**ELECCIONES EEUU 2020 – DOCUMENTACIÓN PROCESO ANALÍTICO**

**DATOS ENTRADA**

Partimos de un TXT con la transcripción del debate entre Biden y Trump de 2020.

Fuente de datos Kaggle: https://www.kaggle.com/datasets/headsortails/us-election-2020-presidential-debates/data

Aunque **parece caótico**, tiene **cierta estructura repetida**:

* Nombre del hablante
* Timestamp
* Fragmento del discurso

👉 **Esto lo convierte en datos semiestructurados.**

**🛠️ ¿Qué usar en Spark: RDD, DataFrame o Dataset?**

| **Herramienta** | **¿Cuándo usarla?** | **¿Aquí aplica?** |
| --- | --- | --- |
| **RDD** | Texto plano, control fino, parsing personalizado. | ✅ Ideal para parsear este .txt. |
| **DataFrame** | Datos estructurados con columnas fijas. Muy eficiente para análisis y agregación. | ✅ Una vez parseado a columnas. |
| **Dataset** | Casos donde necesitas tipos fuertes (Scala/Java). | ❌ No esencial aquí, salvo que quieras trabajar fuertemente tipado en Scala. |

👉 **Conclusión:**  
**1. Primero RDD**, para hacer el parsing del texto.  
**2. Luego DataFrame**, para hacer análisis y agregaciones más fáciles y eficientes.

**PASOS ANALISIS**

**✅ Paso 0: Iniciar una sesión de Spark**

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName("Debate2020Analysis") \

.getOrCreate()

**📄 Paso 1: Leer el archivo como RDD**

* Subir archivo a Databricks
  + File > Upload data to DBFS > Drag and drop TXT file > Upload

rdd = spark.sparkContext.textFile("us\_election\_2020\_1st\_presidential\_debate.txt

")

**🧹 Paso 2: Parsear el texto**

El objetivo es identificar líneas con este patrón:

import re

def parse\_lines(lines):

results = []

current\_speaker = None

current\_timestamp = None

current\_text = []

for line in lines:

match = re.match**(r"^(.+?): \((\d{2}:\d{2})\)$",** line.strip())

if match:

# Si ya teníamos un bloque anterior, lo guardamos

if current\_speaker:

results.append((current\_speaker, current\_timestamp, " ".join(current\_text)))

current\_speaker = match.group(1)

current\_timestamp = match.group(2)

current\_text = []

else:

if line.strip():

current\_text.append(line.strip())

# Guardar el último bloque

if current\_speaker:

results.append((current\_speaker, current\_timestamp, " ".join(current\_text)))

return results

parsed\_rdd = rdd.mapPartitions(lambda lines: iter(parse\_lines(lines)))

Explicación código

**1. pattern = r"^(.\*?):\s\*\((\d{2}:\d{2})\)\s\*(.\*)"**

Este **patrón de expresión regular (regex)** dice:

* **^(.\*?)**: desde el principio, captura todo hasta el primer : (el nombre del hablante).
* **\s\*:** posibles espacios en blanco.
* **\((\d{2}:\d{2})\):** entre paréntesis, captura el timestamp tipo 01:20.
* **\s\***: más espacios.
* **(.\*)**: el resto de la línea, es el texto hablado.

**🧱 Paso 3: Convertir a DataFrame**

python

CopiarEditar

from pyspark.sql import Row

df = parsed\_rdd.map(lambda x: Row(speaker=x[0], timestamp=x[1], text=x[2])).toDF()

df.show(truncate=False)

**🧪 Ya puedes hacer análisis como:**

python

CopiarEditar

# Número de intervenciones por orador

df.groupBy("speaker").count().show()

# Palabras por intervención (y total por orador)

from pyspark.sql.functions import length, split

df = df.withColumn("word\_count", length("text") - length("text").replace(" ", "") + 1)

df.groupBy("speaker").sum("word\_count").show()