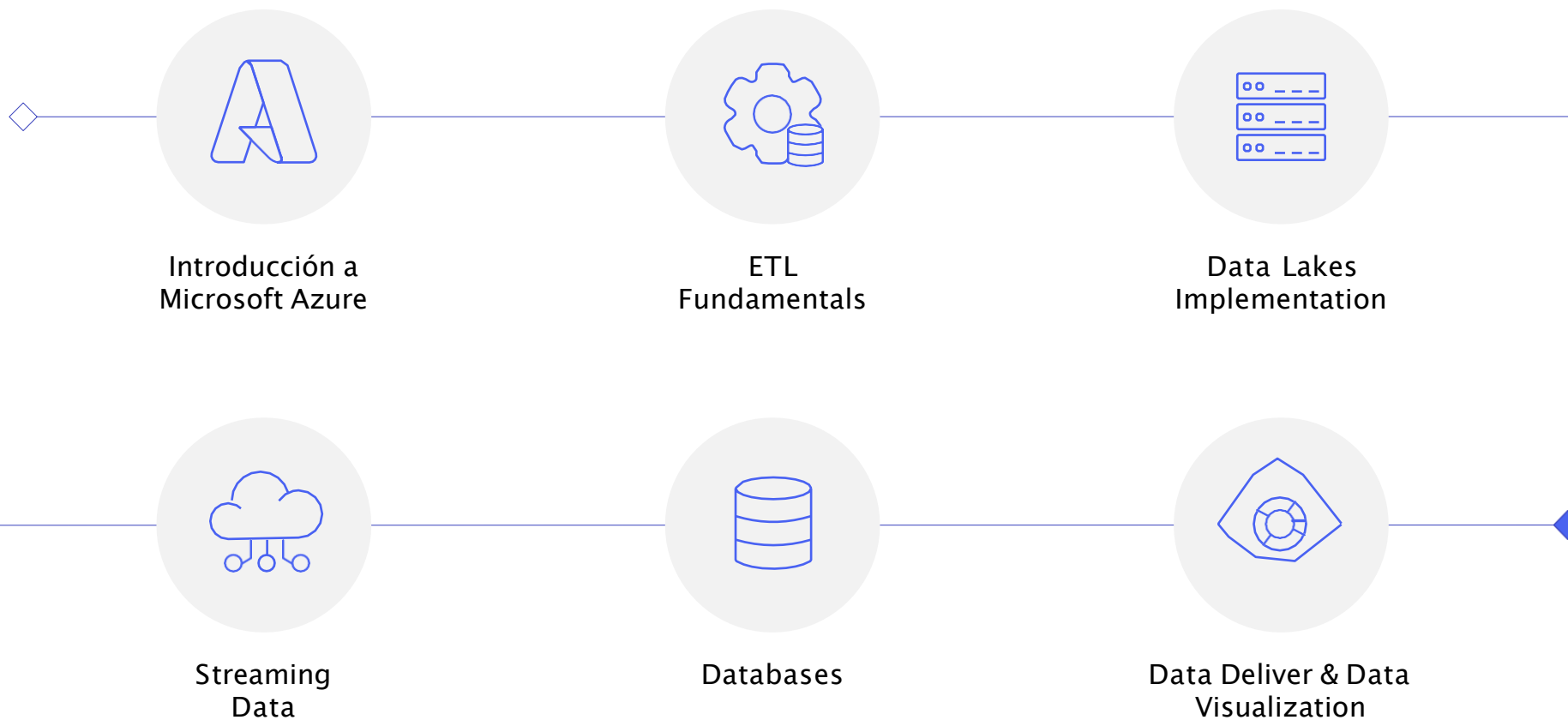


ESPECIALIZACIÓN **En Ingeniería de datos con Azure**

**CERIFICACIÓN FINAL**

por **Aprobación** de la Especialización en **Ingeniería de Datos con Microsoft Azure** (48 horas académicas)



Introducción a Microsoft Azure

- Introducción a Cloud Computing. Proveedores de servicios Cloud, On-Premise vs. On-Cloud, principales servicios, descripción de los modelos de costos.
- Identify and Access Management (IAM). Overview de los roles principales, ejemplos de gestión de permisos.



ETL Fundamentals

- Introducción a las soluciones ETL. Definición, descripción de sus etapas.
- Introducción a los servicios Azure Data Factory y Data Flow. Características generales, casos de uso.
- Taller: Implementación de un ETL Básico con Azure.



Data Lakes Implementation

- Introducción a Data Lakes. Definición, arquitectura, capas (Raw, Stage, Analytics).
- Introducción a los servicios Azure Blob Storage y Storage Account.
- Taller: Implementación de un Datalake en Azure.



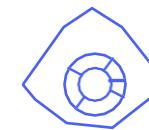
Streaming Data

- Introducción a procesamiento de datos Batch y Streaming. Diferencias Near-Real-Time y Real-Time.
- Introducción a IoT. Definición, uso de sensores, aplicaciones.
- Revisión de servicios: Azure EventHubs y IoT Hub. Características generales, ejemplos de implementación y uso.
- Taller: Manejo de Streaming al Data.



Databases

- Introducción a las bases de datos Relacionales y No-Relacionales. Definición, características, casos de uso.
- Azure SQL Database for MariaDB. Descripción y características generales.
- Azure SQL Database for PostgreSQL. Descripción y características generales.
- Azure SQL Database for CosmosDB. Descripción y características generales.
- Taller: Diseño de una base de datos relacional y técnicas para poblarla.

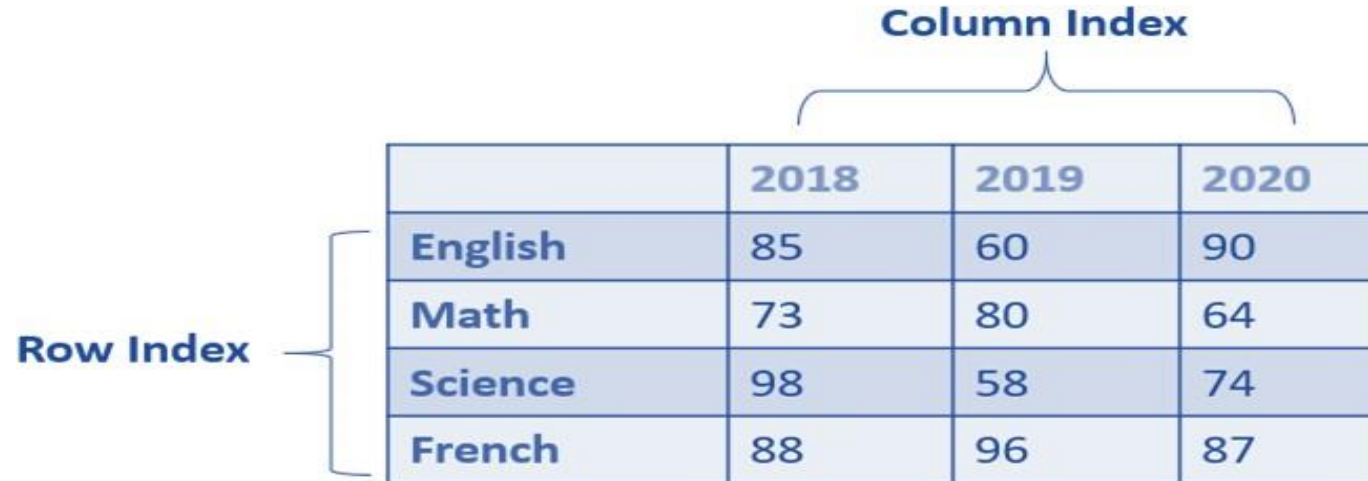


Data Deliver & Data Visualization

- Azure Synapse Analytics. Propósito del servicio, características generales.
- Fabric. Propósito del servicio, características generales.
- Taller: Conexión de Power BI a servicios de datos de Azure.

DataFrames en Apache Spark

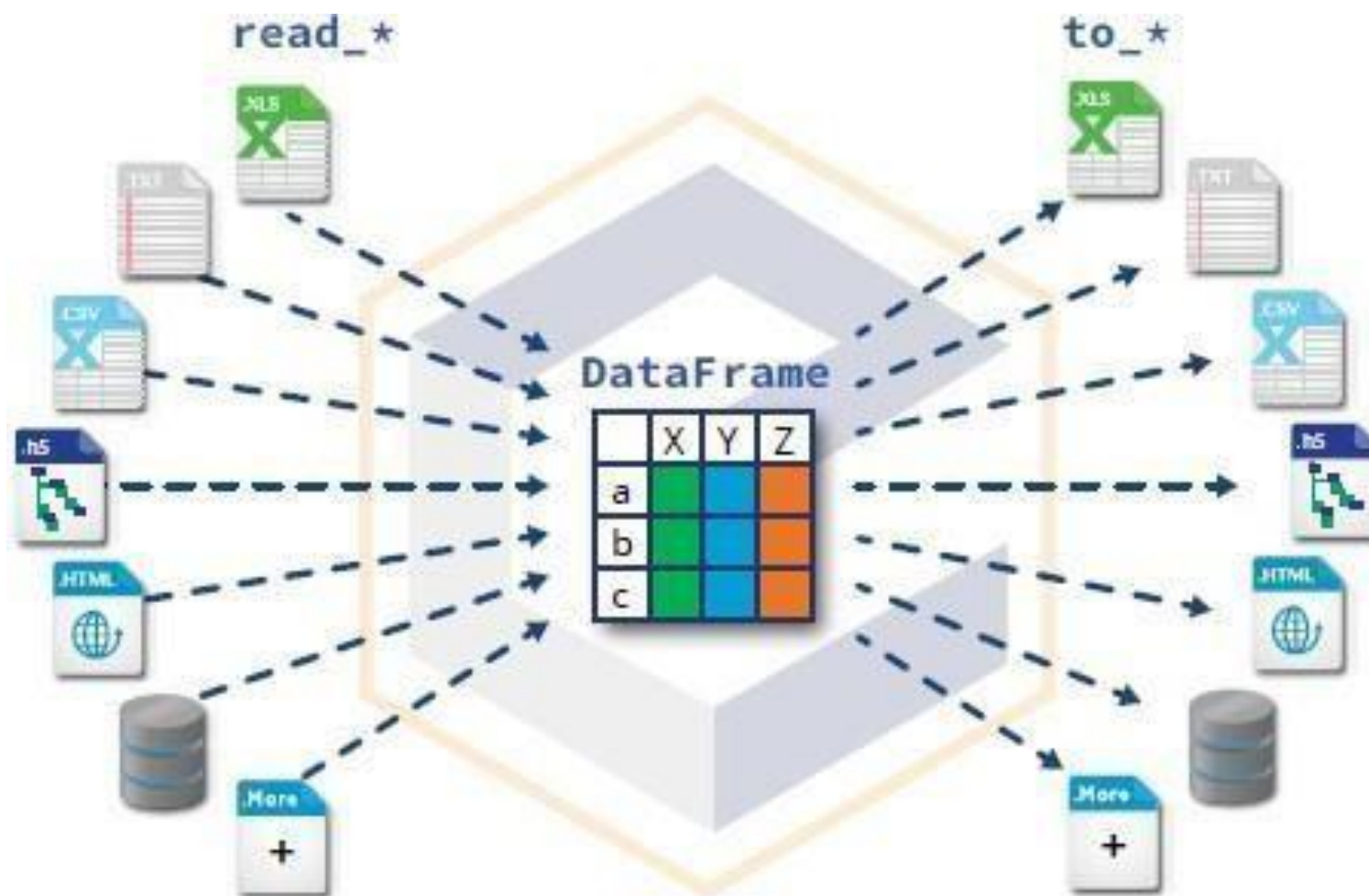
Los **DataFrames** son de naturaleza **tabular**. Permiten varios formatos dentro de una misma tabla (heterogéneos), mientras que cada variable suele tener valores con un único formato (homogéneos). Similares a las tablas SQL o a las hojas de cálculo.



The diagram illustrates a DataFrame as a table. A bracket on the left labeled "Row Index" points to the subject names in the first column. A bracket on top labeled "Column Index" points to the years in the first row. The table contains the following data:

	2018	2019	2020
English	85	60	90
Math	73	80	64
Science	98	58	74
French	88	96	87

Importar y exportar datos con pandas



Importar y exportar datos con pandas

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	
text	JSON	read_json	
text	HTML	read_html	
text	Local clipboard	read_clipboard	
binary	MS Excel		to_excel
binary	HDF5 Format		to_hdf
binary	Feather Format		to_feather
binary	Msgpack		to_msgpack
binary	Stata		to_stata
binary	SAS		to_sas
binary	Pickle		to_pickle
SQL			to_sql
SQL			read_sql

- ▶ Estas funciones son muy versátiles ya que cuentan con docenas de parámetros opcionales que permiten definir cómo se van a cargar los datos.

Importar y exportar datos con pandas

- ▶ Importar datos en formato csv.

```
d = pd.read_csv('Data/students.csv')
```

- ▶ Importar datos en formato xls.

```
d = pd.read_excel('Data/students.xls')
```

- Exportar datos de una data frame a xls y csv

In [56]:

```
# Exportar datos de un data frame a un xls  
d.to_excel('E:/JFB/Python/PYTHON_1/DATOS/sample_data.xls')
```

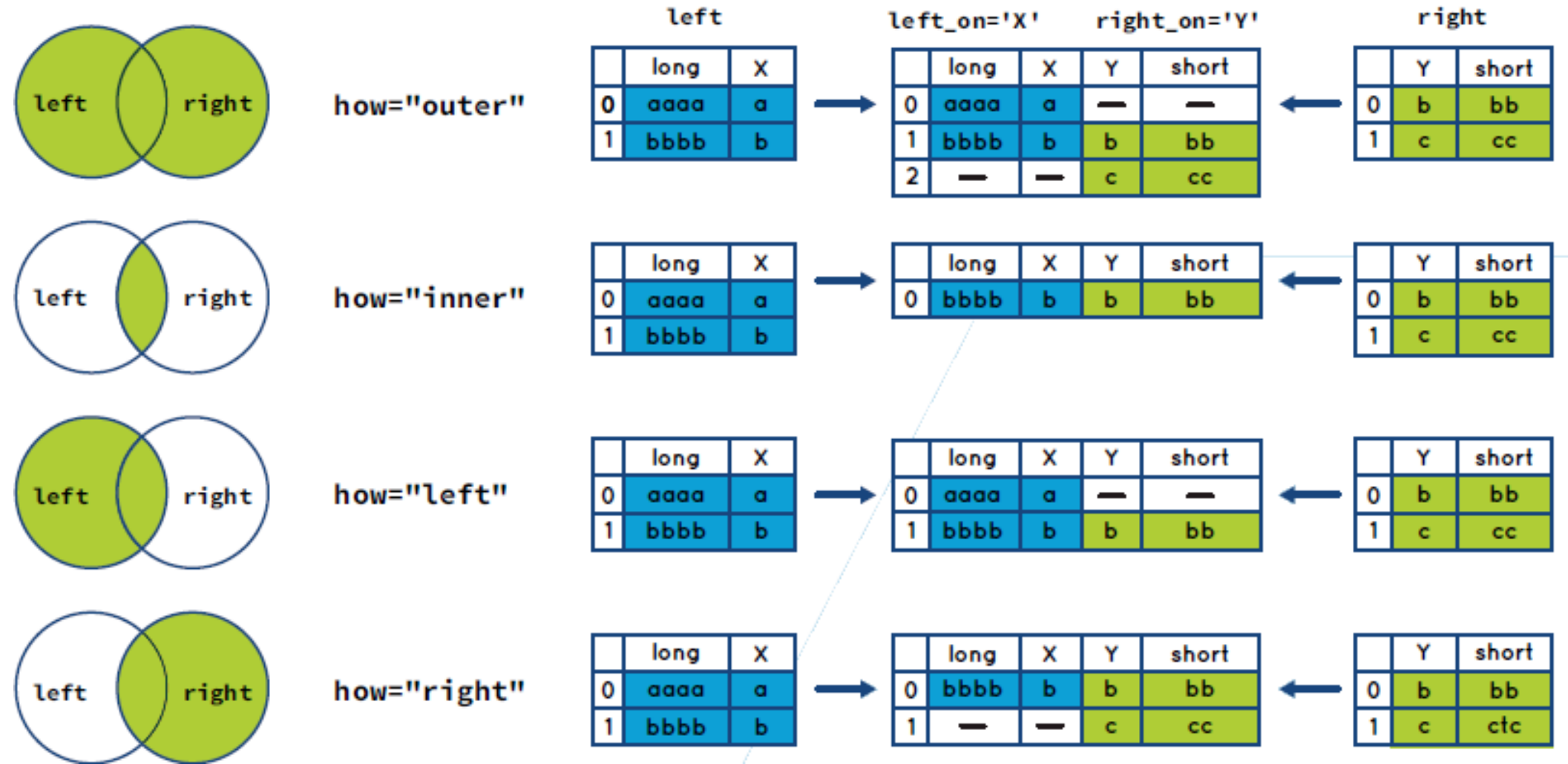
In [57]:

```
# Exportar datos de un data frame a un csv  
d.to_csv('E:/JFB/Python/PYTHON_1/DATOS/sample_data.csv')
```


Importar y exportar datos con pandas

- ▶ El método `read_csv` permite leer datos de un fichero CSV (*Comma Separated Values*) y volcarlos a una DF.
- ▶ Los parámetros más utilizados son:
 - `Path`: ruta al fichero a cargar
 - `Sep`: carácter a utilizar como separador (por defecto, ',')
 - `Names`: lista de nombres de columnas (opcional)
 - `Header`: indica el número de línea que contiene los nombres de las columnas. Por defecto asume que es la primera (0) salvo si se especifican los nombres de las columnas con `names`.
 - `na_values`: valores a considerar como "NA" además de los "estándar"
 - `index_col`: columna(s) a usar como índice(s) en el DF

Pandas: combinando Dataframes con merge y join



DataFrames en Apache Spark

1. Introducción a Azure Databricks y PySpark

Azure Databricks es una plataforma basada en Apache Spark que facilita el procesamiento de grandes volúmenes de datos de manera distribuida.

- **PySpark** permite manipular datos en estructuras llamadas DataFrames, similares a las tablas en bases de datos relacionales.
- **Beneficio clave:** procesamiento distribuido eficiente.

Transformaciones en Apache Spark

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col

# Crear una sesión de Spark
spark = SparkSession.builder.appName("TransformacionDataFrame").getOrCreate()

# Crear un DataFrame de ejemplo
data = [("Juan", "IT", 3000),
        ("Maria", "HR", 4000),
        ("Pedro", "IT", 3500),
        ("Laura", "HR", 4500)]

columns = ["Empleado", "Departamento", "Sueldo"]
df = spark.createDataFrame(data, columns)

# Filtrar empleados del departamento de IT
df_it = df.filter(df.Departamento == "IT")# Aplicar un aumento de 10% al sueldo
df_it = df_it.withColumn("Sueldo_Aumento", df_it["Sueldo"] * 1.10)
df_it.show()
```

Transformaciones con SQL Functions en Pyspark

```
from pyspark.sql.functions import expr
```

```
# Aplicar una expresión SQL directamente en la columna Sueldo
```

```
df = df.withColumn("Sueldo_Aumento", expr("Sueldo * 1.10"))df.show()
```

Ventaja: Usar expresiones SQL permite una sintaxis concisa y optimizada para operaciones en columnas.

Particionamiento en Parquet

```
df.write.partitionBy("Departamento").parquet("/mnt/data/empleados_particionado.parquet")
```

El particionamiento mejora el rendimiento en consultas y almacenamiento.

Particionamiento en formato Delta

```
df.write.partitionBy("Año").format("delta").save("/mnt/data/delta_table")
```

Beneficios del Particionamiento:

- **Mejor rendimiento:** Spark lee solo las particiones necesarias.
- **Optimización de almacenamiento:** Datos mejor organizados.
- **Escalabilidad:** Procesamiento distribuido más eficiente.

Compresión de datos

```
f.write.option("compression", "snappy").parquet("/mnt/data/output.parquet")
```

La compresión reduce el espacio de almacenamiento y mejora la velocidad de lectura.

Conclusiones

- PySpark permite realizar transformaciones eficientes en grandes volúmenes de datos.
- La combinación de transformaciones SQL y PySpark mejora la flexibilidad y escalabilidad.
- El uso de particionamiento y compresión optimiza el rendimiento en Databricks.

¡GRACIAS!

