Spark Streaming

Vélib ile de france

DOUNIA HARAG
CHAIMAE MOUH
AYOUB ERRADI
CHRISTIAN NZEGAING

Sommaire

Contexte :

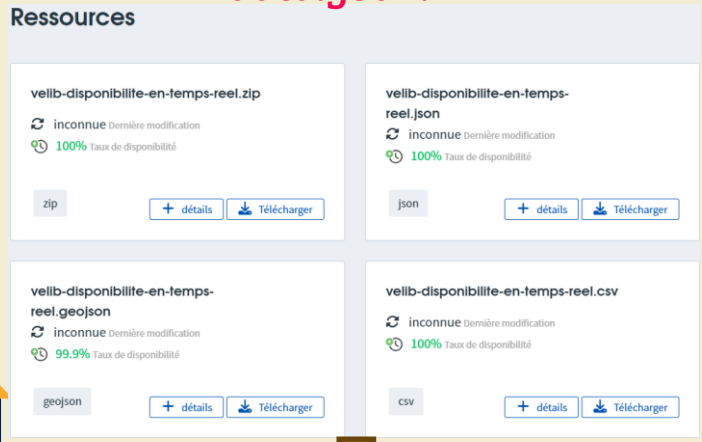
- Automatisation du traitement des données de disponibilité des vélos Velib en île de France avec Apache Spark.
- Calcul des indicateurs de performance clés (KPI) pour chaque station.
- Consolidation des résultats pour une analyse approfondie.
- Visualisation des données sur Power BI

DATASET

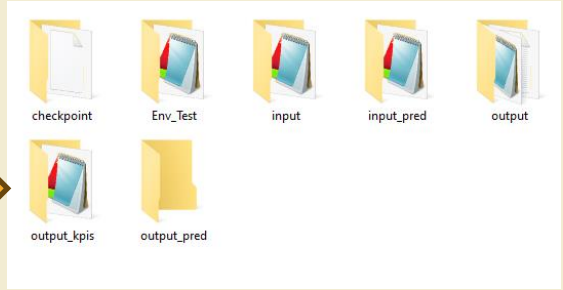
```
Root
 ├── stationcode : string
 ├── name : string
 ├── is_installed : string
 ├── capacity : Int
 ├── numdocksavailable : Int
 ├── numbikesavailable : Int
 ├── mechanical : Int
 ├── ebike : Int
 ├── is_renting : string
 ├── is_returning : string
 ├── duedate : string
 ├── coordonnees_geo_long : string
 ├── coordonnees_geo_lat : string
 ├── nom_arrondissement_communes : string
 └── code_insee_commune : string
```

Traitement de donnés

1. Origine de dataset : data.gouv.fr

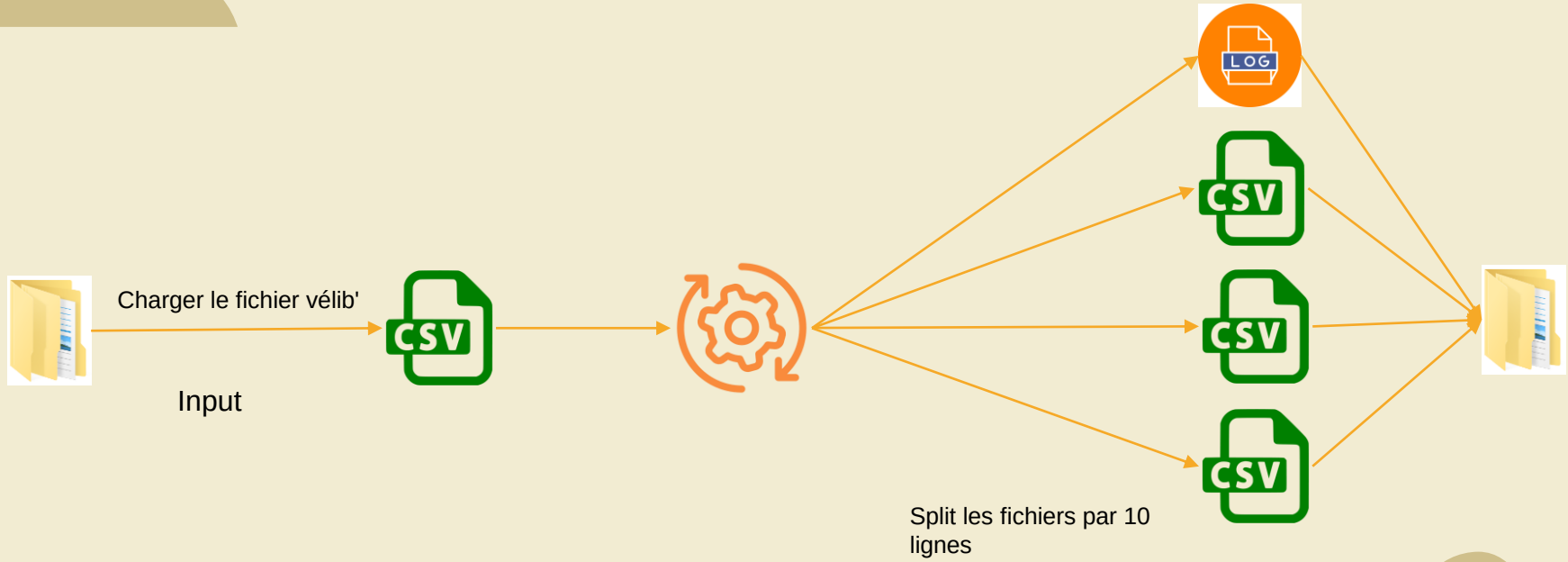


Stocker le fichier Velib-disponibilite-en-temps-réel.csv sur le répertoire input



2. Repertoire de données

Lire .Csv


Charger le fichier vélib'
Input
CSV
LOG
CSV
CSV
CSV
Split les fichiers par 10 lignes

Lire .Csv

```scala
import ...

object LireCSVVelib {
  def main(args: Array[String]): Unit = {
    // Créer une session Spark
    val spark = SparkSession.builder
      .appName("LireCSVVelib")
      .master("local[*]") // Exécuter localement avec toutes les CPU disponibles
      .getOrCreate()

    // Lire le fichier CSV
    val filePath = "C:\\Users\\DELL\\Documents\\input\\velib-disponibilite-en-temps-reel.csv"
    val df = spark.read
      .option("header", "true")
      .option("inferSchema", "true")
      .option("delimiter", ";") Choose schema // Spécifier le délimiteur correct
      .csv(filePath)

    // Définir le chemin du répertoire de sortie
    val outputDir = "C:\\Users\\DELL\\Documents\\output"
    val logFile = s"$outputDir\\log.txt"

    val totalRows = df.count().toInt
    val batchSize = 10

    // Initialiser le fichier de log
```

Application

```
// Lire tous les fichiers du répertoire de sortie
val inputDir = "C:\\Users\\DELL\\Documents\\output"
val files = new File(inputDir).listFiles.filter(_.getName.endsWith(".csv")).sorted

val kpiDir = "C:\\Users\\DELL\\Documents\\output_kpis"
new File(kpiDir).mkdirs()

val outputFilePath = s"$kpiDir/resultat_kpis.csv"

// Initialiser le fichier de sortie avec l'en-tête correct
val header = "stationcode;name;nom_arrondissement_communes;coordonnees_geo;duedate;total_bikes_available;avg_oc
Files.write(Paths.get(outputFilePath), (header + "\n").getBytes, StandardOpenOption.CREATE, StandardOpenOption.

// Boucle pour traiter les fichiers périodiquement
var nextRunTime = LocalDateTime.now()
files.foreach { file =>
  var currentTime = LocalDateTime.now()
  while (currentTime.isBefore(nextRunTime)) {
    // Boucle d'attente active jusqu'à ce que l'heure prévue pour le prochain traitement soit atteinte
    currentTime = LocalDateTime.now()
  }

  val kpiDF = processFile(file.getAbsolutePath, spark)
```

KPIS

Disponibilité des Bornes et Vélos

- Taux d'occupation des stations : (numbikesavailable/capacity)×100
- Taux de disponibilité des bornes : (numdocksavailable/capacity)×100
- Nombre de vélos disponibles

Répartition des Types de Vélos

- Pourcentage de vélos mécaniques : mechanical/numbikesavailable×100
- Pourcentage de vélos électriques : ebike/numbikesavailable×100

Performance des Stations

- Stations Pleines
- Stations Vides

Application – Fichier Final CSV

```
// Lire tous les fichiers du répertoire de sortie
val inputDir = "C:\\Users\\chaimae\\Projet_Spark_streaming\\src\\output"
val files = new File(inputDir).listFiles.filter(_.getName.endsWith(".csv")).sorted

val kpiDir = "C:\\Users\\chaimae\\Projet_Spark_streaming\\src\\output_kpi"
new File(kpiDir).mkdirs()

val outputFilePath = s"$kpiDir/resultat_kpis.csv"

// Initialiser le fichier de sortie avec l'en-tête correct
val header = "stationcode;name;nom_arrondissement_communes;coordonnees_geo;duedate;total_bikes_available;avg_oc
Files.write(Paths.get(outputFilePath), (header + "\n").getBytes, StandardOpenOption.CREATE, StandardOpenOption.

// Boucle pour traiter les fichiers périodiquement
var nextRunTime = LocalDateTime.now()
files.foreach { file =>
  var currentTime = LocalDateTime.now()
  while (currentTime.isBefore(nextRunTime)) {
    // Boucle d'attente active jusqu'à ce que l'heure prévue pour le prochain traitement soit atteinte
    currentTime = LocalDateTime.now()
  }

  val kpiDF = processFile(file.getAbsolutePath, spark)

  val outputDirPath = s"$kpiDir/resultat_kpis_temp"
  kpiDF.coalesce(1).write.mode("overwrite").option("header", "true").option("delimiter", ";").csv(outputDirPath
```

- Lecture des fichiers
- Création du répertoire de sortie
- Initialisation du fichier de sortie
- Boucle de traitement périodique

visualisation

Analyse en Temps Réel des Stations Vélib Métropole : Disponibilité des Vélos et des Bornes

Vélos Disponibles 9782

Stations Pleines 36

Stations Vides 76

Filtres

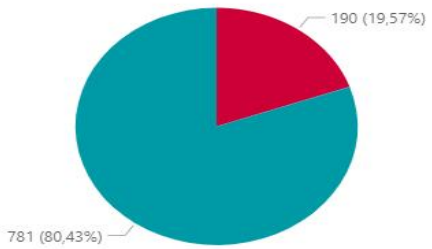
Arrondissement: Tout

Nom: Tout

Code_station: Tout

24/07/2024 15:47:54
Premier Date de Mise à Jour

Visualiser la répartition des types de vélos



Code_station	Nom	Vélos Disponibles	Arrondissement	Disponibilité des Bornes	Taux d'Occupation Moyen	Vélos Mécaniques	Vélos Électriques
15012	Cambronne - Mademoiselle	63	Paris	4.477611940298507	94.02985074626866	66.66666666666666	33.33333333333333
16025	Jean de la Fontaine - Boulainvilliers	63	Paris	2.941176470588235	92.64705882352942	85.71428571428571	14.285714285714285
13118	Regnault - Patay	61	Paris	1.5873015873015872	96.82539682539682	91.80327868852459	8.19672131147541
15059	Parc André Citroën	61	Paris	3.076923076923077	93.84615384615384	91.80327868852459	8.19672131147541
15021	Joseph Liouville - Croix Nivert	59	Paris	6.25	92.1875	79.66101694915254	20.33898305084746
15028	Grenelle - Dr Finlay	59	Paris	14.084507042253522	83.09859154929578	89.83050847457628	10.16949152542373
5015	Epée de Bois - Mouffetard	57	Paris	3.1746031746031744	90.47619047619048	84.21052631578947	15.789473684210526
15032	Félix Faure - Lourmel	57	Paris	0.0	98.27586206896551	96.49122807017544	3.508771929824561
11007	Jean Macé - Faidherbe	56	Paris	0.0	88.88888888888889	98.21428571428571	1.7857142857142856
15010	Square Cambronne	53	Paris	8.47457627118644	89.83050847457628	98.11320754716981	1.8867924528301887
17001	Guy Môquet - Saint-Ouen	49	Paris	7.547169811320755	92.45283018867924	65.3061224489796	34.69387755102041
21108	Mairie de Clichy	49	Clichy	1.9607843137254901	96.07843137254902	85.71428571428571	14.285714285714285
5031	Lacépède - Linné	46	Paris	1.8518518518518516	85.18518518518519	93.47826086956522	6.521739130434782
11024	Mairie du 11ème	46	Paris	3.9215686274509802	90.19607843137256	93.47826086956522	6.521739130434782

Prédiction

```
// Générer des lignes supplémentaires pour chaque station avec des disponibilités variables
val availabilityVariations = Seq(
  ("numdocksavailable", 5),
  ("numdocksavailable", -5),
  ("numbikesavailable", 5),
  ("numbikesavailable", -5),
  ("mechanical", 2),
  ("mechanical", -2),
  ("ebike", 2),
  ("ebike", -2)
)
```

Prédiction

```scala
// Appliquer les variations de disponibilité
val dfWithVariations = availabilityVariations.foldLeft(dfProcessed) { (accDf, variation) =>
  val (column, value) = variation
  accDf.union(
    dfProcessed.withColumn(column, col = $"$column" + lit(value))
  )
}

// Définir les variations temporelles
val timeVariations = Seq(
  1, 2, 3 // Variations temporelles en heures
)

// Appliquer les variations temporelles
val dfWithTimeVariations = timeVariations.foldLeft(dfWithVariations) { (accDf, hours) =>
  accDf.union(
    dfWithVariations.withColumn( colName = "duedate", col = $"duedate" + expr( expr = s"INTERVAL $hours HOUR"))
  )
}
```

Prédiction

```scala
// Appliquer les variations de disponibilité
val dfWithVariations = availabilityVariations.foldLeft(dfProcessed) { (accDf, variation) =>
  val (column, value) = variation
  accDf.union(
    dfProcessed.withColumn(column, col = $"$column" + lit(value))
  )
}

// Définir les variations temporelles
val timeVariations = Seq(
  1, 2, 3 // Variations temporelles en heures
)

// Appliquer les variations temporelles
val dfWithTimeVariations = timeVariations.foldLeft(dfWithVariations) { (accDf, hours) =>
  accDf.union(
    dfWithVariations.withColumn( colName = "duedate", col = $"duedate" + expr( expr = s"INTERVAL $hours HOUR"))
  )
}
```

Prédiction

```
// Définir le modèle RandomForestRegressor pour la régression multi-cible
val rfMechanical = new RandomForestRegressor()
  .setLabelCol("mechanical")
  .setFeaturesCol("features")
  .setPredictionCol("pred_mechanical")

val rfEbike = new RandomForestRegressor()
  .setLabelCol("ebike")
  .setFeaturesCol("features")
  .setPredictionCol("pred_ebike")

// Créer des pipelines pour chaque modèle
val pipelineMechanical = new Pipeline().setStages(Array(assembler, rfMechanical))
val pipelineEbike = new Pipeline().setStages(Array(assembler, rfEbike))

// Diviser les données en ensembles d'entraînement et de test
val Array(trainingData, testData) = dfWithFeatures.randomSplit(Array(0.7, 0.3))
```

Prédiction

```
// Bornage des prédictions entre 0 et la capacité de la station
val predictionsBounded = predictions
  .withColumn( colName = "pred_mechanical", when( condition = $"pred_mechanical" < 0,  value = 0)
    .when( condition = $"pred_mechanical" > $"capacity",  value = $"capacity")
    .otherwise( value = $"pred_mechanical"))
  .withColumn( colName = "pred_ebike", when( condition = $"pred_ebike" < 0,  value = 0)
    .when( condition = $"pred_ebike" > $"capacity",  value = $"capacity")
    .otherwise( value = $"pred_ebike"))
```

Merci !