# Rapport Projet BIG DATA SPARK

Réalisé par : **OUASSOU ANAS** et **MOUJTAHIDE MOHAMED ADNANE**, *3A MIS*

*Lien Github de notre repo de ce projet : https://github.com/anasouassou/spark\_stock\_prediction/tree/main*

### **1. Présentation de Spark**

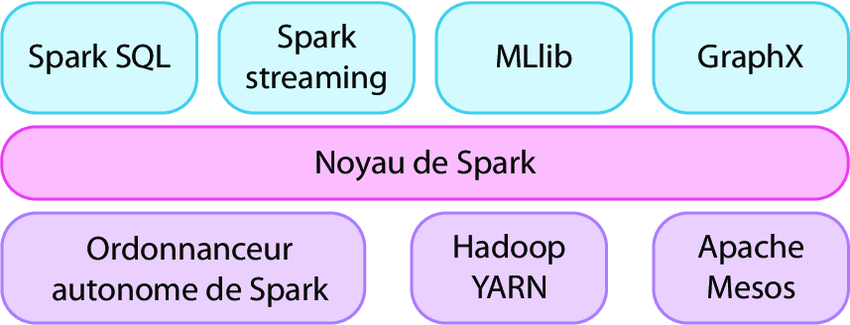
Apache Spark est un framework open-source conçu pour le traitement de données massives (**Big Data**) de manière **rapide** et **distribuée**. Il a été développé en 2009 au sein de l’**AMPLab** de l’Université de Californie à Berkeley et est aujourd'hui un projet majeur de la **Fondation Apache**. Spark se distingue par ses capacités de **traitement en mémoire**, ce qui le rend beaucoup plus performant que Hadoop MapReduce pour des analyses interactives et complexes.

L’architecture modulaire de Spark permet de traiter différents types de données (structurées, non structurées, en temps réel, etc.) tout en offrant un large éventail d'outils pour l'analyse, le **Machine Learning**, le **traitement de graphes** et les requêtes SQL.

Spark repose sur son concept central : les **RDD** (**Resilient Distributed Datasets**), qui sont des collections de données distribuées sur un cluster. Les RDD permettent le traitement des données de manière tolérante aux pannes tout en optimisant l'exécution des tâches en mémoire.

### **1.1 Composants de Spark**

L’architecture de Spark est modulaire et se compose de plusieurs couches et bibliothèques. Voici une description détaillée des principaux **composants de Spark** représentés dans l'image ci-jointe.



#### **A. Noyau de Spark (Spark Core)**

Le **noyau de Spark** est le composant fondamental sur lequel reposent tous les autres modules. Il fournit les fonctionnalités de base pour le traitement distribué des données.

* **Gestion des RDD** :  
  Les **RDD** (**Resilient Distributed Dataset**) sont des ensembles de données **tolérants aux pannes** qui permettent d’exécuter des opérations de transformation et d’action sur des clusters.
  + **Transformations** : Filtrage, mappage, jointure, agrégation, etc.
  + **Actions** : Collecte des résultats, comptage des données, affichage des résultats, etc.
* **Gestion des tâches et des ressources** :  
  Le noyau de Spark gère l’ordonnancement des tâches et leur distribution sur plusieurs nœuds du cluster.
* **Optimisation du traitement en mémoire** :  
  Spark utilise des mécanismes de **caching** et de **persist** pour conserver les données en mémoire, évitant ainsi des lectures et écritures coûteuses sur disque.

#### **B. Spark SQL**

**Spark SQL** est le module de Spark qui permet de manipuler des **données structurées** à l’aide du langage **SQL**. Il permet de créer, interroger et gérer des données stockées dans différents formats.

* **Caractéristiques principales** :
  + Exécution de requêtes SQL standardisées.
  + Support de **DataFrames** et **Datasets** pour une manipulation efficace des données en mémoire.
  + Intégration avec des systèmes de stockage comme **Hive**, **HDFS**, **Parquet**, **ORC**, **Avro**, et **JDBC**.
* **Exemple de code PySpark** **avec SparkSQL** :

|  |
| --- |
| * from pyspark.sql import SparkSession * # Créer une session Spark * spark = SparkSession.builder.appName("SparkSQL Example").getOrCreate() * # Charger un DataFrame * df = spark.read.csv("data.csv", header=True, inferSchema=True) * # Exécuter une requête SQL * df.createOrReplaceTempView("table\_name") * spark.sql("SELECT \* FROM table\_name WHERE column = 'value'").show() |

#### **C. Spark Streaming**

**Spark Streaming** permet le traitement **temps réel** des flux de données. Contrairement aux systèmes traditionnels qui nécessitent des batchs, Spark Streaming divise les flux entrants en **micro-batchs** pour un traitement distribué en mémoire.

* **Sources de données prises en charge** :
  + Kafka
  + HDFS
  + Sockets TCP
  + Amazon Kinesis
* **Cas d’usage** :
  + Analyse de données temps réel provenant de capteurs IoT.
  + Analyse des journaux de serveurs.
* **Exemple de code PySpark Streaming** :

|  |
| --- |
| * from pyspark import SparkContext * from pyspark.streaming import StreamingContext * sc = SparkContext("local[2]", "NetworkWordCount") * ssc = StreamingContext(sc, 1) * # Flux de données à partir d'un socket TCP * lines = ssc.socketTextStream("localhost", 9999) * words = lines.flatMap(lambda line: line.split(" ")) * wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a, b: a + b) * wordCounts.pprint() * ssc.start() * ssc.awaitTermination() |

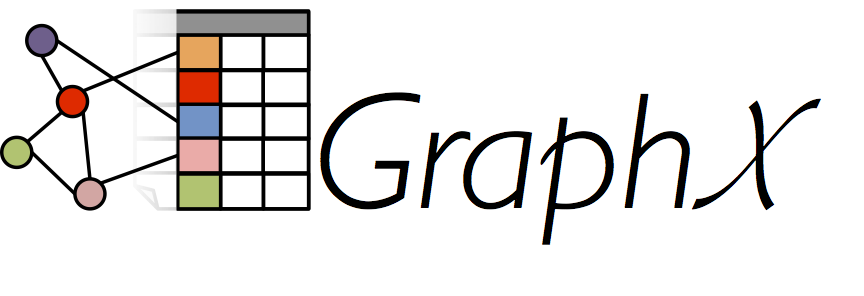
#### **D. MLlib (Machine Learning Library)**

**MLlib** est la bibliothèque de **Machine Learning** intégrée à Spark. Elle propose des algorithmes optimisés pour le traitement distribué.

* **Fonctionnalités principales** :
  + **Classification** : Régression logistique, SVM.
  + **Régression** : Régression linéaire, arbre de décision.
  + **Clustering** : K-Means.
  + **Collaborative Filtering** : Recommandation.
* **Exemple d’utilisation** :

|  |
| --- |
| * from pyspark.ml.classification import LogisticRegression * from pyspark.ml.feature import VectorAssembler * # Préparation des données * assembler = VectorAssembler(inputCols=["feature1", "feature2"], outputCol="features") * data = assembler.transform(df) * # Entraîner un modèle * lr = LogisticRegression(labelCol="label", featuresCol="features") * model = lr.fit(data) * # Prédictions * predictions = model.transform(data) * predictions.show() |

#### **E. GraphX**



**GraphX** est le module de traitement des graphes dans Spark. Il est utilisé pour l’analyse des relations entre les données sous forme de graphes.

* **Exemples d’algorithmes pris en charge** :
  + **PageRank** : Pour les systèmes de recommandation et les moteurs de recherche.
  + **Shortest Path** : Trouver le chemin le plus court entre deux points.

#### **F. Ordonnanceurs et Intégrations**

Spark s'intègre avec plusieurs solutions pour gérer les tâches :

* **Ordonnanceur autonome de Spark** : Gestion native des ressources.
* **Hadoop YARN** : Intégration avec l’écosystème Hadoop.
* **Apache Mesos** : Gestion avancée des clusters.

### **2. Présentation Succincte des Outils Utilisés et des Modèles**

#### **Rappel : Apache Spark MLlib**

Comme déjà expliqué, **MLlib** est la bibliothèque d’apprentissage automatique d’Apache Spark. Elle a été utilisée pour développer et entraîner les modèles de machine learning de ce projet grâce à sa capacité à traiter de grandes quantités de données et à fournir des algorithmes performants.

#### **Autres Outils Utilisés**

1. **PySpark** : Interface Python pour manipuler les données et intégrer MLlib.
2. **SparkSQL** : Utilisé pour exécuter des requêtes SQL distribuées sur les données.
3. **Matplotlib** et **Pandas** : Pour visualiser et manipuler les résultats.

***Présentation des Trois Modèles Utilisés (avec Code)***

#### **A. Régression Linéaire (Linear Regression)**

* **Description** :
  + Un modèle supervisé établissant une relation linéaire entre les variables explicatives (**features**) et la variable cible (**label**).
  + Optimisé avec régularisation Lasso (L1) et Ridge (L2).
* **Code Résumé** :

|  |
| --- |
| * from pyspark.ml.regression import LinearRegression * from pyspark.ml.tuning import ParamGridBuilder, CrossValidator * from pyspark.ml.evaluation import RegressionEvaluator * # Initialisation du modèle * lr = LinearRegression(featuresCol="features", labelCol="Close") * # Grille de paramètres pour l'optimisation * param\_grid = ParamGridBuilder() \ * .addGrid(lr.regParam, [0.01, 0.1, 0.5]) \ * .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \ * .build() * # Validation croisée * cv = CrossValidator(estimator=lr, estimatorParamMaps=param\_grid, evaluator=RegressionEvaluator(labelCol="Close"), numFolds=3) * # Entraînement et sauvegarde * lr\_model = cv.fit(train\_data) * lr\_model.bestModel.write().overwrite().save("models/linear\_regression\_optimized") |

#### **B. Random Forest Regressor**

* **Description** :
  + Algorithme basé sur des arbres de décision. Il agrège plusieurs arbres pour capturer des relations complexes et non linéaires.
* **Code Résumé** :

|  |
| --- |
| * from pyspark.ml.regression import RandomForestRegressor * # Initialisation du modèle * rf = RandomForestRegressor(featuresCol="features", labelCol="Close", numTrees=50, maxDepth=10) * # Entraînement et sauvegarde * rf\_model = rf.fit(train\_data) * rf\_model.write().overwrite().save("models/random\_forest") |

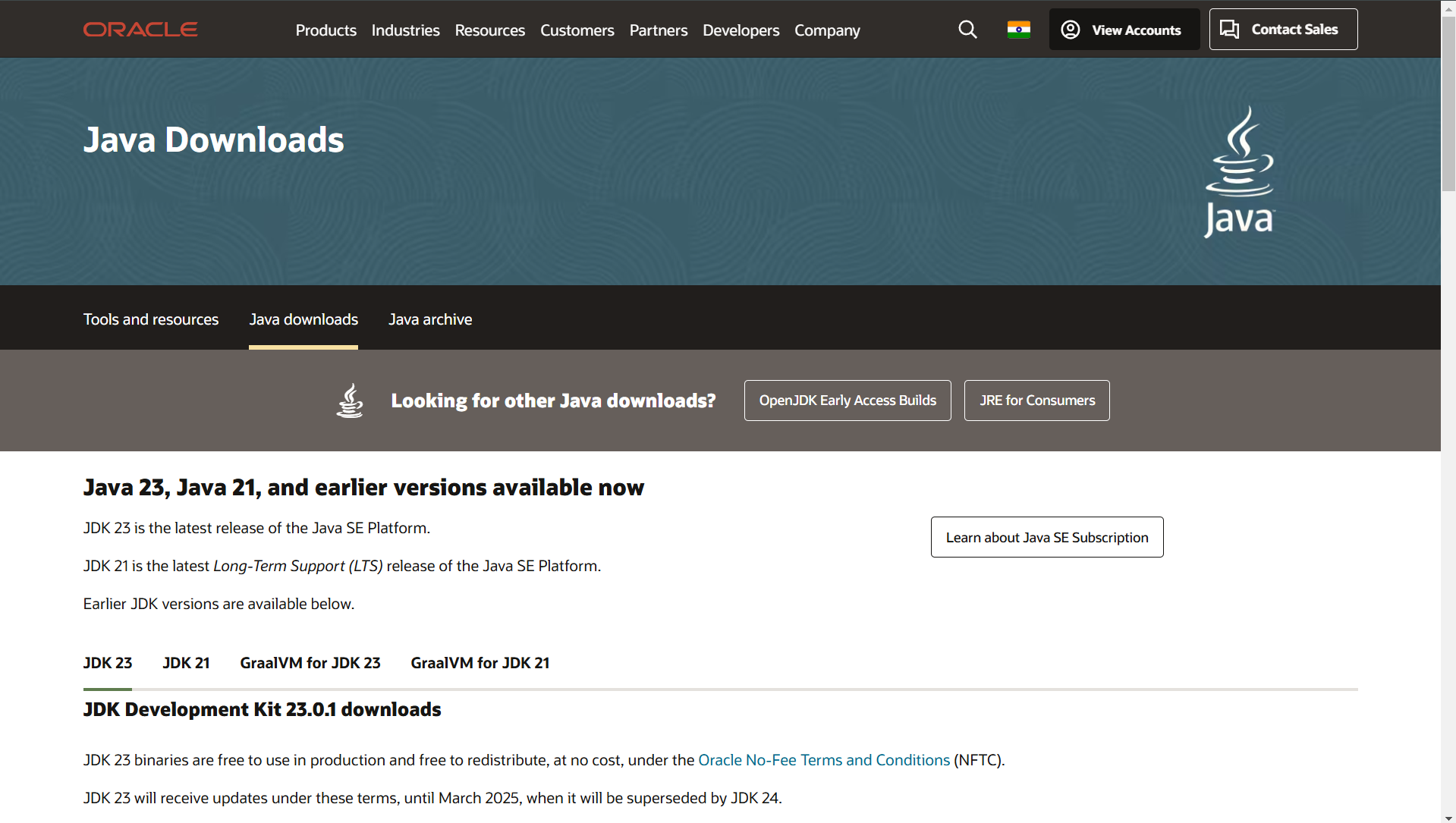
#### **3. Modèle Temporel Basé sur les Lags**

* **Description** :
  + Approche adaptée aux séries temporelles, utilisant les valeurs retardées (**lags**) du prix de clôture pour capturer les tendances historiques.
* **Code Résumé** :

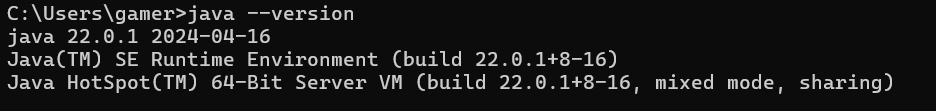
|  |
| --- |
| * from pyspark.sql.functions import lag * from pyspark.sql.window import Window * from pyspark.ml.feature import VectorAssembler * # Création de colonnes de lags * window\_spec = Window.orderBy("Date") * data = data.withColumn("Lag\_1", lag("Close", 1).over(window\_spec)) * data = data.withColumn("Lag\_2", lag("Close", 2).over(window\_spec)) * data = data.withColumn("Lag\_3", lag("Close", 3).over(window\_spec)).dropna() * # Assemblage des caractéristiques * assembler = VectorAssembler(inputCols=["Lag\_1", "Lag\_2", "Lag\_3"], outputCol="features") * time\_model\_data = assembler.transform(data).select("features", "Close") * # Entraînement et sauvegarde * time\_lr = LinearRegression(featuresCol="features", labelCol="Close") * time\_lr\_model = time\_lr.fit(time\_model\_data) * time\_lr\_model.write().overwrite().save("models/time\_series\_lr") |

### **3. Étapes d'Installation de Spark avec Commentaires**

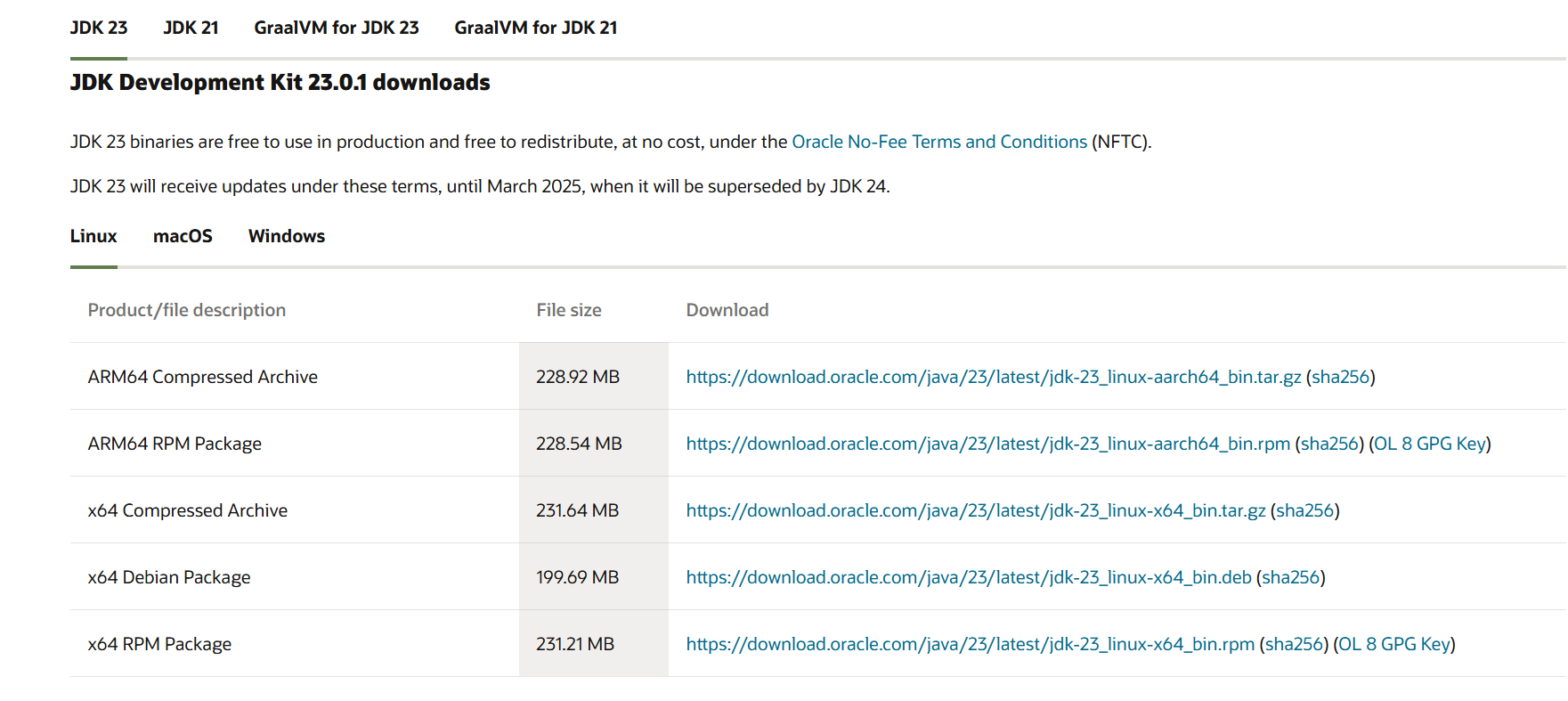
***3.1. Installer JAVA (JDK)***

On se rend sur la page officielle d’Oracle pour installer Java sur notre machine :  


La version Java 22 est déjà installée sur la machine :

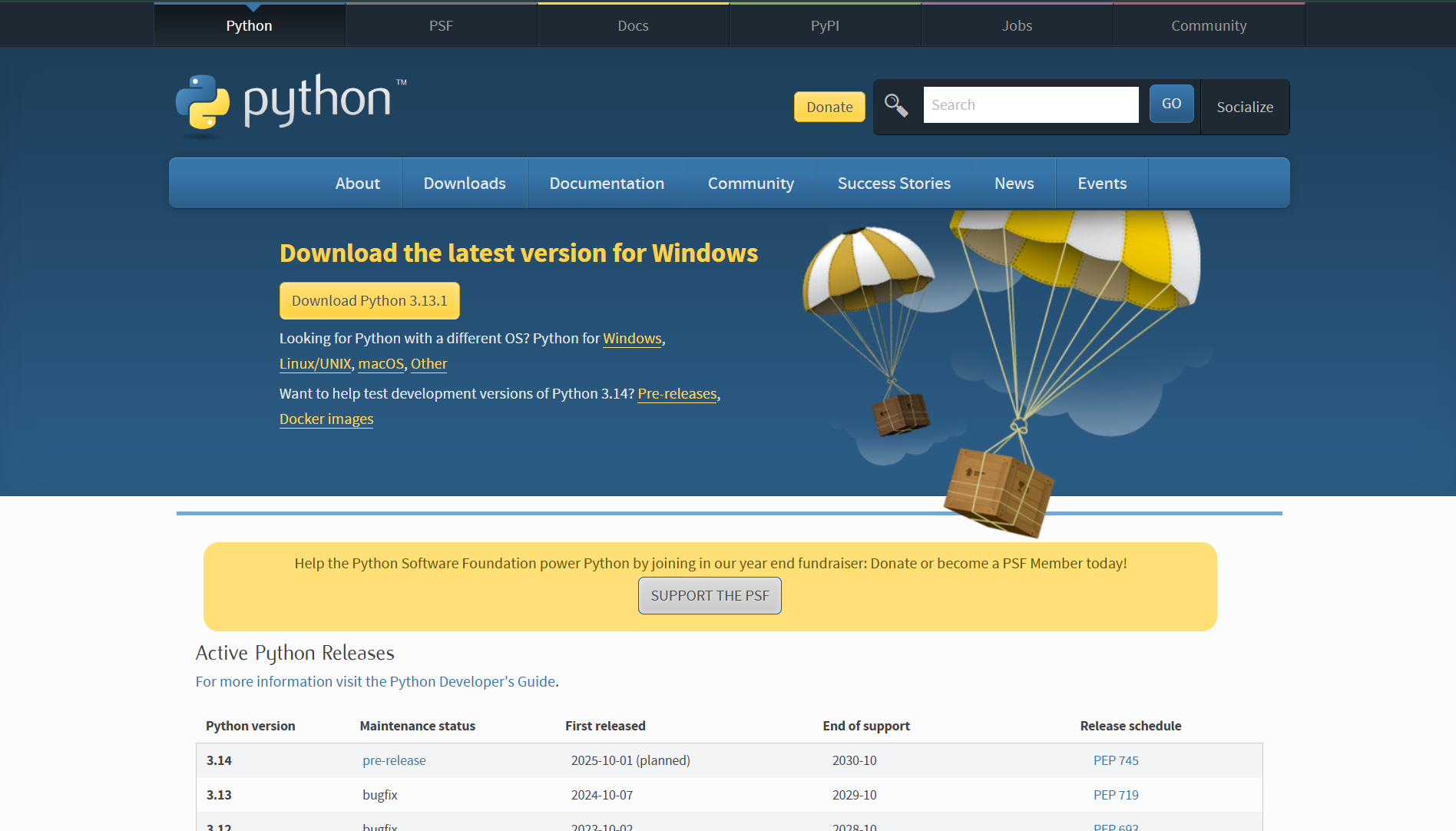


Mais si l’on veut l’installer from scratch, on sélectionne la version de Java souhaitée et on suit les étapes, qui sont triviales :

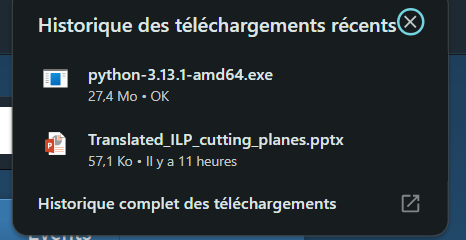


***3.2. Installer Python***

Même chose pour Java : on se rend sur le site officiel de Python pour l’installer.



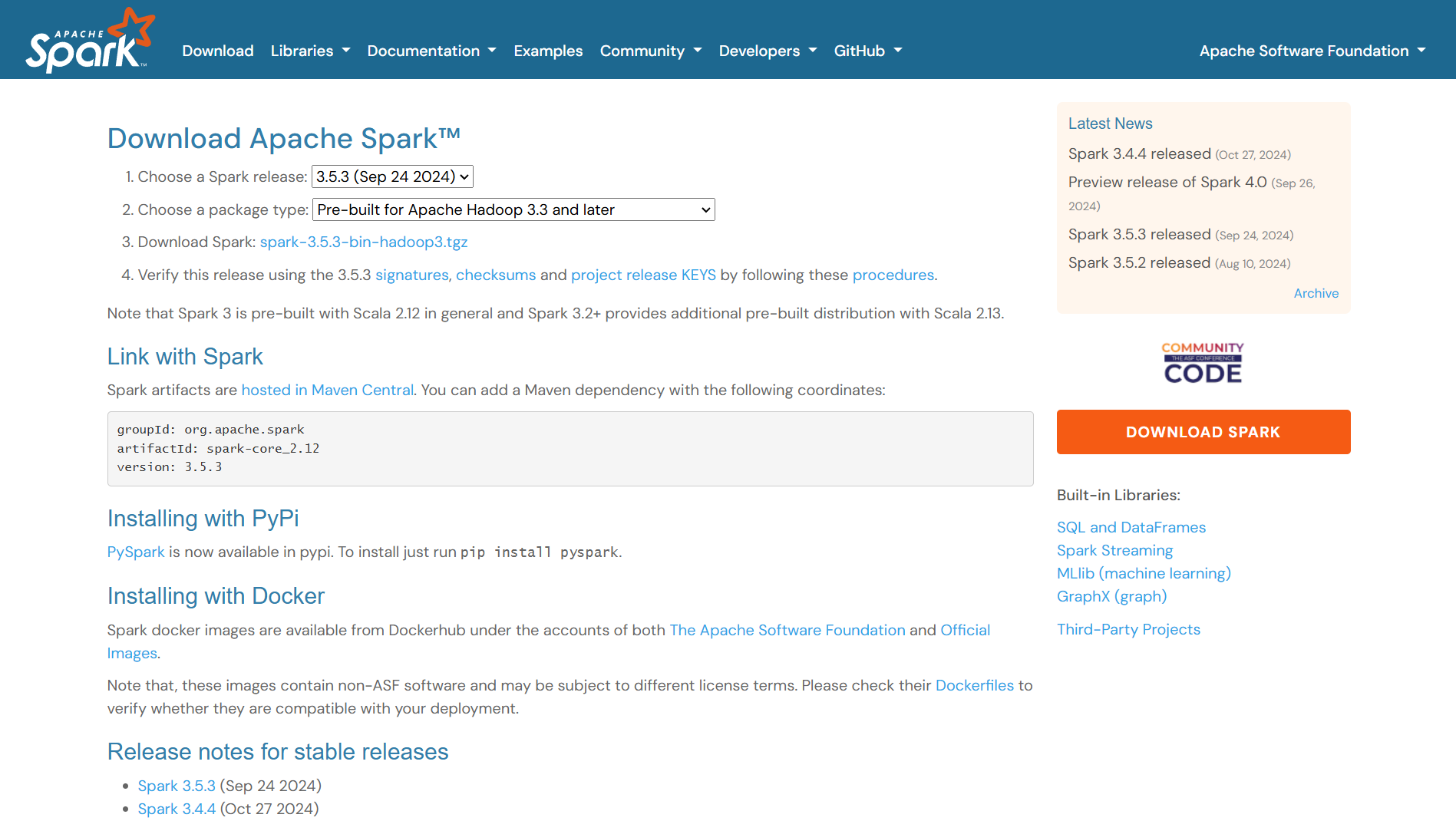
On choisit la version que l’on souhaite :



Puis, on suit les étapes, et Python sera installé sur notre appareil.

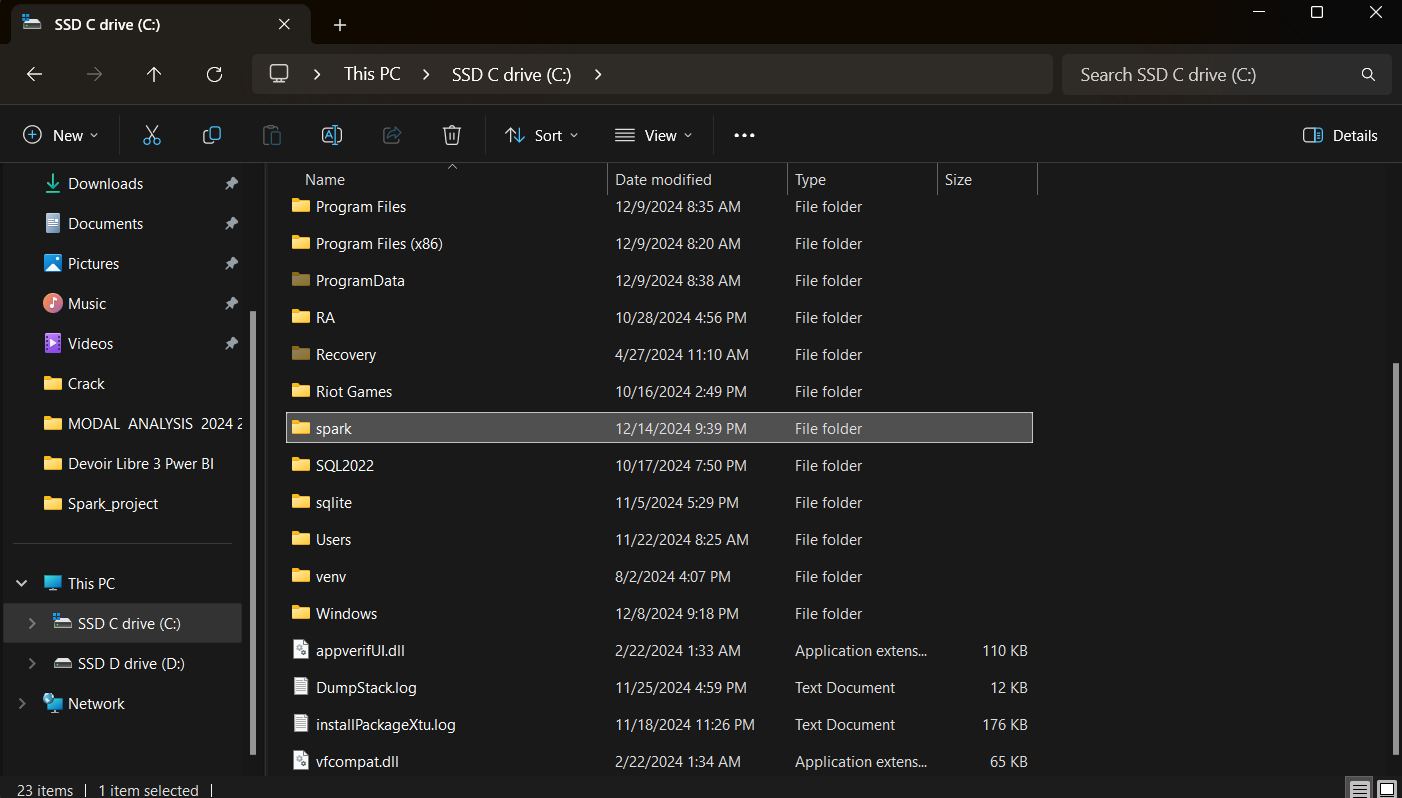
***3.2. Installer Spark***

On se rend sur le site officiel d’Apache pour obtenir Spark :

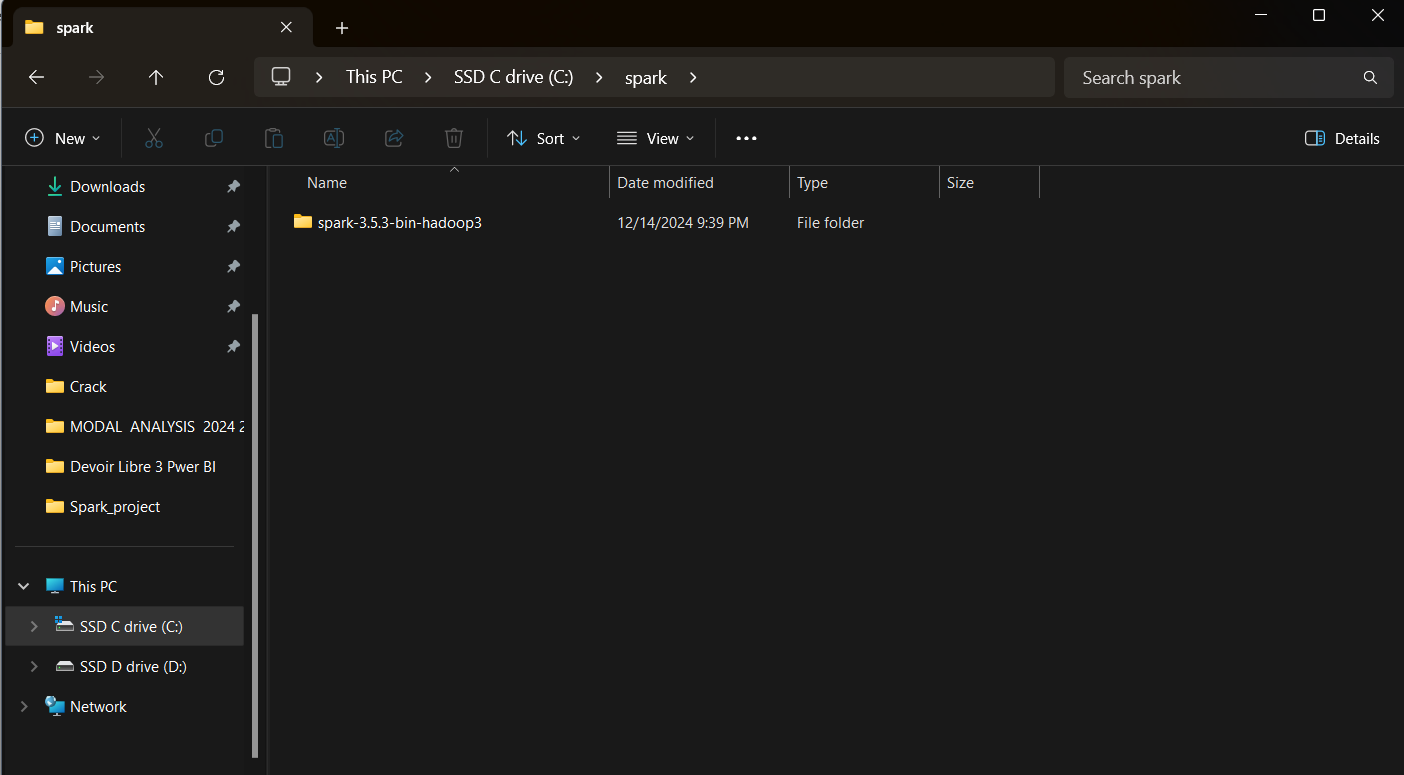
******

On choisit la version souhaitée, dans notre cas ***Spark 3.5.3 (2024)***, ainsi que le package "***Pre-built for Apache Hadoop 3.3 and later***".

On clique sur "Download Spark", puis on extrait le fichier et on le déplace vers le disque **C/**.

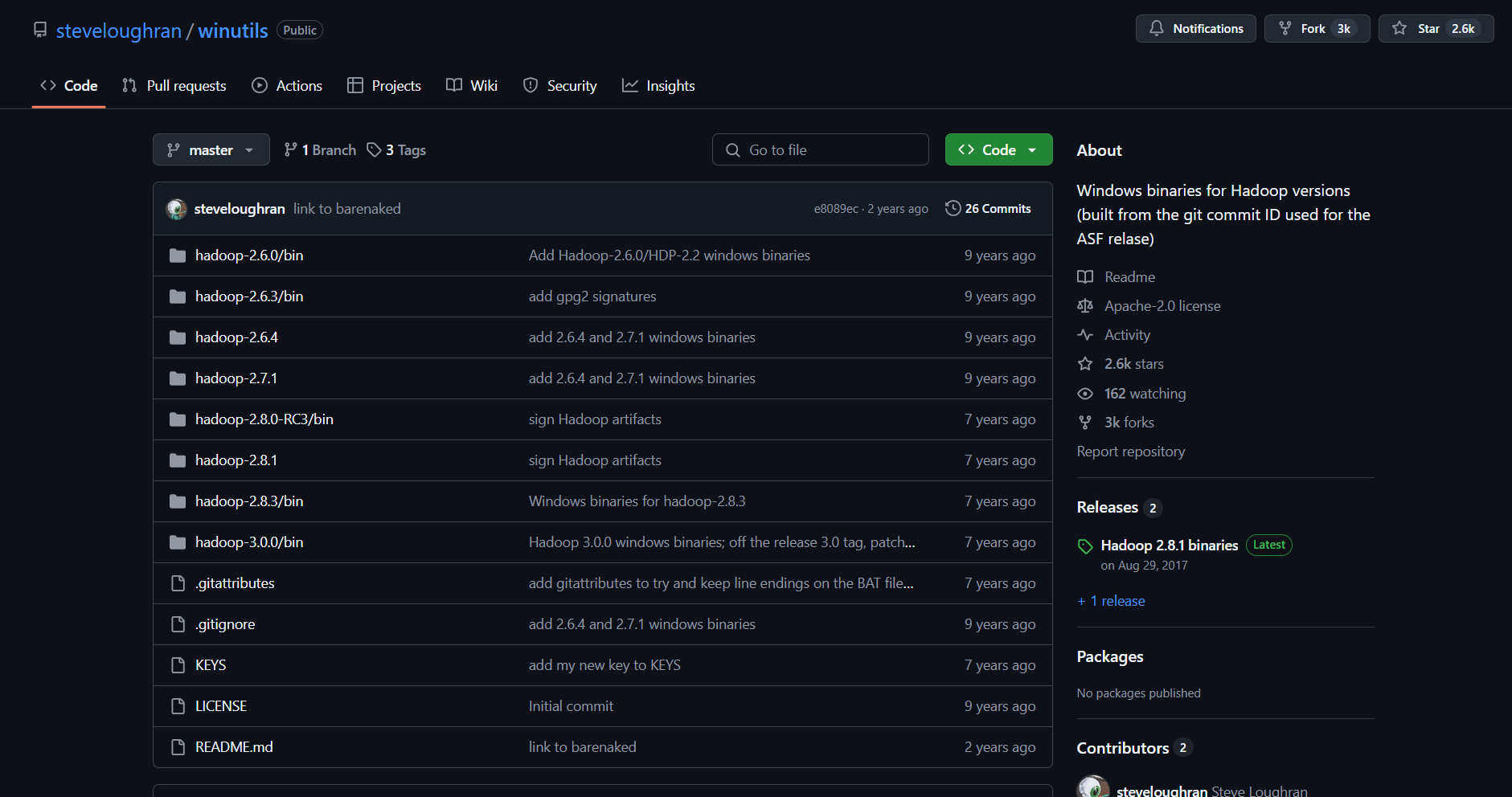


Le fichier Spark contient l’image de Spark téléchargée :

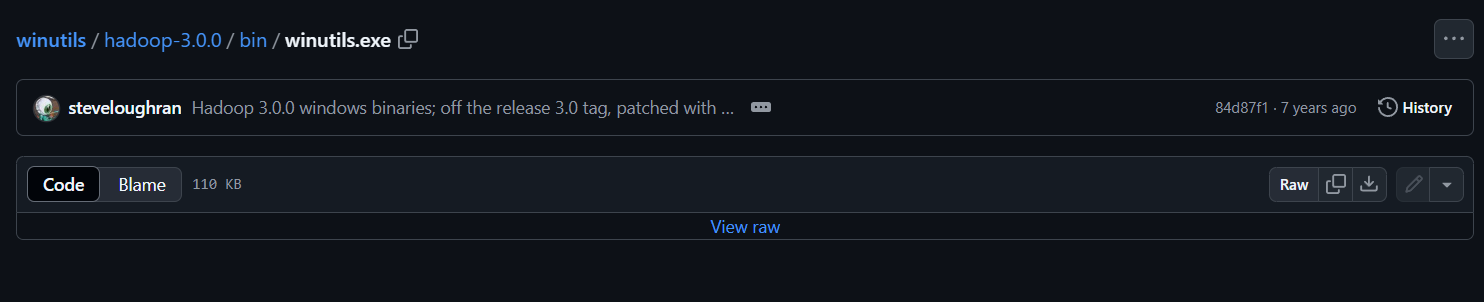


***3.2. Installer Winutils.exe***

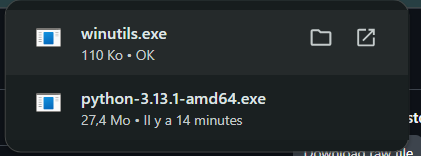
Puis, pour lier Spark avec Hadoop, nous devons télécharger **Winutils.exe**, que l’on peut trouver dans ce dépôt GitHub : [***https://github.com/steveloughran/winutils***](https://github.com/steveloughran/winutils) ***:***



On choisit ***hadoop-3.0.0/bin, puis on telecharge winutils.exe :***

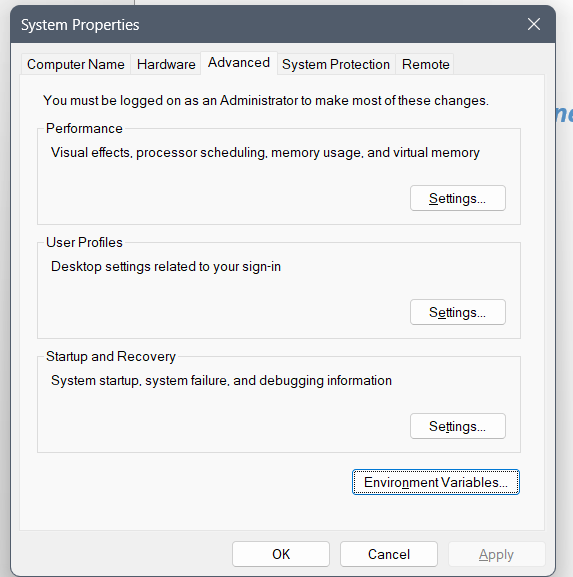


Et on a :

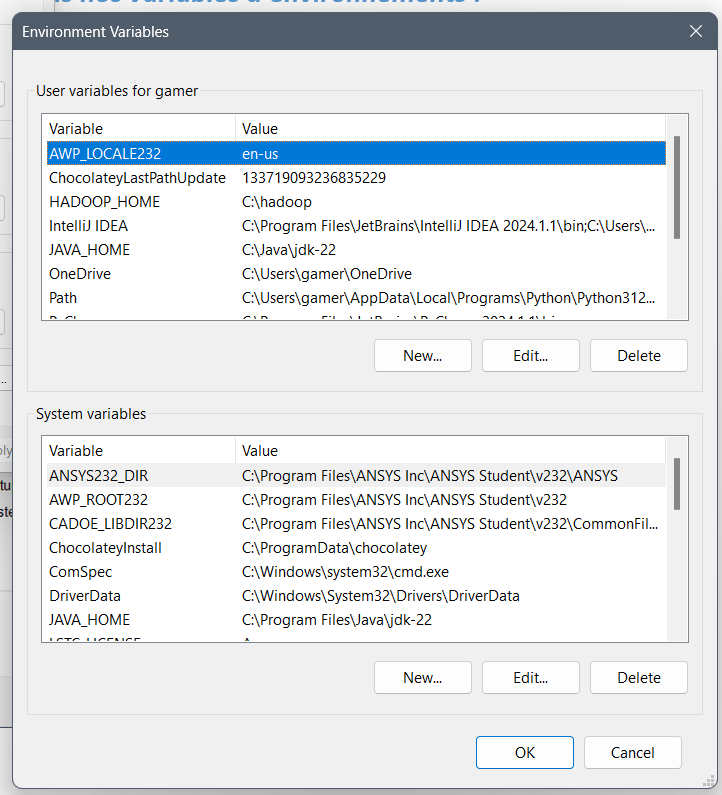
****

***3.2. Variables d’environnements :***

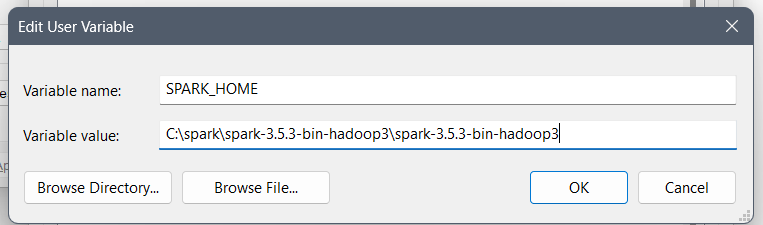
On ajoute les chemins des dossiers de Spark et de Hadoop (winutils.exe) dans nos variables d’environnement :

****

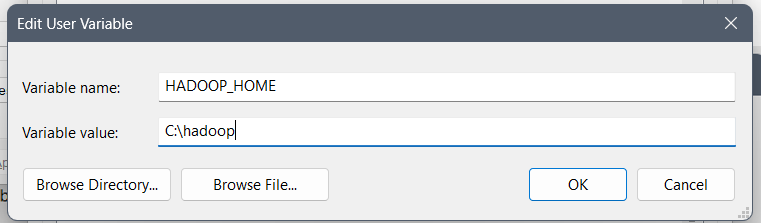
Puis :

****

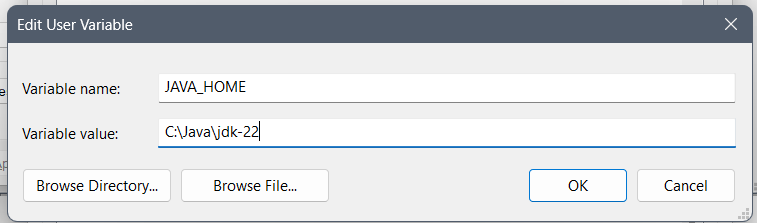
On ajoute le chemin de Spark :



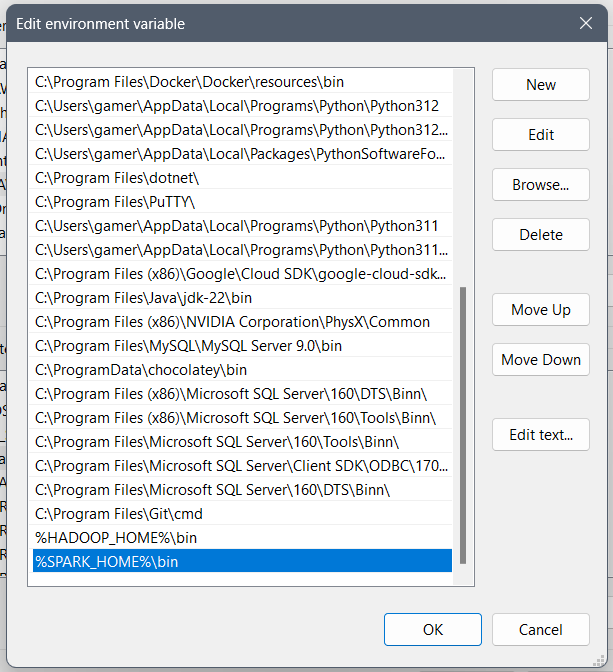
Puis le chemin de Hadoop (winutils.exe) :



Même chose pour Java\_Home :

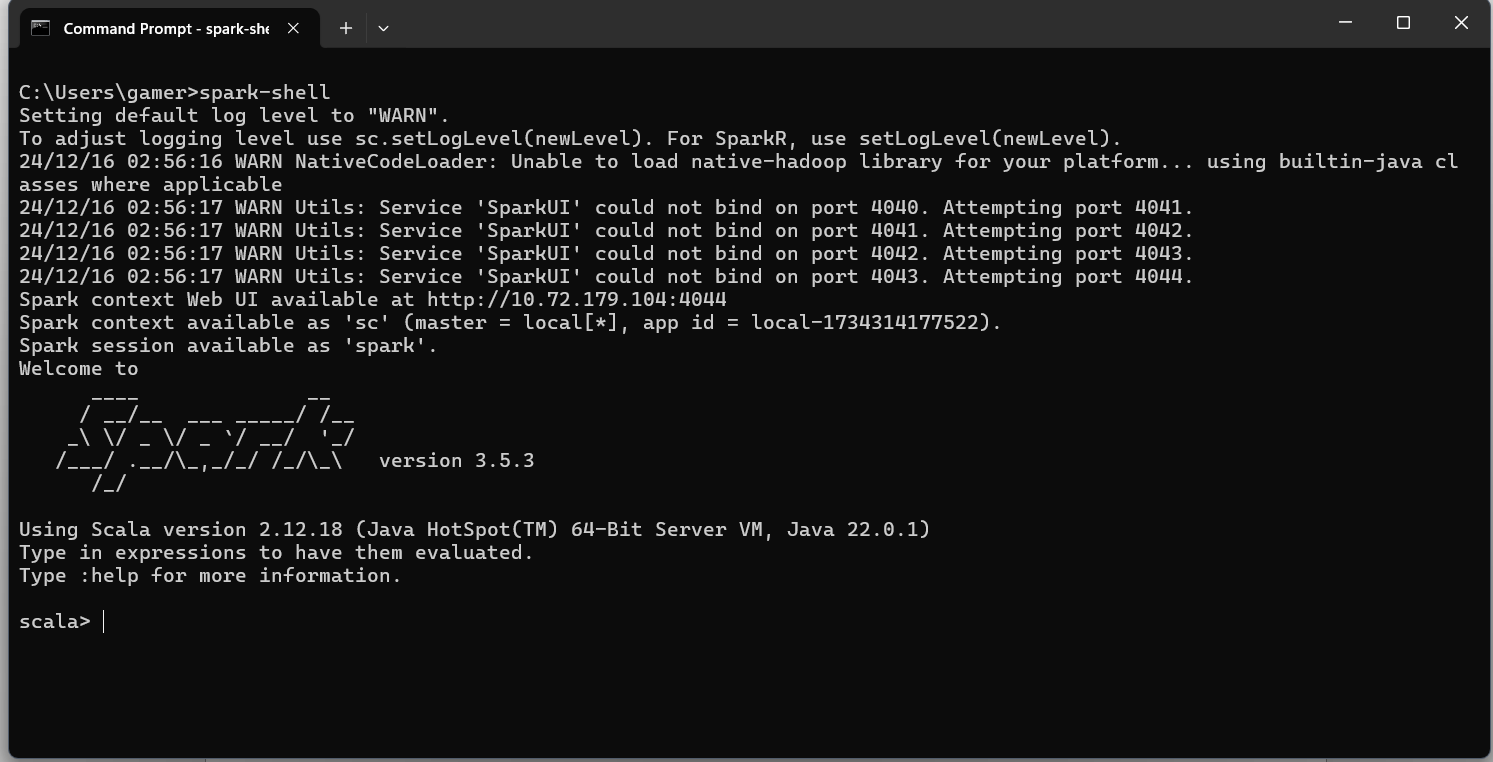


Puis, on ajoute les chemins de ***Spark*** et de *Hadoop* (***winutils.exe***) dans le **PATH** :

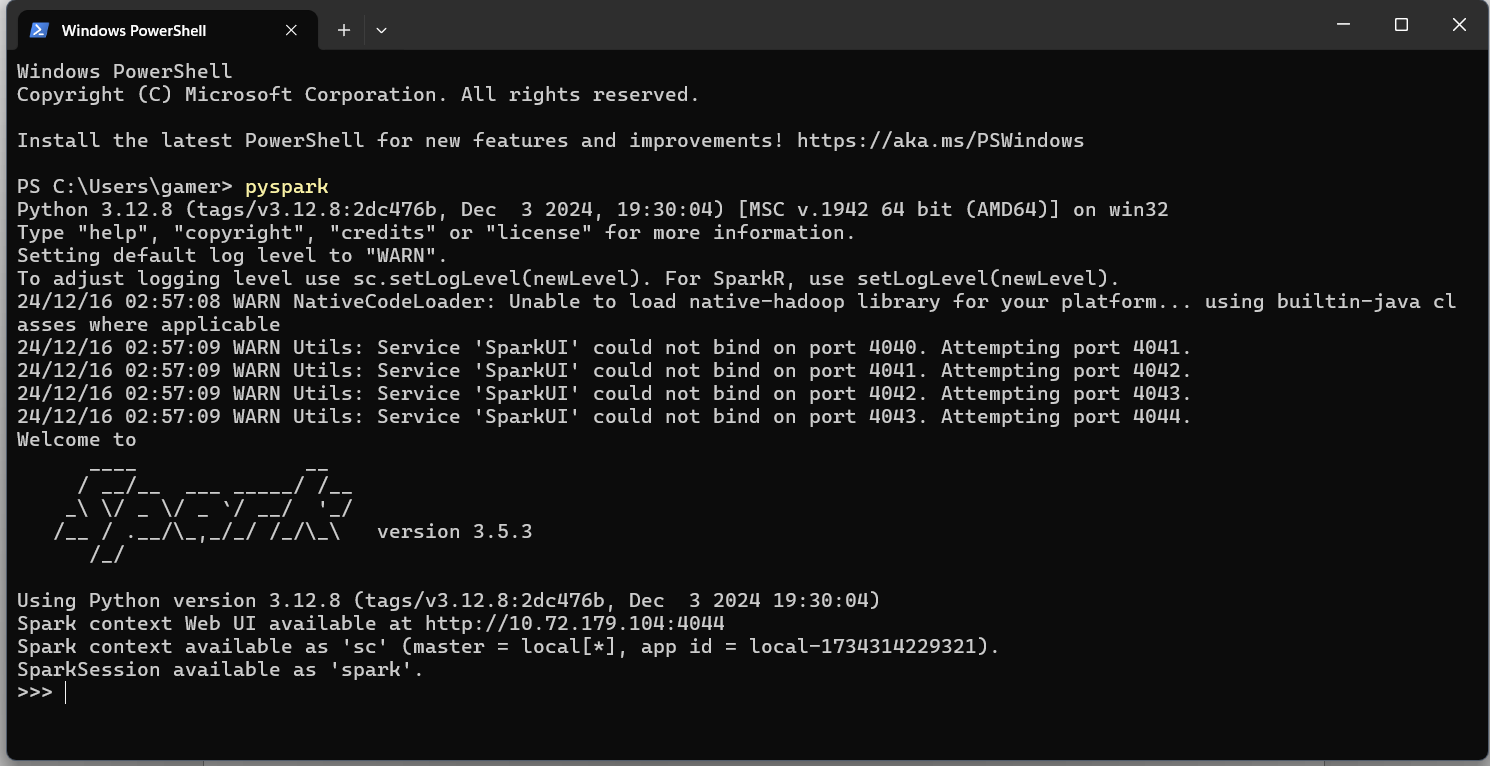


Ainsi, nous terminons l'installation et la configuration de toutes les variables d'environnement nécessaires.

Si nous vérifions Spark dans le terminal avec la commande **spark-shell**, nous obtiendrons :



Et si nous vérifions PySpark avec la commande **pyspark**, nous obtiendrons :



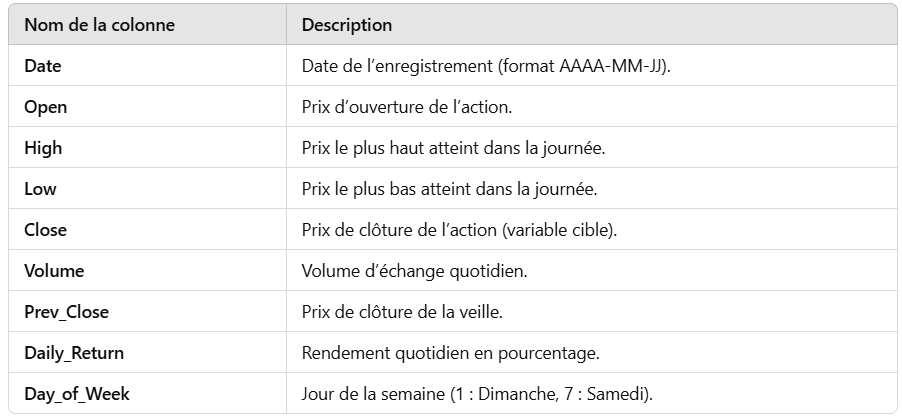
### **4. Présentation du Dataset, de la Variable à Prédire et de l'Algorithme de ML Choisi :**

#### **4.1 Présentation du Dataset**

Le dataset utilisé dans ce projet est un ensemble de données financières contenant les prix des actions d’**Apple** sur une période allant de **1980 à nos jours**. Les données incluent plusieurs informations utiles pour l’analyse des tendances boursières et la prédiction des prix.

**Caractéristiques du Dataset** :

* **Source** : Provenant d’une source fiable (HuggingFace ou API de données financières).
* **Format** : JSON initialement converti en **Parquet** et **CSV** pour faciliter le traitement dans Spark.
* **Nombre de colonnes** : 9 colonnes principales.
* **Nombre de lignes** : Plusieurs milliers de lignes couvrant l’évolution des prix sur plusieurs décennies.



**Prétraitement Appliqué** :

* Création de nouvelles colonnes comme **Prev\_Close**, **Daily\_Return**, et **Day\_of\_Week**.
* Suppression des lignes avec des valeurs manquantes.
* Filtrage des années récentes pour concentrer l’analyse sur les tendances actuelles.

#### **4.2 Variable à Prédire**

La variable cible dans ce projet est le **prix de clôture** de l’action (**Close**), qui représente la valeur de l’action à la fin de chaque journée boursière.

* Cette variable est continue, ce qui en fait un problème de **régression**.
* La prédiction du prix de clôture est cruciale pour les investisseurs afin d’anticiper les tendances du marché.

#### **4.3 Algorithmes de Machine Learning Choisis**

Pour prédire la variable Close, trois algorithmes de **régression** ont été utilisés dans le cadre de **PySpark MLlib** :

##### **A. Régression Linéaire (Linear Regression)**

* **Description** :  
  La régression linéaire est un modèle simple qui établit une relation linéaire entre les variables explicatives (**features**) et la variable cible (**label**).
* **Pourquoi cet algorithme ?**
  + Il est facile à interpréter et rapide à exécuter.
  + C’est une première approche pour évaluer la relation entre les variables d'entrée et la cible.
* **Amélioration** :
  + Ajout d’une **régularisation Lasso (L1)** et Ridge (L2) pour éviter le surapprentissage.
  + Optimisation des hyperparamètres avec **validation croisée**.

##### **B. Random Forest Regressor**

* **Description** :  
  Le Random Forest est un ensemble d’arbres de décision où chaque arbre est entraîné sur un sous-échantillon des données. Les prédictions sont moyennées pour améliorer la robustesse.
* **Pourquoi cet algorithme ?**
  + Il capture les relations **non linéaires** dans les données.
  + Il est robuste face au surapprentissage et offre de bonnes performances avec des données complexes.
* **Hyperparamètres utilisés** :
  + numTrees : Nombre d’arbres (50).
  + maxDepth : Profondeur maximale des arbres (10).

##### **C. Modèle Temporel Basé sur les Lags**

* **Description** :  
  Ce modèle utilise la régression linéaire en exploitant les valeurs retardées (**lags**) de la variable cible Close comme caractéristiques (**features**).
* **Pourquoi cet algorithme ?**
  + Les séries temporelles présentent souvent des dépendances temporelles.
  + Les **lags** permettent d’exploiter les valeurs historiques pour prédire les valeurs futures.
* **Caractéristiques ajoutées** :
  + Lag\_1 : Prix de clôture à la veille.
  + Lag\_2, Lag\_3 : Prix de clôture sur les deux jours précédents.

#### **4.4 Comparaison des Modèles**

Chaque modèle a été entraîné et évalué avec des métriques standard comme :

* **RMSE** (Root Mean Square Error) : Mesure l’erreur quadratique moyenne.
* **R²** (Coefficient de détermination) : Mesure la capacité du modèle à expliquer la variance des données.

Les résultats finaux montrent que :

1. **Régression Linéaire** : Simple et interprétable, mais moins performante pour les relations complexes.
2. **Random Forest** : Offre de meilleures performances grâce à sa capacité à modéliser des relations non linéaires.
3. **Modèle Temporel** : Adapté aux séries chronologiques, mais dépend fortement des valeurs passées.

### **5. Présentation du Pipeline de Traitement des Données**

Dans cette partie, nous présentons le **pipeline complet** de traitement des données, de la préparation initiale jusqu'à l'évaluation des modèles de machine learning. Chaque étape est justifiée et accompagnée de son implémentation.

### **5.1 Prétraitement des Données**

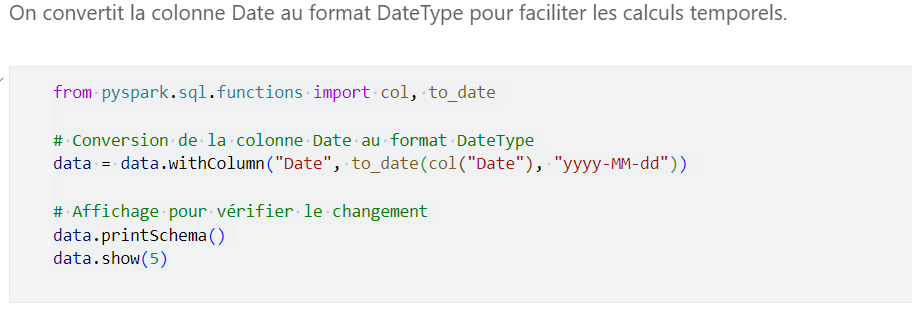
* 1. Using Pyspark

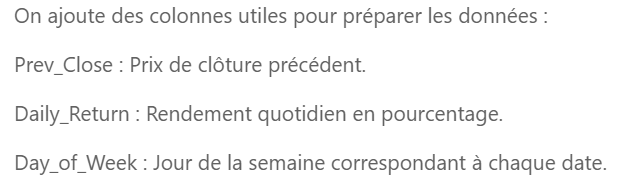
#### **Objectif :**

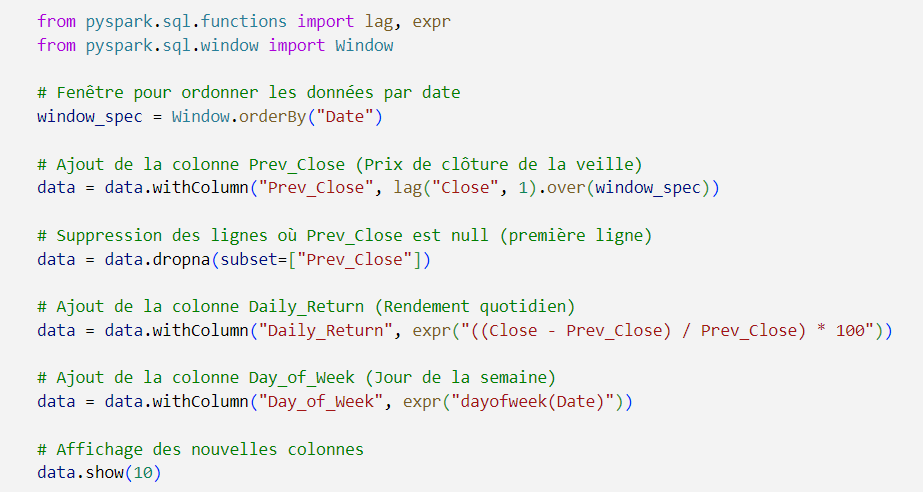
Transformer les données brutes en un format adapté aux modèles de machine learning.

1. **Nettoyage des Données** :
   * Suppression des valeurs manquantes.
   * Ajout de nouvelles colonnes pour enrichir les données :
     + **Prev\_Close** : Prix de clôture de la veille.
     + **Daily\_Return** : Rendement quotidien.
     + **Day\_of\_Week** : Jour de la semaine.

### 





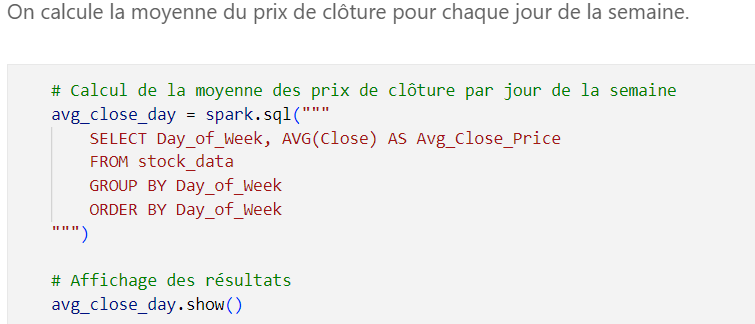
x

* 1. Using SparkSQL

On a aussi utilisé SparkSQL pour faire le prétraitement de notre data :







### **5.2 Assemblage des Caractéristiques (Features)**

#### **Objectif :**

Préparer les colonnes explicatives (**features**) pour les modèles de MLlib.

1. **Sélection des Variables** :
   * Les colonnes choisies pour les features sont :  
     Open, High, Low, Volume, Prev\_Close, Daily\_Return, Day\_of\_Week.
2. **Utilisation de VectorAssembler** pour regrouper les features en une seule colonne.

|  |
| --- |
| 1. from pyspark.ml.feature import VectorAssembler 2. # Définir les colonnes des features 3. feature\_columns = ["Open", "High", "Low", "Volume", "Prev\_Close", "Daily\_Return", "Day\_of\_Week"] 4. # Créer l'assembler 5. assembler = VectorAssembler(inputCols=feature\_columns, outputCol="features") 6. # Appliquer l'assembler 7. data = assembler.transform(data) 8. # Sélectionner les colonnes finales pour le modèle 9. model\_data = data.select("features", "Close").dropna() 10. # Afficher les données prêtes pour la modélisation 11. model\_data.show(5) |

### **5.3 Division en Train et Test**

#### **Objectif :**

Diviser les données en deux ensembles pour l’entraînement et le test des modèles.

|  |
| --- |
| # Diviser les données en train et test  train\_data, test\_data = model\_data.randomSplit([0.8, 0.2], seed=42)  print(f"Taille des données d'entraînement : {train\_data.count()}")  print(f"Taille des données de test : {test\_data.count()}") |

### **5.4 Entraînement des Modèles**

Trois modèles de machine learning ont été utilisés :

#### **1. Régression Linéaire**

|  |
| --- |
| from pyspark.ml.regression import LinearRegression  # Initialiser et entraîner le modèle  lr = LinearRegression(featuresCol="features", labelCol="Close")  lr\_model = lr.fit(train\_data)  # Prédictions  lr\_predictions = lr\_model.transform(test\_data) |

#### **2.** Random Forest Regressor

|  |
| --- |
| from pyspark.ml.regression import RandomForestRegressor  # Initialiser et entraîner le modèle  rf = RandomForestRegressor(featuresCol="features", labelCol="Close", numTrees=50, maxDepth=10)  rf\_model = rf.fit(train\_data)  # Prédictions  rf\_predictions = rf\_model.transform(test\_data) |

#### 3. Modèle Temporel Basé sur les Lags

|  |
| --- |
| from pyspark.ml.regression import LinearRegression  # Créer un modèle sur les valeurs retardées  time\_lr = LinearRegression(featuresCol="features", labelCol="Close")  time\_lr\_model = time\_lr.fit(train\_data)  # Prédictions  time\_lr\_predictions = time\_lr\_model.transform(test\_data) |

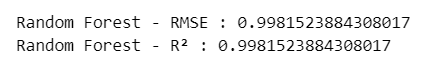
### **5.5 Évaluation des Modèles**

#### **Objectif :**

Comparer les performances des modèles en utilisant les métriques suivantes :

* **RMSE** : Root Mean Square Error.
* **R²** : Coefficient de détermination.

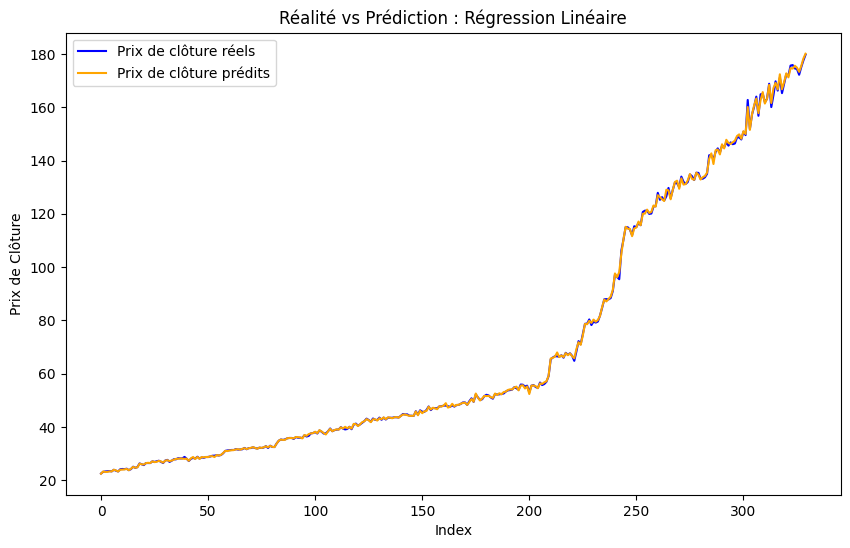


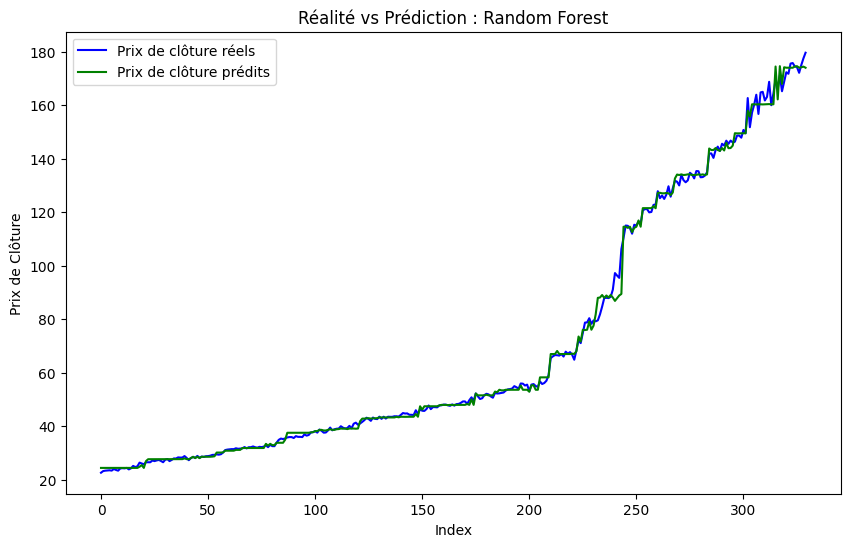


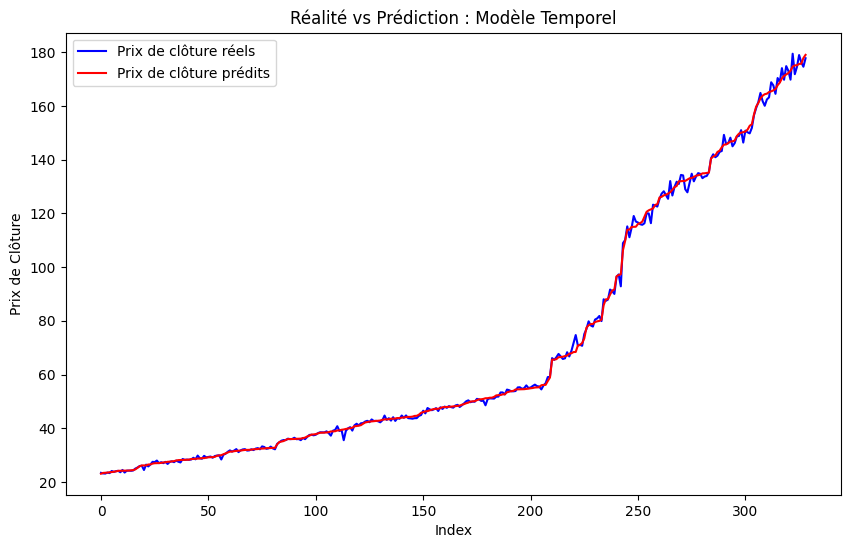


### **5.6 Visualisation des résultats**

On Visualise les plots Acutal vs Predicted pour chaque modele :







### **5.7 Amelioration du modele de la regression :**

Nous avons appliqué la régularisation Lasso ou Ridge à la régression linéaire en ajustant les hyperparamètres regParam (régularisation) et elasticNetParam (L1 vs L2).

### 

|  |
| --- |
| from pyspark.ml.tuning import ParamGridBuilder, CrossValidator  from pyspark.ml.evaluation import RegressionEvaluator  # Définir l'évaluateur pour le RMSE  evaluator = RegressionEvaluator(labelCol="Close", predictionCol="prediction", metricName="rmse")  # Configuration de la grille de paramètres  param\_grid = ParamGridBuilder() \  .addGrid(lr.regParam, [0.01, 0.1, 0.5]) \  .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \  .build()  cv = CrossValidator(estimator=lr, estimatorParamMaps=param\_grid, evaluator=evaluator, numFolds=3) # Validation croisée  # Entraînement avec validation croisée  cv\_model = cv.fit(train\_data)  # Évaluation des performances sur les données de test  lr\_optimized\_predictions = cv\_model.bestModel.transform(test\_data)  lr\_optimized\_rmse = evaluator.evaluate(lr\_optimized\_predictions)  lr\_optimized\_r2 = RegressionEvaluator(labelCol="Close", predictionCol="prediction", metricName="r2").evaluate(lr\_optimized\_predictions)  print(f"Régression Linéaire Optimisée - RMSE : {lr\_optimized\_rmse}")  print(f"Régression Linéaire Optimisée - R² : {lr\_optimized\_r2}") |

On Obtient les résultats pour le premier modèle :



### **Conclusion**

Cette section détaille le pipeline complet de traitement des données, de la préparation à l’évaluation des modèles. Chaque étape est justifiée et optimisée pour garantir des prédictions robustes sur les prix de clôture des actions. 🚀

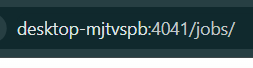
**6. Suivi des Jobs avec le GUI de Spark:**

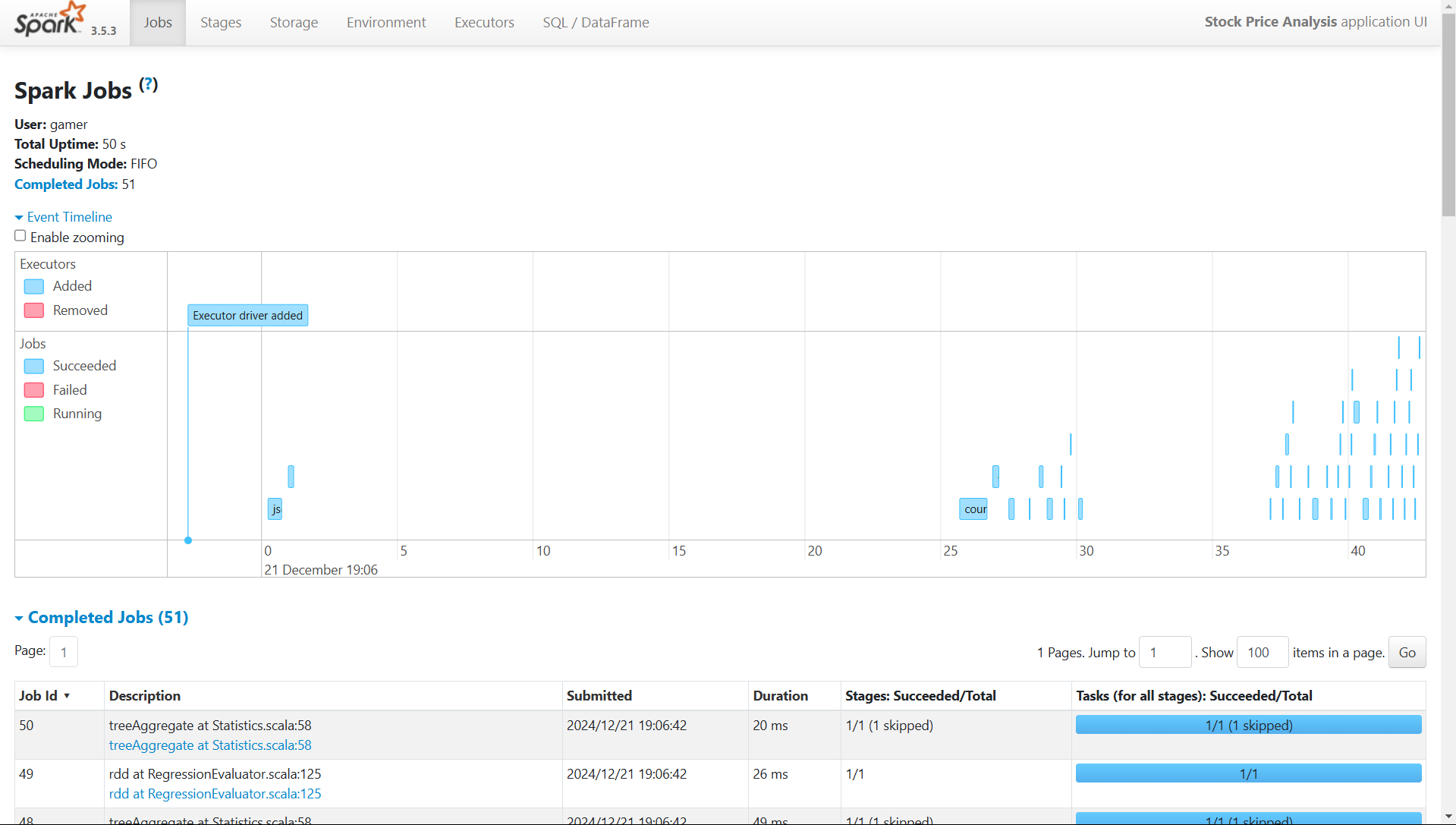
Apache Spark propose une interface utilisateur graphique (GUI) intégrée qui permet de suivre et de superviser les tâches exécutées sur le cluster Spark ou en mode local. Cette interface est essentielle pour diagnostiquer les performances, identifier les goulets d’étranglement et s'assurer que les jobs s'exécutent comme prévu.

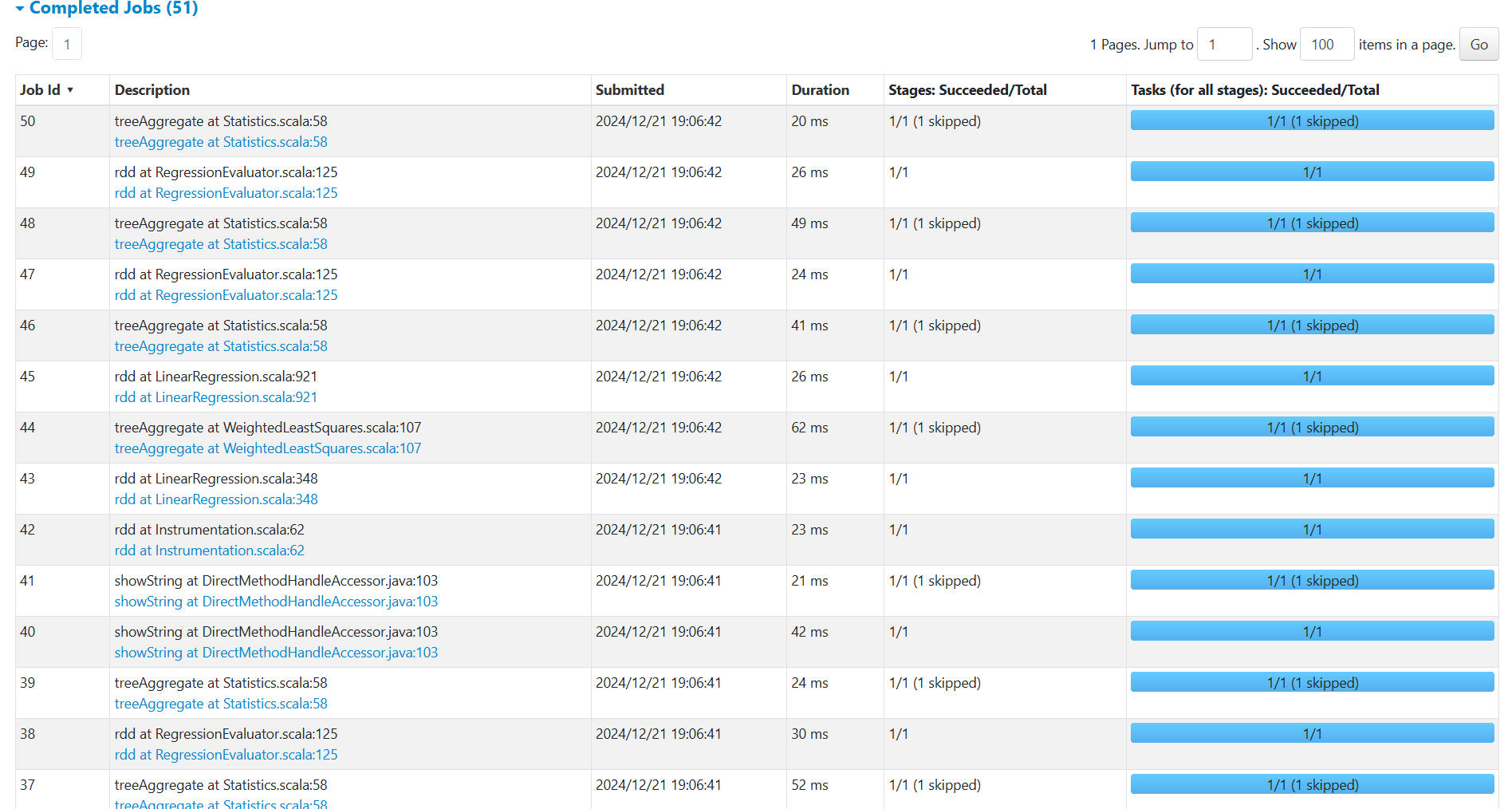
**Avantages du GUI Spark**

1. **Visualisation des Jobs** :
   * La GUI affiche les **jobs soumis**, leur état (succès, échec, en cours), leur durée, et les étapes qu'ils contiennent.
2. **Statistiques de Performance** :
   * Vous pouvez voir combien de tâches (tasks) ont été exécutées pour chaque job, ainsi que leur durée.
3. **Diagnostic des Échecs** :
   * En cas d’échec, le GUI fournit des informations détaillées sur la cause du problème, y compris les traces d'erreur.
4. **Suivi des Ressources** :
   * Il permet de superviser les **exécuteurs** (executors) utilisés, le stockage et les tâches parallèles en cours.

Ce Spark GUI est accede par le port 4041 de mon localhost :

****

****

****

**7. Conclusion**

Ce projet a démontré comment les outils modernes, comme Apache Spark, peuvent être utilisés pour résoudre des problèmes complexes de prédiction dans un environnement distribué. En combinant des techniques de prétraitement rigoureuses et des algorithmes de machine learning robustes, nous avons obtenu des résultats pertinents pour l’analyse des tendances boursières.

Le modèle final et le pipeline mis en place constituent une base solide pour des projets futurs, ouvrant la voie à une automatisation plus poussée et à des analyses financières plus sophistiquées. 🚀

**8. Reference**

#### **1. Documentation Officielle**

1. **Apache Spark** :
   * [Apache Spark Documentation](https://spark.apache.org/docs/latest/) : Documentation officielle d’Apache Spark, incluant MLlib, SparkSQL et les API.
2. **PySpark** :
   * [PySpark API Documentation](https://spark.apache.org/docs/latest/api/python/) : Documentation complète des fonctionnalités PySpark.
3. **Statsmodels** :
   * Statsmodels Documentation : Documentation officielle de la bibliothèque utilisée pour ARIMA.
4. **SparkSQL :**

* **SparkSQL documentation :** https://github.com/itversity/spark-sql

#### **2. Sources pour le Dataset**

1. **Hugging Face Dataset** :
   * Dataset utilisé dans le projet : ***Ammok/apple\_stock\_price\_from\_1980-2021***  
     ***https://huggingface.co/datasets/Ammok/apple\_stock\_price\_from\_1980-2021?viewer\_api=true***

#### **3. Livres et Guides Techniques**

1. **Livres sur Spark** :
   * "Learning Spark: Lightning-Fast Big Data Analysis" (Jules S. Damji, et al.) : Guide essentiel pour maîtriser Spark, y compris MLlib.

#### **4. Articles Académiques et Tutoriels**

1. **ARIMA et Modélisation des Séries Temporelles** :
   * Hyndman, R. J., & Athanasopoulos, G. : "Forecasting: Principles and Practice" (ouvrage librement accessible en ligne). Lien vers le livre.
2. **Optimisation des Modèles Machine Learning** :
   * Articles et blogs techniques sur l’utilisation de CrossValidator pour la validation croisée dans PySpark.

#### **5. Bibliothèques Python**

1. **Pandas** :
   * Pandas Documentation : Documentation pour la manipulation des DataFrames.
2. **Matplotlib** :
   * Matplotlib Documentation : Documentation pour les visualisations graphiques.

#### **6. Video d’installation de SPARK sur Youtube :**

1. **Install Apache PySpark on Windows PC | Apache Spark Installation Guide :**

* https://www.youtube.com/watch?v=OmcSTQVkrvo&t=91s