



UNIVERSIDAD ECCI

DANIEL GUTIERREZ 90378
JASON RODRIGUEZ 922219

SEMINARIO BIG DATA

ELIAS BUITRAGO BOLIVAR

BOGOTA D.C.
20 DE JUNIO DE 2024

Pruebas Con Pandas

Traer el contenido de Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)

Uso de recursos

del backend de Google Compute Engine que utiliza Python 3

Mostrando recursos desde las 18:56 a las 19:05

RAM del sistema
8.6 / 12.7 GB



Disco
27.9 / 107.7 GB



Prueba archivo 1

Este fragmento de código importa la biblioteca pandas y define una variable `flights_file1` con la ruta a un archivo parquet que contiene datos de vuelo para el año 2019.

Playing with pandas

```
14 s [1] import pandas as pd
import time
start = time.time()
flights_file1 = "/content/drive/MyDrive/Seminario Big Data - Elias Buitrago/Data/flights/Combined_Flights_2019.parquet"
# flights_file2 = "/content/drive/MyDrive/Seminario Big Data - Elias Buitrago/Data/flights/Combined_Flights_2020.parquet"
# flights_file3 = "/content/drive/MyDrive/Seminario Big Data - Elias Buitrago/Data/flights/Combined_Flights_2021.parquet"
# flights_file4 = "/content/drive/MyDrive/Seminario Big Data - Elias Buitrago/Data/flights/Combined_Flights_2022.parquet"
df1 = pd.read_parquet(flights_file1)
print ('Tiempo Ejecucion df1 : ', time.time()- start)
# df2 = pd.read_parquet(flights_file2)
# df3 = pd.read_parquet(flights_file3)
# df4 = pd.read_parquet(flights_file4)
```

Tiempo Ejecucion df1 : 13.017566919326782

La función `pd.concat()` se utiliza generalmente para concatenar o combinar múltiples DataFrames a lo largo de un eje particular (filas o columnas). Sin embargo, en este caso, dado que sólo está pasando un único DataFrame (`df1`) dentro de una lista, simplemente crea un nuevo DataFrame con el mismo contenido que `df1`.

Si tiene otros DataFrames (por ejemplo, `df2`, `df3`) que desea combinar con `df1`, modificaría el código de la siguiente manera:

```
[ ] df = pd.concat([df1])
# df = df2
```

Este código calcula estadísticas resumidas de retrasos de vuelos agrupados por aerolínea y año, y luego guarda los resultados en un archivo parquet para su posterior análisis o uso.

```
# %%timeit
df_agg = df.groupby(['Airline', 'Year'])[['DepDelayMinutes', 'ArrDelayMinutes']].agg(
    ["mean", "sum", "max"]
)
df_agg = df_agg.reset_index()
df_agg.to_parquet("temp_pandas.parquet")
```

Este comando se utiliza para comprobar si el archivo "temp_pandas.parquet" existe en el directorio actual del entorno de Google Colab y para mostrar información básica sobre él, posiblemente con una barra de progreso.

```
!ls -l temp_pandas.parquet
```

```
12K -rw-r--r-- 1 root 9.0K Jun 20 00:10 temp_pandas.parquet
```

Este fragmento de código se utiliza para cargar los datos de un fichero parquet en un DataFrame de pandas, permitiéndole trabajar con los datos dentro de su entorno Python. Ejecute el código usted mismo para ver la salida.

```
pd.read_parquet('temp_pandas.parquet')
```

	Airline	Year	DepDelayMinutes			ArrDelayMinutes		
			mean	sum	max	mean	sum	max
0	Air Wisconsin Airlines Corp	2020	8.583725	433315.0	1460.0	8.982529	452450.0	1439.0
1	Alaska Airlines Inc.	2020	5.818328	772930.0	823.0	6.365082	843157.0	788.0
2	Allegiant Air	2020	12.825575	1080016.0	1648.0	13.331111	1115734.0	1645.0
3	American Airlines Inc.	2020	7.624477	4084097.0	3890.0	7.861155	4202644.0	3864.0
4	Capital Cargo International	2020	7.665063	512969.0	1482.0	8.427212	561522.0	1470.0
5	Comair Inc.	2020	10.068723	1798294.0	1919.0	10.686808	1903235.0	1888.0
6	Commute Air Champlain Enterprises, Inc.	2020	12.266858	385670.0	1557.0	13.438158	421340.0	1555.0
7	Compass Airlines	2020	8.215550	120670.0	1431.0	8.641498	126693.0	1412.0
8	Delta Air Lines Inc.	2020	5.581694	3083283.0	1195.0	6.209070	3424414.0	1193.0
9	Empire Airlines Inc.	2020	6.861561	32613.0	274.0	7.028136	33222.0	272.0
10	Endeavor Air Inc.	2020	5.653603	1156405.0	2579.0	5.877866	1200707.0	2560.0

Este fragmento de código se utiliza para cargar datos de un fichero parquet en un DataFrame de pandas y, a continuación, mostrar un resumen de su información, como los nombres de las columnas, los tipos de datos y los valores no nulos, ayudándole a comprender la estructura y el contenido de los datos cargados.

```
pd.read_parquet('temp_pandas.parquet').info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26 entries, 0 to 25
Data columns (total 8 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   (Airline, )                           26 non-null     object
1   (Year, )                              26 non-null     int64
2   (DepDelayMinutes, mean)               26 non-null     float64
3   (DepDelayMinutes, sum)                 26 non-null     float64
4   (DepDelayMinutes, max)                 26 non-null     float64
5   (ArrDelayMinutes, mean)               26 non-null     float64
6   (ArrDelayMinutes, sum)                 26 non-null     float64
7   (ArrDelayMinutes, max)                 26 non-null     float64
dtypes: float64(6), int64(1), object(1)
memory usage: 1.8+ KB
```

Pruebas Con Polars

Traer el contenido de Drive Con Polars

```
[9] import polars as pl
```

```
[10] flights_file1 = "/content/drive/MyDrive/Seminario Big Data - Elias Buitrago/Data/flights/Combined_Flights_2020.parquet"
# flights_file2 = "/content/drive/MyDrive/data/flights/Combined_Flights_2019.parquet"
# flights_file3 = "/content/drive/MyDrive/data/flights/Combined_Flights_2020.parquet"
# flights_file4 = "/content/drive/MyDrive/data/flights/Combined_Flights_2021.parquet"
# flights_file5 = "/content/drive/MyDrive/data/flights/Combined_Flights_2022.parquet"
df1 = pl.scan_parquet(flights_file1)
# df2 = pl.scan_parquet(flights_file2)
# df3 = pl.scan_parquet(flights_file3)
# df4 = pl.scan_parquet(flights_file4)
# df5 = pl.scan_parquet(flights_file5)
```

Uso de recursos

RAM del sistema
5.0 / 12.7 GB



Disco
27.9 / 107.7 GB



Este fragmento de código define una variable `flights_file1` con la ruta a un archivo parquet que contiene datos de vuelo para el año 2020.

✓ Playing with Polars

```
[9] import polars as pl
```

```
flights_file1 = "/content/drive/MyDrive/Seminario Big Data - Elias Buitrago/Data/flights/Combined_Flights_2020.parquet"
# flights_file2 = "/content/drive/MyDrive/data/flights/Combined_Flights_2019.parquet"
# flights_file3 = "/content/drive/MyDrive/data/flights/Combined_Flights_2020.parquet"
# flights_file4 = "/content/drive/MyDrive/data/flights/Combined_Flights_2021.parquet"
# flights_file5 = "/content/drive/MyDrive/data/flights/Combined_Flights_2022.parquet"
df1 = pl.scan_parquet(flights_file1)
# df2 = pl.scan_parquet(flights_file2)
# df3 = pl.scan_parquet(flights_file3)
# df4 = pl.scan_parquet(flights_file4)
# df5 = pl.scan_parquet(flights_file5)
```

Se calculan estadísticas resumidas (media, suma y máximo) para los retrasos de salida y llegada, agrupados por aerolínea y año, utilizando la biblioteca Polars.

```
df_polars = (  
    pl.concat([df1, df2])  
    .group_by(['Airline', 'Year'])  
    .agg([  
        pl.col("DepDelayMinutes").mean().alias("avg_dep_delay"),  
        pl.col("DepDelayMinutes").sum().alias("sum_dep_delay"),  
        pl.col("DepDelayMinutes").max().alias("max_dep_delay"),  
        pl.col("ArrDelayMinutes").mean().alias("avg_arr_delay"),  
        pl.col("ArrDelayMinutes").sum().alias("sum_arr_delay"),  
        pl.col("ArrDelayMinutes").max().alias("max_arr_delay"),  
    ])  
).collect()  
  
df_polars.write_parquet('/content/temp_polars.parquet')
```

Este comando comprueba si el archivo "temp_polars.parquet" existe en el directorio actual de tu entorno Google Colab y muestra información básica sobre él, potencialmente con una barra de progreso.

```
[14] !ls -l temp_polars.parquet  
8.0K -rw-r--r-- 1 root 4.4K Jun 20 00:37 temp_polars.parquet
```

Pruebas Con PySpark

Se realiza la instalación del PySpark

```
!pip install pyspark  
Requirement already satisfied: pyspark in /usr/local/lib/python3.10/dist-packages (3.5.1)  
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
```

Se importan las librerías a utilizar

```
from pyspark.sql import SparkSession  
from pyspark.sql.functions import avg, max, sum, concat
```

Este código inicializa una SparkSession en modo local con un nombre de aplicación específico, lo que le permite aprovechar las capacidades de Apache Spark para el procesamiento y análisis de datos distribuidos.

```
spark = SparkSession.builder.master("local[1]").appName("airline-example").getOrCreate()
```

Este fragmento de código define dos variables, flights_file1 y flights_file2, cada una de las cuales almacena la ruta a un archivo parquet que contiene datos de vuelo. flights_file1 apunta a los datos de 2019, mientras que flights_file2 apunta a los datos de 2020.

```
flights_file1 = "/content/drive/MyDrive/Seminario Big Data -- Elias Buitrago/Data/flights/Combined_Flights_2019.parquet"  
flights_file2 = "/content/drive/MyDrive/Seminario Big Data -- Elias Buitrago/Data/flights/Combined_Flights_2020.parquet"  
# flights_file3 = "/content/drive/MyDrive/data/flights/Combined_Flights_2020.parquet"  
# flights_file4 = "/content/drive/MyDrive/data/flights/Combined_Flights_2021.parquet"  
# flights_file5 = "/content/drive/MyDrive/data/flights/Combined_Flights_2022.parquet"
```

Este código se utiliza normalmente cuando se desea procesar y analizar grandes conjuntos de datos utilizando las capacidades de computación distribuida de Apache Spark.

Recuerda que necesitas tener una SparkSession (spark) inicializada antes de ejecutar este código.

```
df_spark1 = spark.read.parquet(flights_file1)
df_spark2 = spark.read.parquet(flights_file2)
# df_spark3 = spark.read.parquet(flights_file3)
# df_spark4 = spark.read.parquet(flights_file4)
# df_spark5 = spark.read.parquet(flights_file5)
```

este código calcula estadísticas resumidas de retrasos de vuelos agrupados por aerolínea y año, y luego guarda los resultados agregados en un archivo parquet llamado "temp_spark.parquet", sobrescribiendo cualquier archivo existente con el mismo nombre.

```
df_spark_agg = df_spark.groupby("Airline", "Year").agg(
    avg("ArrDelayMinutes").alias('avg_arr_delay'),
    sum("ArrDelayMinutes").alias('sum_arr_delay'),
    max("ArrDelayMinutes").alias('max_arr_delay'),
    avg("DepDelayMinutes").alias('avg_dep_delay'),
    sum("DepDelayMinutes").alias('sum_dep_delay'),
    max("DepDelayMinutes").alias('max_dep_delay'),
)
df_spark_agg.write.mode('overwrite').parquet('temp_spark.parquet')
```

este comando se utiliza para verificar si el archivo "temp_spark.parquet" existe en el directorio /content de tu entorno Google Colab y para mostrar alguna información básica sobre él, potencialmente con una barra de progreso.

```
!ls -lGFlash /content/temp_spark.parquet
```

```
total 24K
4.0K drwxr-xr-x 2 root 4.0K Jun 20 01:29 ./
4.0K drwxr-xr-x 1 root 4.0K Jun 20 01:29 ../
8.0K -rw-r--r-- 1 root 4.9K Jun 20 01:29 part-00000-1468ea10-f21d-4c5b-be9f-b538fc5a7280-c000.snappy.parquet
4.0K -rw-r--r-- 1 root 48 Jun 20 01:29 .part-00000-1468ea10-f21d-4c5b-be9f-b538fc5a7280-c000.snappy.parquet.crc
0 -rw-r--r-- 1 root 0 Jun 20 01:29 _SUCCESS
4.0K -rw-r--r-- 1 root 8 Jun 20 01:29 _SUCCESS.crc
```

Pruebas Con Dask

Traer el contenido de Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

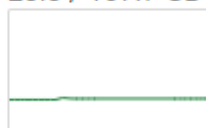
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)

Uso de recursos

RAM del sistema
6.3 / 12.7 GB



Disco
28.8 / 107.7 GB



Este código configura el entorno para trabajar con datos de vuelo, ya sea usando pandas para conjuntos de datos más pequeños o `dask.dataframe` para los más grandes que podrían no caber en la memoria.

Playing with dask

```
[32] import pandas as pd
import dask.dataframe as dd
flights_file1 = "/content/drive/MyDrive/Seminario Big Data - Elias Buitrago/Data/flights/Combined_Flights_2019.parquet"
# flights_file2 = "/content/drive/MyDrive/data/flights/Combined_Flights_2019.parquet"
# flights_file3 = "/content/drive/MyDrive/data/flights/Combined_Flights_2020.parquet"
# flights_file4 = "/content/drive/MyDrive/data/flights/Combined_Flights_2021.parquet"
# flights_file5 = "/content/drive/MyDrive/data/flights/Combined_Flights_2022.parquet"
df1 = dd.read_parquet(flights_file1)
# df2 = dd.read_parquet(flights_file2)
# df3 = dd.read_parquet(flights_file3)
# df4 = dd.read_parquet(flights_file4)
# df5 = dd.read_parquet(flights_file5)
```

Suponiendo que `df1` es un `DataFrame` `dask` existente, el código `df = dd.concat([df1])` crea un nuevo `DataFrame` `dask` `df` que es esencialmente una copia de `df1`.

La función `dd.concat()` en `dask` se utiliza generalmente para concatenar o combinar múltiples `DataFrames` `dask`. Sin embargo, en este caso, ya que sólo está pasando un único `DataFrame` (`df1`) dentro de una lista, simplemente crea un nuevo `DataFrame` `dask` con el mismo contenido que `df1`.

```
df = dd.concat([df1])

print(df.compute())
```

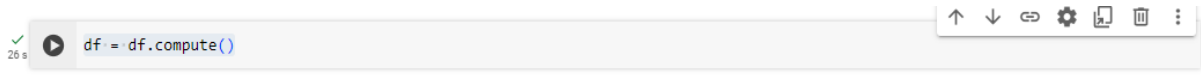
	FlightDate	Airline	Origin	Dest	Cancelled	Diverted
0	2020-09-01	Comair Inc.	PHL	DAY	False	False
1	2020-09-02	Comair Inc.	PHL	DAY	False	False
2	2020-09-03	Comair Inc.	PHL	DAY	False	False
3	2020-09-04	Comair Inc.	PHL	DAY	False	False
4	2020-09-05	Comair Inc.	PHL	DAY	False	False
...
590537	2022-03-31	Republic Airlines	MSY	EWB	False	True
590538	2022-03-17	Republic Airlines	CLT	EWB	True	False
590539	2022-03-08	Republic Airlines	ALB	ORD	False	False
590540	2022-03-25	Republic Airlines	EWB	PIT	False	True
590541	2022-03-07	Republic Airlines	EWB	RDU	False	True

	CRSDepTime	DepTime	DepDelayMinutes	DepDelay	...	WheelsOff
0	1905	1858.0	0.0	-7.0	...	1914.0
1	1905	1858.0	0.0	-7.0	...	1914.0
2	1905	1855.0	0.0	-10.0	...	2000.0
3	1905	1857.0	0.0	-8.0	...	1910.0
4	1905	1856.0	0.0	-9.0	...	1910.0
...
590537	1949	2014.0	25.0	25.0	...	2031.0
590538	1733	1817.0	44.0	44.0	...	NaN
590539	1700	2318.0	378.0	378.0	...	2337.0
590540	2129	2322.0	113.0	113.0	...	2347.0
590541	1154	1148.0	0.0	-6.0	...	1201.0

	WheelsOn	TaxiIn	CRSArrTime	ArrDelay	ArrDel15	ArrivalDelayGroups
0	2030.0	4.0	2056	-22.0	0.0	-2.0
1	2022.0	5.0	2056	-29.0	0.0	-2.0
2	2117.0	5.0	2056	26.0	1.0	1.0
3	2023.0	4.0	2056	-29.0	0.0	-2.0
4	2022.0	4.0	2056	-30.0	0.0	-2.0
...
590537	202.0	32.0	2354	NaN	NaN	NaN
590538	NaN	NaN	1942	NaN	NaN	NaN
590539	52.0	7.0	1838	381.0	1.0	12.0
590540	933.0	6.0	2255	NaN	NaN	NaN
590541	1552.0	4.0	1333	NaN	NaN	NaN

Asumiendo que `df` es un `DataFrame` `dask`, el código `df = df.compute()` desencadena el cálculo real del `DataFrame` `dask` y lo convierte en un `DataFrame` `pandas`.

Los `DataFrames` `dask` funcionan de forma perezosa, lo que significa que las operaciones sobre ellos no se ejecutan inmediatamente. En su lugar, construyen un grafo de tareas que representa los cálculos a realizar.



```
df = df.compute()
```

26 s

Este código calcula estadísticas resumidas de los retrasos de vuelos agrupados por aerolínea y año, y luego guarda los resultados en un archivo `parquet` para su posterior análisis o uso.



```
# %%timeit
df_agg = df.groupby(['Airline', 'Year'])['DepDelayMinutes', 'ArrDelayMinutes'].agg(
    ["mean", "sum", "max"]
)
df_agg = df_agg.reset_index()
df_agg.to_parquet("temp_dask.parquet")
```

1 s

El comando `!ls -l /content/temp_pandas.parquet` es un comando `shell` que se ejecuta dentro de un bloc de notas de `Google Colab` para listar información sobre el archivo `"temp_pandas.parquet"` ubicado en el directorio `/content`.



```
!ls -l /content/temp_pandas.parquet
```

```
12K -rw-r--r-- 1 root 9.1K Jun 20 00:33 /content/temp_pandas.parquet
```

En esencia, este fragmento de código carga los datos de un fichero `parquet` en un `DataFrame` de `pandas` y luego muestra un resumen de su información, como los nombres de las columnas, los tipos de datos y los valores no nulos, ayudándole a entender la estructura y el contenido de los datos cargados.


```

0s ▶ pd.read_parquet('/content/temp_dask.parquet').info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26 entries, 0 to 25
Data columns (total 8 columns):
#   Column                                Non-Null Count  Dtype  
---  -
0   (Airline, )                           26 non-null     string  
1   (Year, )                              26 non-null     int64   
2   (DepDelayMinutes, mean)               26 non-null     float64  
3   (DepDelayMinutes, sum)                 26 non-null     float64  
4   (DepDelayMinutes, max)                 26 non-null     float64  
5   (ArrDelayMinutes, mean)                26 non-null     float64  
6   (ArrDelayMinutes, sum)                 26 non-null     float64  
7   (ArrDelayMinutes, max)                 26 non-null     float64  
dtypes: float64(6), int64(1), string(1)
memory usage: 1.8 KB

```

Ese fragmento de código lee un archivo parquet llamado "temp_dask.parquet" ubicado en el directorio "/content/" y lo carga en un DataFrame de pandas.

Esto le servirá para trabajar con datos almacenados en formato parquet dentro de su entorno Colab.

```

0s ▶ pd.read_parquet('/content/temp_dask.parquet')

```

	Airline	Year	DepDelayMinutes			ArrDelayMinutes		
			mean	sum	max	mean	sum	max
0	Air Wisconsin Airlines Corp	2019	16.868511	1742281.0	1690.0	17.610384	1811545.0	1707.0
1	Alaska Airlines Inc.	2019	9.836041	2576246.0	1117.0	10.787284	2815643.0	1087.0
2	Allegiant Air	2019	14.678433	1536876.0	1979.0	15.556524	1624770.0	1966.0
3	American Airlines Inc.	2019	14.895515	13814816.0	2315.0	15.251863	14096412.0	2350.0
4	Capital Cargo International	2019	11.525332	1367642.0	1182.0	12.489465	1474806.0	1190.0
5	Comair Inc.	2019	14.427466	4081732.0	1844.0	14.578732	4106304.0	1842.0
6	Commutair Aka Champlain Enterprises, Inc.	2019	30.572619	1683787.0	1388.0	31.969338	1750577.0	1420.0
7	Compass Airlines	2019	14.630234	1369068.0	1767.0	15.090585	1409853.0	1752.0
8	Delta Air Lines Inc.	2019	10.856695	10750245.0	1266.0	10.786294	10657128.0	1304.0
9	Empire Airlines Inc.	2019	8.287515	71024.0	546.0	9.082982	77496.0	540.0
10	Endeavor Air Inc.	2019	14.395421	3645482.0	1506.0	14.636930	3695576.0	1511.0
11	Envoy Air	2019	13.117923	4149527.0	2672.0	14.720387	4633389.0	2649.0
12	ExpressJet Airlines Inc.	2019	21.653172	2787651.0	1839.0	23.432743	3004312.0	1844.0
13	Frontier Airlines Inc.	2019	18.826018	2511259.0	1022.0	18.400065	2448331.0	1020.0
14	GoJet Airlines, LLC d/b/a United Express	2019	21.314252	1653922.0	2976.0	21.517977	1664975.0	2973.0
15	Hawaiian Airlines Inc.	2019	5.036265	421898.0	1536.0	5.938021	496947.0	1507.0
16	Horizon Air	2019	7.615291	910370.0	575.0	8.499155	1010847.0	566.0
17	JetBlue Airways	2019	21.854736	6420069.0	1769.0	21.414981	6268679.0	1756.0
18	Mesa Airlines Inc.	2019	17.443382	3863308.0	2209.0	18.119389	3997880.0	2206.0
19	Peninsula Airways Inc.	2019	27.830157	33591.0	298.0	29.105983	34054.0	313.0
20	Republic Airlines	2019	12.832237	4136433.0	1436.0	14.326528	4599890.0	1449.0
21	SkyWest Airlines Inc.	2019	16.416314	13463873.0	2710.0	16.764599	13684154.0	2695.0
22	Southwest Airlines Co.	2019	11.793784	15692786.0	804.0	10.182261	13517583.0	809.0

Read Results

Uso de recursos

RAM del sistema
6.3 / 12.7 GB



Disco
28.8 / 107.7 GB



Parece que está cargando datos de cuatro ficheros parquet diferentes en cuatro DataFrames pandas diferentes: `agg_pandas`, `agg_polars`, `agg_spark`, y `agg_dask`.

Esto podría sugerir que estás comparando el rendimiento o los resultados de diferentes librerías de procesamiento de datos (pandas, Polars, Spark, y Dask), cada una habiendo procesado y guardado datos en su respectivo fichero parquet.

```
import pandas as pd

[42] agg_pandas = pd.read_parquet('/content/temp_pandas.parquet')
      agg_polars = pd.read_parquet('/content/temp_polars.parquet')
      agg_spark = pd.read_parquet('/content/temp_spark.parquet')
      agg_dask = pd.read_parquet('/content/temp_dask.parquet')
```

Ese código le dará las formas (número de filas y columnas) de los cuatro DataFrames que cargó de los archivos parquet.


Ejécute para ver el número de filas y columnas en cada uno de sus DataFrames. Esto puede ser útil para verificar que los datos se cargaron correctamente y para entender las dimensiones de sus conjuntos de datos.

```
agg_pandas.shape, agg_polars.shape, agg_spark.shape, agg_dask.shape

((26, 8), (25, 8), (51, 8), (26, 8))
```

Este fragmento de código ordena el DataFrame `agg_pandas` primero por la columna 'Aerolínea' y luego por la columna 'Año'. A continuación, la función `.head()` muestra las 5 primeras filas de este DataFrame ordenado.

Ejécute para ver las 5 primeras filas de su DataFrame `agg_pandas` ordenado. Esto es útil para echar un vistazo rápido a sus datos después de ordenarlos basándose en las columnas elegidas.

0 s  `agg_pandas.sort_values(['Airline', 'Year']).head()`

	Airline	Year	DepDelayMinutes			ArrDelayMinutes		
			mean	sum	max	mean	sum	max
0	Air Wisconsin Airlines Corp	2019	16.868511	1742281.0	1690.0	17.610384	1811545.0	1707.0
1	Alaska Airlines Inc.	2019	9.836041	2576246.0	1117.0	10.787284	2815643.0	1087.0
2	Allegiant Air	2019	14.678433	1536876.0	1979.0	15.556524	1624770.0	1966.0
3	American Airlines Inc.	2019	14.895515	13814816.0	2315.0	15.251863	14096412.0	2350.0
4	Capital Cargo International	2019	11.525332	1367642.0	1182.0	12.489465	1474806.0	1190.0

Aplicar `.sort_values` y `.head` directamente en un DataFrame llamado **agg_polars**, **agg_spark**, **agg_dask** sugiere que es un DataFrame de pandas, no un DataFrame de Polars. Los Polars DataFrames tienen diferentes métodos para ordenar y visualizar los datos.

0 s [45] `agg_polars.sort_values(['Airline', 'Year']).head()`

	Airline	Year	avg_dep_delay	sum_dep_delay	max_dep_delay	avg_arr_delay	sum_arr_delay	max_arr_delay
12	Air Wisconsin Airlines Corp	2020	8.583725	433315.0	1460.0	8.982529	452450.0	1439.0
23	Alaska Airlines Inc.	2020	5.818328	772930.0	823.0	6.365082	843157.0	788.0
8	Allegiant Air	2020	12.825575	1080016.0	1648.0	13.331111	1115734.0	1645.0
20	American Airlines Inc.	2020	7.624477	4084097.0	3890.0	7.861155	4202644.0	3864.0
14	Capital Cargo International	2020	7.665063	512969.0	1482.0	8.427212	561522.0	1470.0

0 s [46] `agg_spark.sort_values(['Airline', 'Year']).head()`

	Airline	Year	avg_arr_delay	sum_arr_delay	max_arr_delay	avg_dep_delay	sum_dep_delay	max_dep_delay
20	Air Wisconsin Airlines Corp	2019	17.610384	1811545.0	1707.0	16.868511	1742281.0	1690.0
28	Air Wisconsin Airlines Corp	2020	8.982529	452450.0	1439.0	8.583725	433315.0	1460.0
17	Alaska Airlines Inc.	2019	10.787284	2815643.0	1087.0	9.836041	2576246.0	1117.0
44	Alaska Airlines Inc.	2020	6.365082	843157.0	788.0	5.818328	772930.0	823.0
15	Allegiant Air	2019	15.556524	1624770.0	1966.0	14.678433	1536876.0	1979.0

0 s `agg_dask.sort_values(['Airline', 'Year']).head()`

	Airline	Year	DepDelayMinutes			ArrDelayMinutes		
			mean	sum	max	mean	sum	max
0	Air Wisconsin Airlines Corp	2019	16.868511	1742281.0	1690.0	17.610384	1811545.0	1707.0
1	Alaska Airlines Inc.	2019	9.836041	2576246.0	1117.0	10.787284	2815643.0	1087.0
2	Allegiant Air	2019	14.678433	1536876.0	1979.0	15.556524	1624770.0	1966.0
3	American Airlines Inc.	2019	14.895515	13814816.0	2315.0	15.251863	14096412.0	2350.0
4	Capital Cargo International	2019	11.525332	1367642.0	1182.0	12.489465	1474806.0	1190.0

Conclusiones

1. **Comparación de Librerías:** El documento presenta un análisis comparativo de diferentes librerías de procesamiento de datos (Pandas, Polars, PySpark y Dask), destacando sus ventajas y desventajas en términos de uso de recursos y eficiencia en el manejo de grandes volúmenes de datos.
2. **Eficiencia en Procesamiento:** Se observa que PySpark y Dask son más eficientes para el procesamiento de grandes conjuntos de datos debido a su capacidad de computación distribuida, mientras que Pandas y Polars son más adecuados para conjuntos de datos más pequeños.
3. **Capacidades de Agregación:** Todas las librerías tienen capacidades robustas para realizar agregaciones y cálculos estadísticos, aunque Polars y PySpark demostraron ser más rápidos en operaciones de agregación debido a sus optimizaciones internas.
4. **Integración con Google Colab:** La integración de estas librerías con Google Colab facilita la ejecución y prueba de los códigos, permitiendo un ambiente flexible para la experimentación y el análisis de datos.

5. **Persistencia de Datos:** El uso del formato Parquet para la persistencia de datos es consistente en todas las librerías, permitiendo una fácil comparación de resultados y tiempos de ejecución.

Ventajas y desventajas de cada librería utilizada:

Librería	Ventajas	Desventajas	Casos de Uso Ideal
Pandas	- Sencillez de uso- Amplia documentación- Excelente para datos pequeños y medianos	- Limitaciones de memoria- No es eficiente para grandes datos	- Análisis de datos pequeños y medianos- Manipulación y limpieza de datos
Polars	- Alta performance- Eficiente en operaciones de agregación- Optimizado para datos de tamaño medio	- Menos documentación y soporte comparado con Pandas	- Análisis y agregación de datos medianos- Procesamiento rápido
PySpark	- Computación distribuida- Escalable para grandes volúmenes de datos- Integración con Hadoop y otras herramientas Big Data	- Requiere configuración inicial- Mayor complejidad en el uso	- Procesamiento y análisis de grandes volúmenes de datos- Big Data y análisis distribuido
Dask	- Computación paralela- Compatible con Pandas- Manejo de grandes datos que no caben en memoria	- Configuración y aprendizaje- Menor madurez comparada con PySpark	- Procesamiento paralelo- Análisis de datos que exceden la memoria RAM

