minutes. Il est à noter que le seuil de 30 minutes est choisi arbitrairement : nous aurions pu retenir une autre valeur, néanmoins 30 minutes semble un bon compromis compte tenu des différents essais menés dans le papier d’origine.

Avant toute modélisation, il est important de fixer dès le départ les différentes métriques qui permettront d’évaluer la pertinence d’un modèle soit dans l’absolue, soit par rapport à un autre modèle. Pour ce faire nous allons retenir la « precision », le « recall » ainsi que le « f1-score ».

Dans le dataset préparé pour le modèle de machine learning, les retards au-delà du seuil précédemment définis représentent environ 3 % de la totalité des vols : ce jeu de donnée est déséquilibré. L’« accuracy » n’est pas une bonne métrique car prédire que la totalité des vols seraient à l’heure entraînerait une accuracy de 97% !

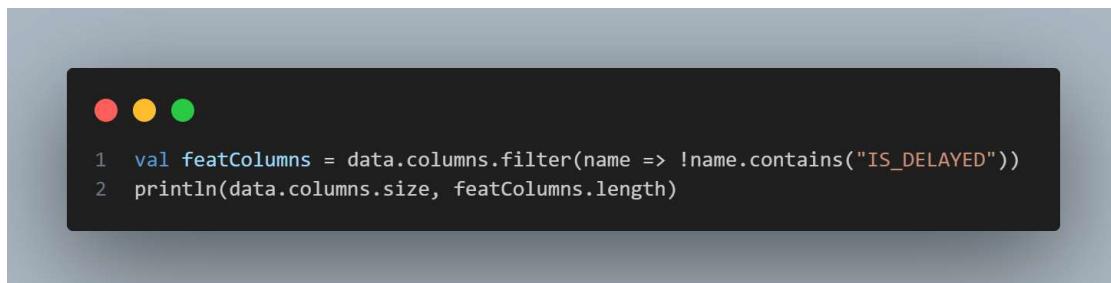


La précision et le recall sont plus adaptés. Le f1-score constitue la moyenne harmonique des deux : pour avoir un bon f1score, il faut que le recall et la précision soient tous les deux bons. Ces 3 métriques - allant de 0 à 1 - permettent de se faire une idée dans l'absolue de la pertinence du modèle.

Le modèle est entraîné sur le « train set » alors que les métriques sont évaluées sur le « test set » de manière à mesurer l'aptitude du modèle à faire de bonnes prédictions sur un dataset qu'il n'a pas « vu », ainsi on s'assure de sa capacité à généraliser.

Différentes étapes :

- Création de la liste des « features » dissocié de la « target » ou du label. Nous avons ici plus d'une centaine de colonnes correspondants aux conditions météo pour H0, H-1, H-2 et H-3 par rapport aux horaires de décollage et d'arrivée du vol.



```
● ● ●  
1 val featColumns = data.columns.filter(name => !name.contains("IS_DELAYED"))  
2 println(data.columns.size, featColumns.length)
```

- Création du vecteur de features à l'aide du transformateur Spark « VectorAssembler », qui combine les colonnes de features en un vecteur sur une seule colonne qui sera utilisé par le modèle :



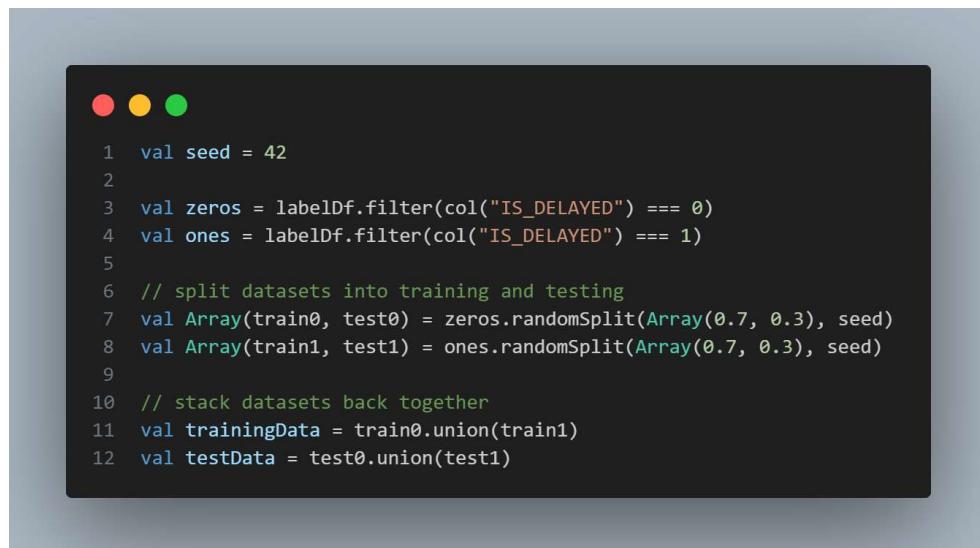
```
● ● ●  
1 val vectorAssembler = new VectorAssembler()  
2   .setInputCols(featColumns)  
3   .setOutputCol("features")  
4   .setHandleInvalid("skip") // options are "keep", "error" or "skip"  
5  
6  
7 val featureDf = vectorAssembler.transform(data)
```

- Puis indexation du label (ici il s'agit de notre colonne « IS_DELAYED » qui est transformée :



```
● ● ●  
1 val labelIndexer = new StringIndexer()  
2   .setInputCol("IS_DELAYED")  
3   .setOutputCol("label")  
4  
5  
6 val labelDf = labelIndexer.fit(featureDf).transform(featureDf)
```

- Split de notre dataset en deux jeux distincts « train » et en « test » avec éventuellement rééquilibrage des labels (cf. ci-après) :



```

1 val seed = 42
2
3 val zeros = labelDf.filter(col("IS_DELAYED") === 0)
4 val ones = labelDf.filter(col("IS_DELAYED") === 1)
5
6 // split datasets into training and testing
7 val Array(train0, test0) = zeros.randomSplit(Array(0.7, 0.3), seed)
8 val Array(train1, test1) = ones.randomSplit(Array(0.7, 0.3), seed)
9
10 // stack datasets back together
11 val trainingData = train0.union(train1)
12 val testData = test0.union(test1)

```

- Finalement instantiation d'un modèle avec ses hyperparamètres, entraînement sur le train et création d'un dataframe de prédictions à partir des features du jeu de test :

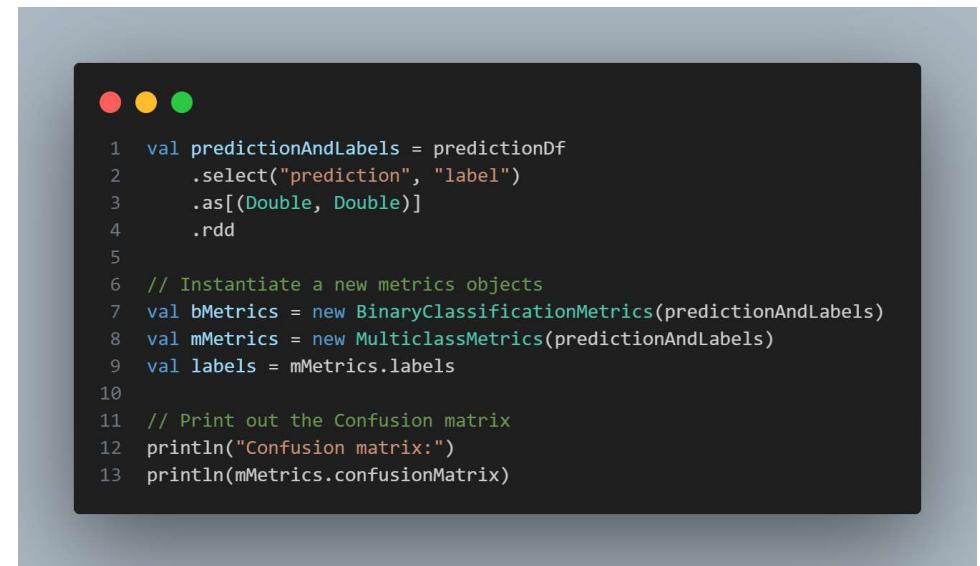


```

1 val decisionTreeClassifier = new DecisionTreeClassifier()
2   .setImpurity("gini")
3   .setMaxDepth(20)
4   .setSeed(seed)
5   .setLabelCol("label")
6   .setFeaturesCol("features")
7
8 val decisionTreeModel = decisionTreeClassifier.fit(trainingData)
9
10 val predictionDf = decisionTreeModel.transform(testData)

```

- Pour terminer calcul des métriques afin d'évaluer le modèle en question



```

1 val predictionAndLabels = predictionDf
2   .select("prediction", "label")
3   .as[(Double, Double)]
4   .rdd
5
6 // Instantiate a new metrics objects
7 val bMetrics = new BinaryClassificationMetrics(predictionAndLabels)
8 val mMetrics = new MulticlassMetrics(predictionAndLabels)
9 val labels = mMetrics.labels
10
11 // Print out the Confusion matrix
12 println("Confusion matrix:")
13 println(mMetrics.confusionMatrix)

```

```

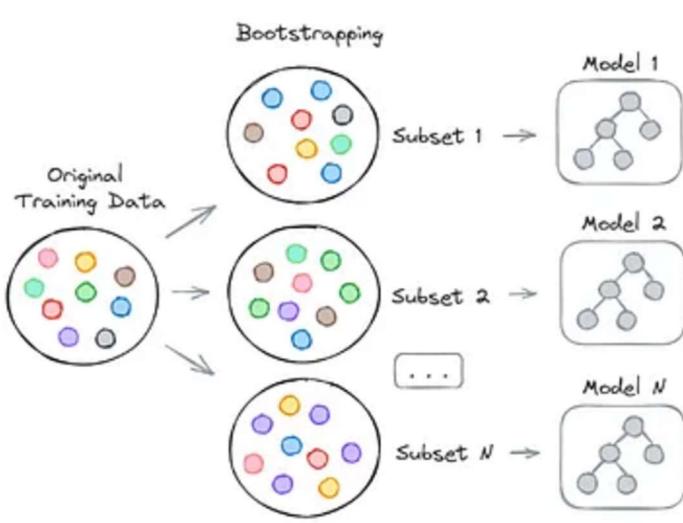
1 val trueNegative = predictionDf.filter(col("prediction") === 0 && col("label") === col("prediction")).count().toDouble
2 val truePositive = predictionDf.filter(col("prediction") === 1 && col("label") === col("prediction")).count().toDouble
3 val falseNegative = predictionDf.filter(col("prediction") === 0 && col("label") != col("prediction")).count().toDouble
4 val falsePositive = predictionDf.filter(col("prediction") === 1 && col("label") != col("prediction")).count().toDouble
5
6 val precision = truePositive / (truePositive + falsePositive)
7 val recall = truePositive / (truePositive + falseNegative)
8 val acc = (truePositive + trueNegative) / (truePositive + trueNegative + falsePositive + falseNegative)
9 val f1score = 2 * precision * recall / (precision + recall)

```

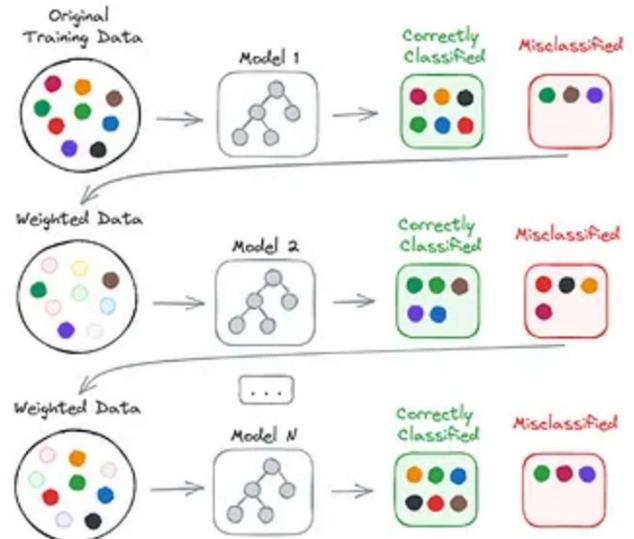
Différents types de modèles utilisés

- **Decision tree** : Le fonctionnement des arbres de décision est basé sur **des règles de logiques très simples**. Les décisions possibles sont situées sur les feuilles de l'arbre aux extrémités des branches. Elles sont modifiées en fonction des décisions prises à chaque nœud. A chaque embranchement le modèle choisit une feature et une valeur qui lui permet d'être le plus discriminant en fonction du label. Les arbres de décision sont non paramétriques et nécessitent très peu de prétraitement de données. Ils sont faciles à interpréter et à entraîner.
- **Random Forest** : est constitué d'un ensemble d'arbres de décision indépendants. Chaque arbre dispose d'une vision parcellaire du problème du fait d'un double tirage aléatoire :
 - o un tirage aléatoire avec remplacement sur les observations (les lignes de votre base de données). Ce processus s'appelle le tree bagging,
 - o un tirage aléatoire sur les variables (les colonnes de votre base de données). Ce processus s'appelle le feature sampling.
 A la fin, tous ces arbres de décisions indépendants sont assemblés. La prédiction faite par le random forest pour des données inconnues est alors la moyenne (ou le vote, dans le cas d'un problème de classification) de tous les arbres.
- **Gradient Boosting** : il s'agit d'un ensemble de « weak learners » i.e des arbres de décision, créés les uns après les autres de manière séquentielle, formant un strong learner. Chaque « weak learner » est entraîné pour corriger les erreurs des « weak learners » précédents. A chaque étape un pas est affinée par une descente de gradient qui pondère les corrections des résidus.

Bagging



Boosting



Differentes stratégies :

- **Baseline** : la base line correspond à un modèle relativement simple et « naïf » qui sert de base pour être comparer à des modèles ultérieurs plus aboutis. Ici nous avons retenu un simple « decision tree » (MaxDepth(30)). Les métriques obtenues sont : **precision = 0.52 / recall = 0.08 / f1score = 0.15**.
- **Stratified split** : le randomsplit utilisé ne permet pas d'avoir la même proportion de vols en retard dans le train et le test sets. Il se peut qu'il n'y en ait très peu dans le test set (bien moins de 3%), ce qui rend l'évaluation peu représentative.

Même si cela est moins vrai en « big data », nous avons quand même constaté un écart pouvant approcher 1%, qui varie en faisant changer le « seed ». Pour cette raison, il est intéressant de procéder en découplant le jeu de donnée par label, de pratiquer deux randomsplit avec les mêmes proportions (dans notre cas 70% en train et 30% en test) puis de fusionner les labels.

Les métriques obtenues sont : **precision = 0.72 / recall = 0.66** avec un random forest (et légèrement inférieur avec un décision tree). A noter que nous avons testé plusieurs hyperparamètres : nombre d'arbre pour le RF et profondeur du ou des arbres.

- **Reéquilibrage du dataset** : en pratiquant de l'undersampling de la classe majoritaire (vols à l'heure), il est possible de rééquilibrer les deux classes. Nous avons réalisés plusieurs tests en allant jusqu'à environ 22% de vols en retards sur le total de lignes.

Les métriques obtenues sont : **precision = 0.79 / recall = 0.25** avec un random forest.

- **Class weights** : nous avons également mené plusieurs essais avec des poids accordés aux classes correspondant au ratio de la classe opposée, afin d'accorder plus d'importance à la classe minoritaire : les retards.

Ceci a permis de limiter le nombre de faux négatifs et de faire remonter le recall. Mais la precision s'est dégradée : Les métriques obtenues sont : **precision = 0.55 / recall = 0.62** avec un random forest. Des résultats sensiblement identiques ont été avec un GradientBoostingTree classifier.

- **Selection de features et historique** : un dernier essai a été mené en selection que certains types de données météo ou en ne conservant moins d'historique. Cela à malheureusement dégradé les résultats : Les métriques obtenues sont : **precision = 0.75 / recall = 0.24** avec un random forest.
- **Adaptation de la target / du dataset**. Nous avons conservé jusque-là l'équivalent du dataset D2 tel que décrit dans le papier d'origine. Néanmoins, en supprimant les cas (i.e les lignes) correspondants à des données météo « calmes » (cf. analyse du chapitre 3), en utilisant un random forest sur un dataset légèrement rééquilibré les métriques obtenues sont alors bien meilleures: **precision = 0.85 / recall = 0.84**.

5.2. Model final, mise en production

Une fois l'étape de faisabilité ou de « proof of concept » validée, le modèle peut être mis en production. Il est possible alors de ne conserver que les features ayant le plus de poids grâce à la « **feature importance** » qui donne les modèles à base d'arbres (le pipeline de préparation des données peut même être optimisé en ce sens).

Puis le modèle final est réentraîné **en conservant les hyperparamètres ayant conduit aux meilleurs résultats** mais cette fois sur un sous ensemble de features plus restreint. Une sauvegarde du modèle « **fitté** » qui sera utilisée par la suite afin de réaliser des inférences à partir de nouveaux datasets.

Il conviendrait également de **suivre les métriques obtenues**, ainsi que de calculer régulièrement des statistiques des features afin de s'assurer de l'absence de « **drift** ».

5.3. Améliorations potentielles

Différentes pistes supplémentaires peuvent être explorées afin d'améliorer encore l'efficacité des modèles :

- **La qualité des données joue un rôle primordial** dans la performance de tout modèle (« garbage in garbage out »). Dans ce projet, nous nous sommes rendu compte que certaines mesures météo peuvent manquer, que certains horaires présentent des trous, que parfois plusieurs mesures sont utilisées pour un même horaire de départ ou d'arrivée d'un vol. La préparation de données est la partie qui prend le plus de temps. Néanmoins, il aurait probablement été possible d'améliorer la qualité de nos données par divers moyens :
 - o Utilisation d'open data pour combler ou affiner certaines données manquantes ou corriger des valeurs erronées.
 - o De même que du web scraping de site météo : à noter que cette technique d'obtention de nouvelles data est à pratiquer avec précaution : si son usage n'est pas forcément explicitement interdit, il n'en demeure pas pour autant toujours autorisé. Certaines entreprises préfèrent parfois s'en dispenser là où d'autres sont parfois moins regardantes...

- Réaliser des interpolations linéaires en fonction du temps et ce pour chaque sonde afin de combler plus finement les valeurs manquantes. De même qu'un lissage de certaines valeurs peut certainement permettre de s'affranchir d'« outliers ».
- **Enrichissement des données** : une autre possibilité consiste à ajouter d'autres informations qui pourraient se révéler intéressantes comme le jour de la semaine, l'heure du vol, le mois de l'année qui peuvent faire intervenir des saisonnalités...
- **Données plus « fraîches / récentes » et réentrainement plus fréquent** : un autre aspect pourrait se révéler très intéressant à explorer – compte tenu d'un certain dérèglement climatique - la variabilité des données météo ou les moyennes / valeurs normales sur une période peuvent changer. Un suivi dans le temps peut s'avérer nécessaire tout comme un réentrainement à intervalle régulier pour tenir compte des effets météo plus récents.
- **Curse of dimensionality** : Le fléau de la dimension ou malédiction de la dimension désigne divers phénomènes qui ont lieu lorsque l'on cherche à analyser ou exploiter des données dans des espaces de grande dimension. Lorsque le nombre de dimensions augmente, le volume de l'espace croît rapidement si bien que les données se retrouvent « isolées » et deviennent éparques.

De plus les random forest appréhendent mal de multiples colonnes car chaque estimateur / arbre ne voit qu'une partie des features. Le paramètre « numTrees » dans spark possède une limite (certainement pour des questions de scalabilité). De fait, il est possible que certaines features échappent au random forest ou soit moins exploitées. Notre usecase présente cependant l'avantage de comporter un grand nombre (toute proportion gardée) de ligne (plus de 2 millions sans undersampling) ; ce qui va dans le bon sens car plus notre dataset possède de colonnes plus il est nécessaire pour le modèle de bénéficier d'un grand nombre d'observations / de lignes. Malgré tout, la scalabilité peut s'en retrouver impactée.

- **Time series** : au final une dernière approche pourrait s'avérer également très intéressante – il s'agit d'aborder les évolutions météo dans le temps comme des « times séries ». A savoir que les valeurs passées peuvent servir à en prédire les nouvelles. Néanmoins, concernant les prévisions météorologiques, cette approche peut se révéler très complexe et difficile à scaler.

Conclusion

Au final, le projet « Flight » a été très enrichissant à différents points de vue :

Ce fut d'une part l'occasion de mettre en pratique nos connaissances sur un cas très concrets : on peut aisément mesurer l'impacts importants et bien réel des retards engendrés pour bon nombre de vols par une météo peu clémene.

Par ailleurs, comme pour tous les cas d'usages bien réels nous avons eu à traiter des données de qualité variable présentant des doublons, avec des valeurs manquantes ou parfois erronées... Sachant que la qualité des données demeure justement primordiale pour toute la partie analytics qui dépend de ces data. Le traitement, le nettoyage ainsi que de manière générale tout le processing a nécessité un gros investissement de notre part. Autant d'efforts qui sont à l'image de l'importance de la tache en entreprise.

Ce projet fut très challengeant tant d'un point vue technique que fonctionnel : il nous a amenés à nous poser une multitude de questions sur les données, à explorer celles-ci pour y trouver des réponses tout en menant en parallèle des recherches en ligne. Avec déjà quelques gigas de data, nous avons pu vite mesurer l'influence de la volumétrie et de la scalabilité des traitements / applications.

Ce fut aussi l'opportunité d'appréhender tout le cycle de vie de la data de façon similaire à ce que l'on peut retrouver couramment en entreprise par le biais d'une architecture reposant sur des tables bronze, silver et gold. Nous permettant ainsi de mieux percevoir les rôles et taches des Data Engineer, Data Analyst et Data Scientist. Cela tout en utilisant des stacks techniques « big data » parmi les plus demandées / recherchées : les développements spark / scala ont été mené à la fois sur un cluster Hadoop on premise, sur Databricks ainsi que chez un cloud provider.

Nous sommes finalement parvenus à des résultats très proches du papier de référence “*Using Scalable Data Mining for Predicting Flight Delays*” (L. Belcastro, Fabrizio Marozzo, Domenico Talia, Paolo Trunfio). Nos métriques en termes de precision / recall demeurent très similaires à celles présentées par les auteurs de l'étude originale.

Pour terminer, ce projet a nécessité beaucoup d'échanges entre nous : ce fut également une très bonne occasion de parfaire nos connaissances et de partager celles-ci !

Annexes

Annexe 1: Using Scalable Data Mining for Predicting Flight Delays



FlightDelay-TIST-PrePrint.pdf

Annexe 2: « Local Climatological Data (LCD) »



weather_documentation.pdf

L'ensemble des codes, scripts, configurations, et notebooks sont disponibles dans le repository Github suivant:

<https://github.com/YoloCall/FlightFinal>

Ce qui nous a permis de développer chacun dans une branche distincte puis de fusionner / merger l'ensemble plus facilement.