

Project Big Data :

Yosser Sdiri 3DNI 2

- Le cluster est automatiquement arrêté après 60 minutes d'inactivité : donc Après la création d'une nouvelle ressource, n'oubliez pas d'installer le connecteur MongoDB Spark dans l'onglet library de cluster running : "org.mongodb.spark:mongo-spark-connector_2.12:3.0.1" de type Maven

1. Chargement des données MongoDB dans un DataFrame :






4

```
# Configurer l'URI MongoDB et les options
mongodb_uri = "mongodb+srv://yosser:projet123456@cluster0.hrkpg.mongodb.net/dbCovid"
database = "dbCovid" # Nom de la base de données MongoDB
collection = "covid" # Nom de la collection MongoDB

# Charger les données de MongoDB dans un DataFrame
dfCovid = spark.read.format("mongo") \
    .option("uri", mongodb_uri) \
    .option("database", database) \
    .option("collection", collection) \
    .load()
```

5

```
# Afficher les 5 premières lignes du dataframe dfCovid
display(dfCovid.limit(5))
```

Table					
	 _id	 aged_65_older	 aged_70_older	 cardiovasc_death_rate	 conti
1	> {"_id":"6725f9d50bf1b502a512e...	2.581	1.337	597.029	Asia
2	> {"_id":"6725f9d50bf1b502a512e...	2.581	1.337	597.029	Asia
3	> {"_id":"6725f9d50bf1b502a512e...	2.581	1.337	597.029	Asia
4	> {"_id":"6725f9d50bf1b502a512e...	2.581	1.337	597.029	Asia

5 rows

2. Ingestion et analyse exploratoire et descriptive des données :

2.1 Pré-traitement des données :

2.1.1 Conversion de la colonne date de type String en type Date :

Dans cette étape, on a converti la colonne date, qui est initialement de type "String" (avec des dates au format dd/MM/yyyy), en un type de données "Date" standard

8

```

from pyspark.sql.functions import udf
from pyspark.sql.types import DateType
from datetime import datetime

# Fonction pour transformer une chaîne en date
def string_to_date(date_str):
    return datetime.strptime(date_str, '%d/%m/%Y').date() # format 'jour/mois/année'

# Enregistrement de la fonction UDF
date_udf = udf(string_to_date, DateType())

# Application de la fonction UDF à la colonne 'date'
dfCovid = dfCovid.withColumn('date', date_udf(dfCovid['date']))

```

2.1.2 Conversion de la colonne "new_cases_smoothed_per_million" de type String en type float :

10

```

# Conversion de type de colonne 'new_cases_smoothed_per_million' de type string en type float
dfCovid = dfCovid.withColumn('new_cases_smoothed_per_million', dfCovid['new_cases_smoothed_per_million'].cast('float'))

```

2.2 Transformation des données :

2.2.1 Sélection des colonnes 'date', 'iso_code' et 'new_cases_smoothed_per_million' + Limitation à 50 000 lignes :

13

```

# Enregistrer le DataFrame dfCovid comme une vue temporaire pour utiliser SQL
dfCovid.createOrReplaceTempView("covid_data")

# Effectuer la projection pour sélectionner les colonnes 'date', 'iso_code' et 'new_cases_smoothed_per_million'
projection = spark.sql("""
    SELECT date, iso_code, new_cases_smoothed_per_million
    FROM covid_data
""").limit(50000) # Limiter à 50 000 lignes

# Affichage de la projection
projection.show()

```

2020-02-26	AFG	null
2020-02-27	AFG	null
2020-02-28	AFG	null
2020-03-02	AFG	0.0
2020-03-01	AFG	0.017
2020-03-04	AFG	0.0
2020-02-29	AFG	0.017
2020-03-06	AFG	0.0
2020-03-08	AFG	0.01
2020-03-10	AFG	0.01
2020-03-09	AFG	0.01
2020-03-03	AFG	0.0
2020-03-11	AFG	0.021
2020-03-12	AFG	0.021
2020-03-07	AFG	0.01
2020-03-14	AFG	0.021
2020-03-13	AFG	0.021
2020-03-15	AFG	0.042

+-----+-----+
only showing top 20 rows

2.2.2 Rename de la colonne 'new_cases_smoothed_per_million' en 'cases' + rendre le DataFrame persistant en mémoire vive :

15

```
# Renommer la colonne 'new_cases_smoothed_per_million' en 'cases'
renamed = projection.withColumnRenamed('new_cases_smoothed_per_million', 'cases')

# Mise en cache du DataFrame après renommage
renamed.cache()

# Affichage du DataFrame après renommage
renamed.show()
```

2020-02-26	AFG	null
2020-02-27	AFG	null
2020-02-28	AFG	null
2020-03-02	AFG	0.0
2020-03-01	AFG	0.017
2020-03-04	AFG	0.0
2020-02-29	AFG	0.017
2020-03-06	AFG	0.0
2020-03-08	AFG	0.01
2020-03-10	AFG	0.01
2020-03-09	AFG	0.01
2020-03-03	AFG	0.0
2020-03-11	AFG	0.021
2020-03-12	AFG	0.021
2020-03-07	AFG	0.01
2020-03-14	AFG	0.021
2020-03-13	AFG	0.021
2020-03-15	AFG	0.042

+-----+-----+-----+
only showing top 20 rows

2.2.3 Suppression des lignes contenant des valeurs manquantes + rendre le DataFrame persistant en mémoire vive :

17

```
# Supprimer les lignes contenant des valeurs manquantes
cleaned = renamed.dropna()

# Mise en cache du DataFrame après nettoyage et limitation
cleaned.cache()

# Affichage du DataFrame après suppression des valeurs manquantes
cleaned.show()
```

2020-03-04	AFG	0.0
2020-02-29	AFG	0.017
2020-03-06	AFG	0.0
2020-03-08	AFG	0.01
2020-03-10	AFG	0.01
2020-03-09	AFG	0.01
2020-03-03	AFG	0.0
2020-03-11	AFG	0.021
2020-03-12	AFG	0.021
2020-03-07	AFG	0.01
2020-03-14	AFG	0.021
2020-03-13	AFG	0.021
2020-03-15	AFG	0.042
2020-03-16	AFG	0.059
2020-03-05	AFG	0.0
2020-03-18	AFG	0.052
2020-03-17	AFG	0.063
2020-03-19	AFG	0.052

+-----+-----+-----+
only showing top 20 rows

2.2.4 Trier les données sur 'date' et 'iso_code' + rendre le DataFrame persistant en mémoire vive :

19

```
# Trier les données sur 'date' et 'iso_code'
sorted_data = cleaned.orderBy('date', 'iso_code')

# Mise en cache du DataFrame trié
sorted_data.cache()

# Affichage des données triées
sorted_data.show()
```

```
|2020-01-28|    CHN|0.496|
|2020-01-28|OWID_ASI|0.151|
|2020-01-29|    CAN|0.015|
|2020-01-29|    CHN|0.553|
|2020-01-29|OWID_ASI|0.169|
|2020-01-30|    CAN|0.007|
|2020-01-30|    CHN| 0.75|
|2020-01-30|OWID_ASI|0.229|
|2020-01-31|    AUS|0.049|
|2020-01-31|    CAN|0.004|
|2020-01-31|    CHN|0.888|
|2020-01-31|OWID_ASI|0.271|
|2020-02-01|    AUS|0.065|
|2020-02-01|    CAN|0.004|
|2020-02-01|    CHN|1.049|
|2020-02-01|    KHM|0.009|
|2020-02-01|OWID_ASI| 0.32|
|2020-02-02|    AUS|0.044|
+-----+-----+-----+
only showing top 20 rows
```

2.2.5 Vérification des types de données :

21

```
# Afficher les 5 premières lignes du dataframe sorted_data
display(sorted_data.limit(5))
```

Table 🔍 🏠 📄				
	📅 date	🌐 iso_code	1.2 cases	
1	2020-01-27	OWID_ASI	0.07100000232458115	
2	2020-01-28	CAN	0.0149999996647238...	
3	2020-01-28	CHN	0.4959999918937683	
4	2020-01-28	OWID_ASI	0.1509999930858612	
5	2020-01-29	CAN	0.0149999996647238...	
5 rows				

- Vérification des valeurs manquantes avec `isNull()` :

23

```
from pyspark.sql import functions as F

# Vérifier les valeurs manquantes dans chaque colonne du DataFrame
missing_values = sorted_data.select([F.count(F.when(F.col(c).isNull(), c)).alias(c) for c in sorted_data.columns])

# Affichage des résultats (nombre de valeurs manquantes par colonne)
missing_values.show()
```

```
+---+-----+-----+
|date|iso_code|cases|
```

```
+---+-----+-----+
|  0|         0|    0|
+---+-----+-----+
```

- Vérification du type de la colonne date après transformation :

25

```
# Vérification du type de la colonne 'date' après le prétraitement
sorted_data.select("date").printSchema()
```

```
root
|-- date: date (nullable = true)
```

2.2.6 Affichage des valeurs minimale, maximale et l'écart-type de la colonne 'cases' :

27

```
# Calculer la valeur minimale de la colonne 'cases'
min_cases = sorted_data.agg(F.min("cases")).collect()[0][0]

# Calculer la valeur maximale de la colonne 'cases'
max_cases = sorted_data.agg(F.max("cases")).collect()[0][0]

# Calculer l'écart-type de la colonne 'cases'
stddev_cases = sorted_data.agg(F.stddev("cases")).collect()[0][0]

# Affichage des résultats
print(f"Valeur minimale des 'cases' : {min_cases}")
print(f"Valeur maximale des 'cases' : {max_cases}")
print(f"Écart-type des 'cases' : {stddev_cases}")
```

```
Valeur minimale des 'cases' : 0.0
Valeur maximale des 'cases' : 9383.349609375
Écart-type des 'cases' : 517.7429020452261
```

2.3 Visualisation , Widget et dashboard :

- Graphique du nombre total de cas par pays
- Graphique du nombre total de cas par date

30

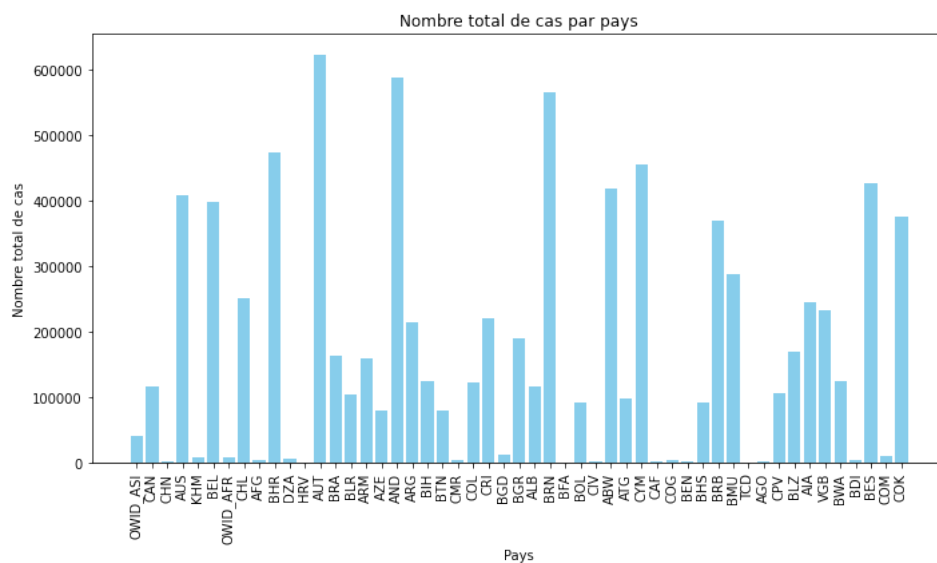
```
import matplotlib.pyplot as plt
import pandas as pd
from pyspark.sql import functions as F

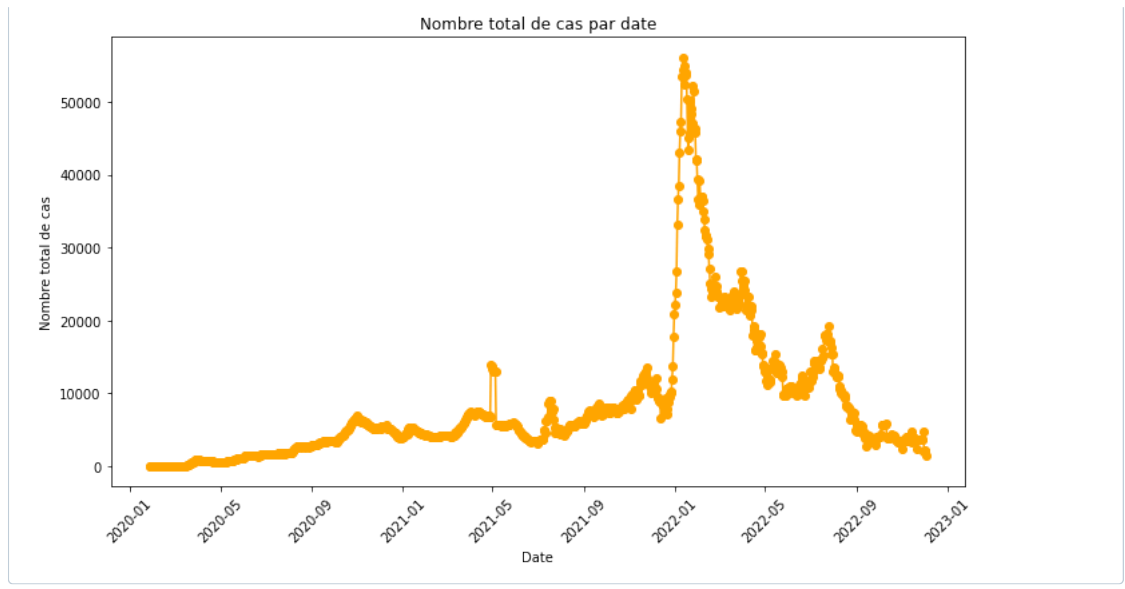
# Agrégation des données pour le nombre total de cas par pays
total_cases_by_country = sorted_data.groupBy("iso_code").agg(F.sum("cases").alias("total_cases")).toPandas()

# Agrégation des données pour le nombre total de cas par date
total_cases_by_date = sorted_data.groupBy("date").agg(F.sum("cases").alias("total_cases")).toPandas()

# 1. Graphique du nombre total de cas par pays
plt.figure(figsize=(10, 6))
plt.bar(total_cases_by_country['iso_code'], total_cases_by_country['total_cases'], color='skyblue')
plt.title('Nombre total de cas par pays')
plt.xlabel('Pays')
plt.ylabel('Nombre total de cas')
plt.xticks(rotation=90) # Rotation des étiquettes
plt.tight_layout() # Ajuste la disposition pour éviter que des étiquettes ne se chevauchent
plt.show()

# 2. Graphique du nombre total de cas par date
plt.figure(figsize=(10, 6))
plt.plot(total_cases_by_date['date'], total_cases_by_date['total_cases'], marker='o', color='orange')
plt.title('Nombre total de cas par date')
plt.xlabel('Date')
plt.ylabel('Nombre total de cas')
plt.xticks(rotation=45) # Rotation des étiquettes
plt.tight_layout() # Ajuste la disposition pour éviter que des étiquettes ne se chevauchent
plt.show()
```





- Map qui presente du nombre total de cas basé sur l'iso_code :

32

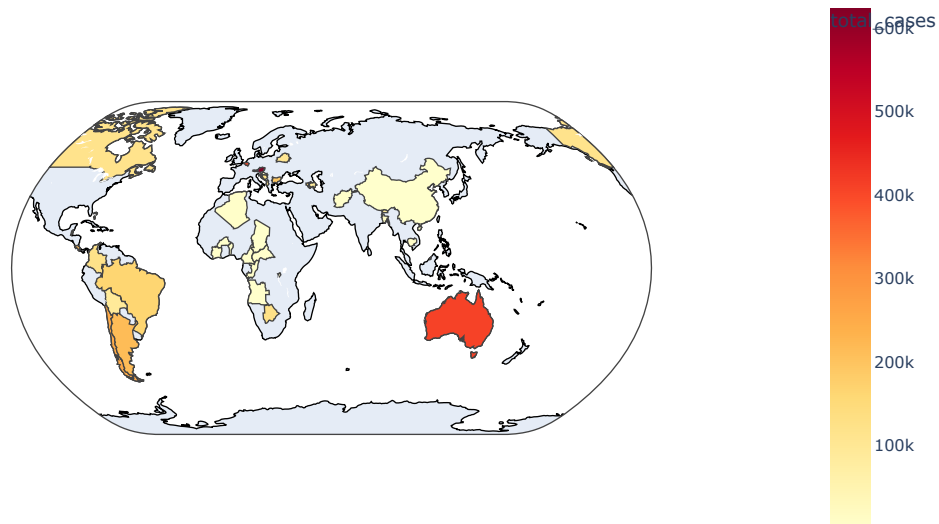
```
import plotly.express as px
import pandas as pd
from pyspark.sql import functions as F

# Agrégation des données pour le nombre total de cas par pays
total_cases_by_country = sorted_data.groupBy("iso_code").agg(F.sum("cases").alias("total_cases")).toPandas()

# Création de la carte choroplèthe
fig = px.choropleth(
    total_cases_by_country,
    locations='iso_code', # Code pays ISO
    color='total_cases', # Valeur à représenter par la couleur
    hover_name='iso_code', # Nom du pays au survol
    color_continuous_scale='YlOrRd', # Palette de couleurs
    title='Nombre total de cas par pays : '
)

# Affichage de la carte
fig.update_geos(showcoastlines=True, coastlinecolor="Black", projection_type="natural earth")
fig.update_layout(margin={"r":0,"t":40,"l":0,"b":0}) # Ajuster les marges
fig.show()
```

Nombre total de cas par pays :



Widget permettant de sélectionner l'iso_code d'un pays pour suivre l'évolution de la pandémie :

!!! il faut run la cellule pour voir la liste déroulante en haut :

34

```
import matplotlib.pyplot as plt
import pandas as pd
from pyspark.sql import functions as F

# Récupérer les pays uniques dans la colonne 'iso_code' du dataframe 'sorted_data'
unique_countries = sorted_data.select("iso_code").distinct().rdd.flatMap(lambda x: x).collect()

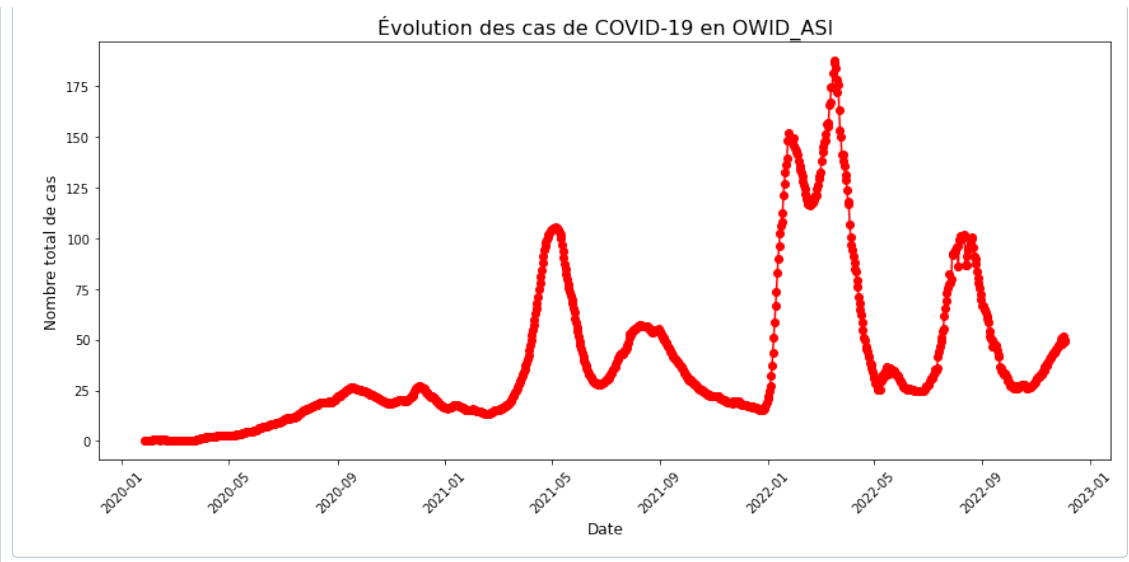
# Créer un widget avec la liste déroulante contenant les pays présents dans la colonne 'iso_code'
dbutils.widgets.dropdown("country_selector", unique_countries[0], unique_countries, "Sélectionner un pays")

# Récupérer le pays sélectionné dans le widget
selected_country = dbutils.widgets.get("country_selector")

# Filtrer les données en fonction du pays sélectionné
filtered_data = sorted_data.filter(sorted_data["iso_code"] == selected_country)

# Agréger les données pour obtenir le nombre total de cas par date pour le pays sélectionné
total_cases_by_date = filtered_data.groupBy("date").agg(F.sum("cases").alias("total_cases")).toPandas()

# --- Graphique : Nombre total de cas par date pour le pays sélectionné ---
plt.figure(figsize=(12, 6))
plt.plot(total_cases_by_date['date'], total_cases_by_date['total_cases'], marker='o', color='red')
plt.title(f'Évolution des cas de COVID-19 en {selected_country}', fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Nombre total de cas', fontsize=12)
plt.xticks(rotation=45) # Rotation des dates pour une meilleure lisibilité
plt.tight_layout()
plt.show()
```

Widget permettant de saisir la date et choisir l'iso_code de pays pour voir les cas du pays sélectionné :

!!! il faut run le cellule pour voir la liste déroulante en haut :

36

```
import pandas as pd
from pyspark.sql import functions as F

# Récupérer les pays uniques dans la colonne 'iso_code' du dataframe 'sorted_data'
unique_countries = sorted_data.select("iso_code").distinct().rdd.flatMap(lambda x: x).collect()

# Créer un widget avec la liste déroulante contenant les pays
dbutils.widgets.dropdown("country_selector", unique_countries[0], unique_countries, "Sélectionner un pays")

# Créer un widget de saisie de texte pour entrer une date
dbutils.widgets.text("date_selector", "2020-02-25", "Saisir une date (yyyy-mm-dd)")

# Récupérer la valeur sélectionnée dans le widget de pays
selected_country = dbutils.widgets.get("country_selector")

# Récupérer la date saisie dans le widget de texte
selected_date = dbutils.widgets.get("date_selector")

# Filtrer les données pour le pays et la date sélectionnés
filtered_data = sorted_data.filter(
    (sorted_data["iso_code"] == selected_country) &
    (sorted_data["date"] == selected_date)
)

# Si des données existent pour cette date et ce pays, afficher le tableau
if filtered_data.count() > 0:
    result_df = filtered_data.select("date", "iso_code", "cases").toPandas()
    display(result_df)
else:
    print("Aucune donnée trouvée pour la date et le pays sélectionnés.")
```

Table				Q 🔍 □	
	📅 date	🇦🇸 iso_code	1.2 cases		
1	2020-02-25	OWID_ASI	0.143999993801116...		

3. Analyse prédictive des données :

L'objectif de l'analyse prédictive est de développer un modèle de régression linéaire permettant de prédire le nombre de nouveaux cas après date + 7 jours, en fonction du pays et du nombre quotidien enregistré à date. Pour ce faire, on doit effectuer des transformations supplémentaires sur les données.

3.1 Transformation des données :

- on va créer un nouveau dataframe avec la colonne j7_cases qui contient le nombre de cas enregistrés 7 jours avant la date spécifiée pour chaque ligne.

41

```
from pyspark.sql import functions as F
from pyspark.sql.window import Window

# Définir la fenêtre partitionnée par 'iso_code' et triée par 'date'
window_spec = Window.partitionBy("iso_code").orderBy("date")

# Créer la colonne 'j7_cases' en décalant les valeurs de 'cases' de 7 jours
df_with_j7_cases = sorted_data.withColumn(
    "j7_cases",
    F.lag("cases", 7).over(window_spec) # Utilisation de lag pour obtenir les valeurs 7 jours avant
)

# Afficher les 20 premières lignes du dataframe df_with_j7_cases
display(df_with_j7_cases.limit(20))
```

Table					
	📅 date	🇦🇹 iso_code	1.2 cases	1.2 j7_cases	
1	2020-03-18	ABW	5.368000030517578	null	
2	2020-03-19	ABW	5.368000030517578	null	
3	2020-03-20	ABW	4.026000022888184	null	
4	2020-03-21	ABW	4.026000022888184	null	
5	2020-03-22	ABW	9.392999649047852	null	
6	2020-03-23	ABW	9.392999649047852	null	
7	2020-03-24	ABW	12.07699966430664	null	
8	2020-03-25	ABW	17.44499969482422	5.368000030517578	
9	2020-03-26	ABW	32.20600128173828	5.368000030517578	
10	2020-03-27	ABW	37.5730018615722...	4.026000022888184	
11	2020-03-28	ABW	55.0180015563964...	4.026000022888184	
12	2020-03-29	ABW	55.0180015563964...	9.392999649047852	
13	2020-03-30	ABW	55.0180015563964...	9.392999649047852	
14	2020-03-31	ABW	57.70199966430664	12.07699966430664	
15	2020-04-01	ABW	50.9920005798339...	17.44499969482422	

20 rows

- on va Transformer la variable catégorielle iso_code en une variable numérique, puis on va créer un vecteur de caractéristiques contenant les différentes variables explicatives (la variable cible étant cases).

43

```

from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml import Pipeline
from pyspark.sql.functions import col

# 1. Vérifier les types des colonnes dans le dataframe avant d'appliquer VectorAssembler
df_with_j7_cases.printSchema()

# Vérifier que j7_cases est de type Double
df_with_j7_cases = df_with_j7_cases.withColumn("j7_cases", col("j7_cases").cast("double"))

# 3. Gérer les valeurs manquantes : Suppression des lignes contenant des valeurs manquantes dans les colonnes nécessaires
df_with_j7_cases = df_with_j7_cases.na.drop(subset=["iso_code", "j7_cases"])

# 4. Transformation de 'iso_code' en une variable numérique avec StringIndexer
indexer = StringIndexer(inputCol="iso_code", outputCol="iso_code_indexed")

# 5. Création du vecteur de caractéristiques avec VectorAssembler
assembler = VectorAssembler(
    inputCols=["iso_code_indexed", "j7_cases"], # Inclure les colonnes nécessaires
    outputCol="features"
)

# 6. Création du pipeline
pipeline = Pipeline(stages=[indexer, assembler])

# 7. Appliquer le pipeline
df_transformed = pipeline.fit(df_with_j7_cases).transform(df_with_j7_cases)

# Affichage des 20 premières lignes du dataframe df_transformed
display(df_transformed.limit(20))

```

```

root
|-- date: date (nullable = true)
|-- iso_code: string (nullable = true)
|-- cases: float (nullable = true)
|-- j7_cases: float (nullable = true)

```

Table							
	📅 date	🔑 iso_code	1.2 cases	1.2 j7_cases	1.2 iso_code_indexed	🔗 features	
1	2020-03-25	ABW	17.44499969482422	5.368000030517578	29	➤ {"vectorType":"dens	
2	2020-03-26	ABW	32.20600128173828	5.368000030517578	29	➤ {"vectorType":"dens	
3	2020-03-27	ABW	37.5730018615722...	4.026000022888184	29	➤ {"vectorType":"dens	
4	2020-03-28	ABW	55.0180015563964...	4.026000022888184	29	➤ {"vectorType":"dens	
5	2020-03-29	ABW	55.0180015563964...	9.392999649047852	29	➤ {"vectorType":"dens	
6	2020-03-30	ABW	55.0180015563964...	9.392999649047852	29	➤ {"vectorType":"dens	
7	2020-03-31	ABW	57.70199966430664	12.07699966430664	29	➤ {"vectorType":"dens	
8	2020-04-01	ABW	50.9920005798339...	17.44499969482422	29	➤ {"vectorType":"dens	
9	2020-04-02	ABW	42.9410018920898...	32.20600128173828	29	➤ {"vectorType":"dens	
10	2020-04-03	ABW	38.9150009155273...	37.5730018615722...	29	➤ {"vectorType":"dens	
11	2020-04-04	ABW	24.15399932861328	55.0180015563964...	29	➤ {"vectorType":"dens	
12	2020-04-05	ABW	18.78700065612793	55.0180015563964...	29	➤ {"vectorType":"dens	
13	2020-04-06	ABW	28.18000030517578	55.0180015563964...	29	➤ {"vectorType":"dens	
14	2020-04-07	ABW	25.4960002899169...	57.70199966430664	29	➤ {"vectorType":"dens	▼

20 rows

Organiser les colonnes et déplacer la colonne "cases" vers la fin vue qu'elle est la variable cible :

```
# Récupérer les colonnes de df_transformed en excluant 'cases'
columns = [col for col in df_transformed.columns if col != 'cases']
# Ajouter 'cases' à la fin de la liste des colonnes
columns.append('cases')
# Créer un nouveau DataFrame avec l'ordre des colonnes modifié et le nommer "df_fin"
df_fin = df_transformed.select(columns)

# Afficher les 20 premières lignes du dataframe df_transformed
display(df_fin.limit(20))
```

	📅 date	🇦🇸 iso_code	1.2 j7_cases	1.2 iso_code_indexed	🔗 features
1	2020-03-25	ABW	5.368000030517578	29	> {"vectorType":"dense","length":2,"values":[29
2	2020-03-26	ABW	5.368000030517578	29	> {"vectorType":"dense","length":2,"values":[29
3	2020-03-27	ABW	4.026000022888184	29	> {"vectorType":"dense","length":2,"values":[29
4	2020-03-28	ABW	4.026000022888184	29	> {"vectorType":"dense","length":2,"values":[29
5	2020-03-29	ABW	9.392999649047852	29	> {"vectorType":"dense","length":2,"values":[29
6	2020-03-30	ABW	9.392999649047852	29	> {"vectorType":"dense","length":2,"values":[29
7	2020-03-31	ABW	12.07699966430664	29	> {"vectorType":"dense","length":2,"values":[29
8	2020-04-01	ABW	17.44499969482422	29	> {"vectorType":"dense","length":2,"values":[29
9	2020-04-02	ABW	32.20600128173828	29	> {"vectorType":"dense","length":2,"values":[29
10	2020-04-03	ABW	37.5730018615722...	29	> {"vectorType":"dense","length":2,"values":[29
11	2020-04-04	ABW	55.0180015563964...	29	> {"vectorType":"dense","length":2,"values":[29
12	2020-04-05	ABW	55.0180015563964...	29	> {"vectorType":"dense","length":2,"values":[29
13	2020-04-06	ABW	55.0180015563964...	29	> {"vectorType":"dense","length":2,"values":[29
14	2020-04-07	ABW	57.70199966430664	29	> {"vectorType":"dense","length":2,"values":[29

20 rows

3.2 Apprentissage et Test :

- Division des données d'apprentissage et de test tout en préservant l'ordre des données :

48

```
# 80 % pour l'apprentissage, 20 % pour le test
training_ratio = 0.8

# Calculer le nombre total de lignes et déterminer la limite pour l'ensemble d'apprentissage
total_rows = df_fin.count()
training_count = int(total_rows * training_ratio)

# Diviser le DataFrame en ensembles d'apprentissage et de test
df_train = df_fin.limit(training_count)
df_test = df_fin.subtract(df_train)

# Afficher un aperçu des deux ensembles
print("Ensemble d'apprentissage :")
display(df_train.limit(20))
print("Ensemble de test :")
display(df_test.limit(20))
```

Ensemble d'apprentissage :

	📅 date	🇦🇸 iso_code	1.2 j7_cases	1.2 iso_code_indexed	🔗 features
1	2020-03-25	ABW	5.368000030517578	29	> {"vectorType":"dense","length":2,"values":[29
2	2020-03-26	ABW	5.368000030517578	29	> {"vectorType":"dense","length":2,"values":[29
3	2020-03-27	ABW	4.026000022888184	29	> {"vectorType":"dense","length":2,"values":[29
4	2020-03-28	ABW	4.026000022888184	29	> {"vectorType":"dense","length":2,"values":[29
5	2020-03-29	ABW	9.392999649047852	29	> {"vectorType":"dense","length":2,"values":[29

6	2020-03-30	ABW	9.392999649047852	29	> {"vectorType":"dense","length":2,"values":[29
7	2020-03-31	ABW	12.07699966430664	29	> {"vectorType":"dense","length":2,"values":[29
8	2020-04-01	ABW	17.44499969482422	29	> {"vectorType":"dense","length":2,"values":[29
9	2020-04-02	ABW	32.20600128173828	29	> {"vectorType":"dense","length":2,"values":[29
10	2020-04-03	ABW	37.5730018615722...	29	> {"vectorType":"dense","length":2,"values":[29
11	2020-04-04	ABW	55.0180015563964...	29	> {"vectorType":"dense","length":2,"values":[29
12	2020-04-05	ABW	55.0180015563964...	29	> {"vectorType":"dense","length":2,"values":[29
13	2020-04-06	ABW	55.0180015563964...	29	> {"vectorType":"dense","length":2,"values":[29
14	2020-04-07	ABW	57.70199966430664	29	> {"vectorType":"dense","length":2,"values":[29

20 rows

Ensemble de test :

Table					
	📅 date	🇸🇨 iso_code	1.2 j7_cases	1.2 iso_code_indexed	🔗 features
1	2020-11-19	COM	2.90199995040893...	48	> {"vectorType":"dense","length":2,"values":[48
2	2020-11-20	COM	2.90199995040893...	48	> {"vectorType":"dense","length":2,"values":[48
3	2020-11-21	COM	2.73200011253356...	48	> {"vectorType":"dense","length":2,"values":[48
4	2020-11-22	COM	2.73200011253356...	48	> {"vectorType":"dense","length":2,"values":[48
5	2020-11-23	COM	2.73200011253356...	48	> {"vectorType":"dense","length":2,"values":[48
6	2020-11-24	COM	3.75600004196167	48	> {"vectorType":"dense","length":2,"values":[48
7	2020-11-25	COM	3.75600004196167	48	> {"vectorType":"dense","length":2,"values":[48
8	2020-11-26	COM	3.072999954223633	48	> {"vectorType":"dense","length":2,"values":[48
9	2020-11-27	COM	3.072999954223633	48	> {"vectorType":"dense","length":2,"values":[48
10	2020-11-28	COM	2.90199995040893...	48	> {"vectorType":"dense","length":2,"values":[48
11	2020-11-29	COM	2.90199995040893...	48	> {"vectorType":"dense","length":2,"values":[48
12	2020-11-30	COM	2.90199995040893...	48	> {"vectorType":"dense","length":2,"values":[48
13	2020-12-01	COM	2.73200011253356...	48	> {"vectorType":"dense","length":2,"values":[48
14	2020-12-02	COM	2.73200011253356...	48	> {"vectorType":"dense","length":2,"values":[48

20 rows

Modèle de régression linéaire pour la prédiction du nombre de nouveaux cas en fonction du pays et du nombre de cas enregistrés 7 jours auparavant:

50

```

from pyspark.ml.regression import LinearRegression

# Initialisation du modèle de régression linéaire
lr = LinearRegression(featuresCol="features", labelCol="cases")

# Entraînement du modèle sur les données d'apprentissage
lr_model = lr.fit(df_train)

# Prédiction sur les données de test
predictions = lr_model.transform(df_test)

# Affichage des prédictions
predictions.select("date", "iso_code", "j7_cases", "cases", "prediction").show(10)

```

date	iso_code	j7_cases	cases	prediction
2020-11-19	COM	2.9019999504089355	3.073	26.679841989759684
2020-11-20	COM	2.9019999504089355	3.073	26.679841989759684
2020-11-21	COM	2.7320001125335693	2.902	26.533724740707864
2020-11-22	COM	2.7320001125335693	2.902	26.533724740707864
2020-11-23	COM	2.7320001125335693	2.902	26.533724740707864
2020-11-24	COM	3.75600004196167	2.732	27.41386706664931
2020-11-25	COM	3.75600004196167	2.732	27.41386706664931
2020-11-26	COM	3.072999954223633	3.073	26.826818895488106

```
|2020-11-27|    COM| 3.072999954223633|3.073|26.826818895488106|
|2020-11-28|    COM|2.9019999504089355|2.561|26.679841989759684|
+-----+-----+-----+-----+
only showing top 10 rows
```

Évaluation des performances du modèle sur les données de test et Affichage de MAE :

52

```
from pyspark.ml.evaluation import RegressionEvaluator

# Initialisation de l'évaluateur avec la colonne de prédiction et la colonne cible
evaluator = RegressionEvaluator(labelCol="cases", predictionCol="prediction", metricName="mae")

# Calcul du MAE (Mean Absolute Error) sur les données de test
mae = evaluator.evaluate(predictions)

# Affichage du MAE
print("Mean Absolute Error (MAE) sur les données de test :", mae)
```

```
Mean Absolute Error (MAE) sur les données de test : 81.45090491339052
```

3.3 Tuning des hyperparamètres et visualisation

- Graphique en nuage de points (scatter plot) permettant de visualiser la différence entre les valeurs prédites par notre modèle et les valeurs réelles en utilisant Matplotlib

55

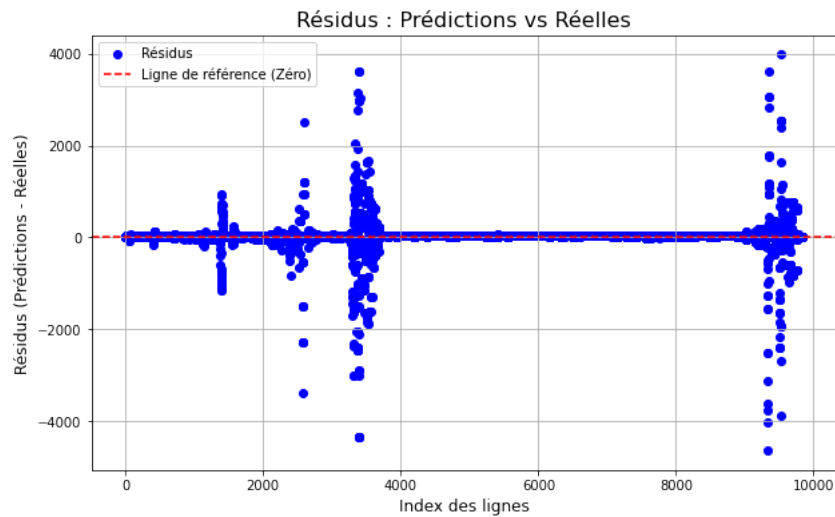
```
import matplotlib.pyplot as plt
import numpy as np

# Convertir le DataFrame des prédictions en Pandas DataFrame pour pouvoir l'utiliser avec Matplotlib
predictions_pd = predictions.select("cases", "prediction").toPandas()

# Convertir les colonnes 'cases' et 'prediction' en tableaux NumPy
y_true = predictions_pd['cases'].values # Valeurs réelles
y_pred = predictions_pd['prediction'].values # Valeurs prédites

# Calculer les résidus (différence entre les valeurs prédites et réelles)
residuals = y_pred - y_true

# Créer un nuage de points
plt.figure(figsize=(10, 6))
plt.scatter(predictions_pd.index, residuals, color='blue', label='Résidus')
plt.axhline(0, color='red', linestyle='--', label='Ligne de référence (Zéro)') # Ligne horizontale à 0 pour visualisation
plt.title('Résidus : Prédictions vs Réelles', fontsize=16)
plt.xlabel('Index des lignes', fontsize=12)
plt.ylabel('Résidus (Prédictions - Réelles)', fontsize=12)
plt.legend()
plt.grid(True)
plt.show()
```



3.4 Le sauvegarde de modèle :

Remarque :

- Installer MLflow : `pip install mlflow`
- Exécuter toutes les cellules au début, car après l'installation des bibliothèques, l'interpréteur sera redémarré et oubliera les exécutions précédentes.

58

```
#pip install mlflow
```

59

```
import mlflow
print(mlflow.__version__)
```

2.17.2

60

```
2024/11/10 17:28:40 INFO mlflow.spark: Inferring pip requirements by reloading the logged model from the databricks
artifact repository, which can be time-consuming. To speed up, explicitly specify the conda_env or pip_requirements
when calling log_model().
2024/11/10 17:28:59 WARNING mlflow.utils.environment: Encountered an unexpected error while inferring pip requiremen
ts (model URI: dbfs:/databricks/mlflow-tracking/3540076995625223/d72274a3cfc346c39ea3fa49f782d833/artifacts/linear_r
egression_model/sparkml, flavor: spark). Fall back to return ['pyspark==3.3.2', 'pandas<2']. Set logging level to DE
BUG to see the full traceback.
2024/11/10 17:28:59 INFO mlflow.tracking._tracking_service.client: 🐞 View run grandiose-cub-404 at: https://communi
ty.cloud.databricks.com/ml/experiments/3540076995625223/runs/d72274a3cfc346c39ea3fa49f782d833 (https://community.clo
ud.databricks.com/ml/experiments/3540076995625223/runs/d72274a3cfc346c39ea3fa49f782d833).
2024/11/10 17:28:59 INFO mlflow.tracking._tracking_service.client: 📄 View experiment at: https://community.cloud.d
atabricks.com/ml/experiments/3540076995625223 (https://community.cloud.databricks.com/ml/experiments/354007699562522
3).
```

