CLEANING DATA WITH SPARK

Spark Schema
- Es un esquema que indica el tipo de dato de cada columna del dataframe
- Ejemplo →

```
import pyspark.sql.types
schema = StructType([
    StructField('nombre', StringType(), True),
    ...
])
```

(nombre, tipo de dato, ¿Puede ser nulo?)

- Acelera la lectura de datos
- Se aplica el esquema a la lectura de datos

```
spark.df = spark.read.format('csv').load(nombre='...', schema=...)
```

Inmutability and lazy processing

Spark DataFrames son inmutables.

Entendiendo parquet

csv es texto de parsear

si no hay un esquema, debe leer todos los datos para crear uno } Problemas de CSV y Spark

Parquet
- Estos archivos incluyen automáticamente un schema.
- No necesita leer todos la información
- Como leerlo
  - leer parquet →
    ```
    df = spark.read.format('parquet').load('archivo.parquet')
    df = spark.read.parquet('archivo.parquet')
    ```
  - escribir archivo →
    ```
    df.write.format('parquet').save('nombre.parquet')
    df.write.parquet('nombre.parquet')
    ```
- Usos → Funcionan bien con queries SQL (mejora el rendimiento).

DataFrame column operations I

Para crear condiciones
- Filtrar valores NO NULOS → `df['col'].isNotNull()`
- " " NULOS → `df['col'].isNull()`

con ~ hacemos como con not

- Filtrar strings que contengan un texto → `df['nombre'].contains('Hola')`

Funciones de transformación de strings (`import pyspark.sql.functions as F`)
- Para pasar a mayúsculas → `F.upper(columna)`
- Para dividir texto → `F.split(columna, ',')`
- Para cambiar el tipo → `df['col'].cast(IntegerType())`
- Para interactuar con valores que sean de tipo ArrayType()
  - len → `.size(col)`
  - Retorna un elemento de la lista → `.getItem(idx)`

CLEANING DATA WITH SPARK

DataFrame column operations I

Operaciones condicionales:

- Condicional clauses →
  - `.when()` / `.otherwise()` → Puede haber varios when consecutivos.
  - `F.when(df.Age >= 18, 'Adult').otherwise('Minor')`

User defined functions

Ejemplo de como se define →

```
from pyspark.sql.function import udf
def ...(col):
    return ...

udf_func = udf(function, StringType())
```

Para usarla:

```
df = df.withColumn('col', udf_func(df.col))
```

Partitioning and lazy processing

Para crear IDS con Spark vendo indices preparados para procesamiento en // →

```
pyspark.sql.functions.monotonically_increasing_id()
```

Para ver el Nº de Particiones que tiene un dataframe → `df.rdd.getNumPartitions()`

Caching

Guarda los DataFrames en memoria o en disco
- → Reduce el uso de recursos
- → Mejora la velocidad.
- → Problemas → Si hay muchos datos igual no caben en memoria

Para implementarlo se cachea antes de una acción →

```
df = spark.read.csv('...')
df.cache().count()
```

se pueden hacer también separado.

Operaciones:
- Comprobar si esta cacheado → `df.is_cached`
- Para descachearlo → `df.unpersist()`

CLEANING DATA WITH SPARK

Improving import performance

data → DataFrame

Parámetros importantes → N° de objetos (Archivos, Network locations, etc)
→ Tamaño general de los objetos (Funciona mejor si son del mismo tamaño 128MB(ideal))

Esquemas bien definidos mejora el funcionamiento

Para leer varios archivos en 1 dataframe → `df = spark.read.csv('archivo_*.txt.gz')` (tipo CSV → GZIP)

Cluster configurations

Para leer la configuración del cluster → `spark.conf.get(<conf name>)`

" escribir " " " → `spark.conf.set(<conf name>)`

Opciones de deployment → Single node
→ Standalone
→ Managed (YARN, Kubernetes...) → Manejado por terceros

Responsabilidades del Driver del cluster → Asignación de tareas
→ Consolidación de resultados
→ Shared data access
→ Tips { - El drive node debe tener 2x memoria que worker node
- Fast local storage

" " Worker del cluster → Ejecuta las tareas

Para leer algunas configuraciones de spark → Nombre → `name = spark.conf.get('spark.app.name')`
→ Puerto del driver (TCP) → `driver_port = spark.conf.get('spark.driver.port')`
→ N° de particiones → `partitions = spark.conf.get('spark.sql.shuffle.partitions')`

Performance improvements

Para ver el plan que ha definido spark para realizar una tarea → `df.explain()`

Usar broadcasting (`.broadcast()`) da una copia de todos los datos a cada worker, esto mejora el rendimiento de `.join()`.

Para usarlo →
```
from pyspark.sql.functions import broadcast
df = df_b.join(broadcast(df_2))
```

CLEANING DATA WITH SPARK

Data Pipelines

Como es un data Pipeline
1. Input
2. Transformaciones
3. Output
4. Validation
5. Analysis

Los Pipelines no estan definidos como un objeto en Spark, simplemente es el proceso

Data handling techniques

Algunos posibles errores en los datos de entrada son
- Datos incorrectos → Nulos, Lineas corruptas, Headers
- Estructuras anidadas → Diferentes separadores
- Datos no regulares

Parseador de CSV de Spark
- Puede remover lineas vacias
- " " " comentarios → `df = spark.read.csv('....', comment='#')`
- Usar los headers → `df = spark.read.csv('....', header=True)`
- Definimos el separador → `df = spark.read.csv('....', sep=';')` → Si el separador no es un String, sep='#' y guarda los datos en una columna
Así podemos leer estructuras anidadas.

Data validation (comprobar que tiene un formato)

Validacion usando join
- Mucho mas rápido que recorrer todos los datos
- Ejemplo →
```
df1 = ...
df2 = ...
verificado = df1.join(df2, df1.att1 == df2.att1)
```
El tercer argumento es el tipo de join ('inner', 'outer'...)

Analysis final y delivery

Usar calculos para datos como medias, medianas etc.