TP7 - Informe Databrics

Análisis de Datos en la nube

Chavez, Moya, Saldeña, Silva
INSTITUTO TECNOLÓGICO UNIVERSITARIO – SEDE ESTE

Contenido


UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO
itu

Introducción

El crecimiento exponencial de los datos en las organizaciones modernas exige herramientas capaces de procesarlos de forma rápida, eficiente y escalable. Apache Spark se ha consolidado como el estándar de facto para el procesamiento distribuido de Big Data, ofreciendo velocidades hasta 100x superiores a MapReduce tradicional gracias a su arquitectura en memoria.

Databricks, la plataforma unificada creada por los fundadores de Apache Spark, democratiza el acceso a estas capacidades mediante una interfaz colaborativa que integra notebooks interactivos, clusters gestionados automáticamente y capacidades avanzadas de visualización.

¿Por qué Databricks?

Databricks ofrece ventajas clave para este tipo de análisis:

- Escalabilidad automática: Clusters que se ajustan a la carga de trabajo
- Notebooks interactivos: Exploración iterativa de datos con PySpark
- Integración nativa con Delta Lake: ACID transactions en data lakes
- Visualizaciones integradas: Matplotlib, Seaborn sin configuración adicional
- Colaboración en tiempo real: Trabajo en equipo sobre el mismo notebook
- Free Edition disponible: Acceso gratuito para aprendizaje

Contexto del Caso: Tarjetas Black

Entre 2003 y 2012, más de 86 directivos, consejeros y sindicalistas de Caja Madrid (posteriormente fusionada en Bankia) utilizaron tarjetas de crédito opacas, conocidas como "tarjetas black", para realizar gastos personales por un total de aproximadamente 15.5 millones de euros de dinero público.

Cronología

- 2003-2012: Período de uso irregular de las tarjetas
- 2014: Se conoce públicamente el caso
- 2017: Sentencia judicial condena a varios implicados

El caso adquirió especial relevancia porque coincidió con la crisis financiera de 2008-2009, cuando:

- Miles de ciudadanos perdían sus empleos
- El desempleo alcanzó el 26%
- Familias perdían sus viviendas
- Caja Madrid requería rescate público

Datos a analizar

Nuestro dataset contiene:

- 64,651 transacciones originales
- Período: 2003-2011 (9 años)
- Columnas: NIF, Nombre, Tarjeta, Fecha, Hora, Operación, Importe, Comercio
- Fuente: Datos judiciales público

Preparación del Entorno en Databricks

Paso 1: Registrarse en Databricks

1. Registro y logueo en https://login.databricks.com/signup
2. Acceso a la interfaz principal

Paso 2: Iniciar Cluster (SQL Warehouse)

1. Vamos a la sección de Compute
2. Buscamos el “Serverless Starter Warehouse”
3. En la parte izquierda le damos play y lo iniciamos.

Paso 3: Creamos Notebook

1. Vamos a “new” en la parte superior derecha de la página.
2. Y clickeamos un nuevo notebook (Lo definimos para programar en Python).

Paso 4: Importación de CSV

1. Volvemos a clicar “new” y ahora clickeamos “App or upload Data”
2. Nos llevará a un apartado para agregar datos y clickeamos en “Upload files to a Volume”
3. Ahí elegimos un catálogo (Workspace en este caso)
4. En ese catálogo creamos un schema (En la cruz de arriba a la derecha)
5. En este caso lo llamamos “transactions”
6. Y dentro del schema creamos un volumen (Lo llamamos “transactions_volume”)
7. Ahí dentro podemos directamente mover el archivo o buscarlo entre los directorios e importar el csv.

Ya estando una vez registrado y tener configurado el entorno en databricks podemos usar el notebook:

Notebook Completo TP Transacciones _ Databricks (Explicación por Celda)

Importación de Librerías y creación de SparkSession

Como primer paso importamos todas las librerías necesarias para el proyecto, entre las más importantes están:

- **SparkSession**: Nos permite iniciar una sesión en spark y por lo tanto usar los recursos del cluster.
- **Pyspark.sql.functions:** Son todas las funciones principales de Pyspark.
- **Pyspark.sql.types:** Los tipos de datos que vamos a utilizar.
- **Matplotlib.pyplot y Seaborn:** Para poder visualizar resultados en forma de gráficos.
- **Pandas:** Se complementa con matplotlib y se usa para mejor visualización de las tablas, como de los gráficos y para la exportación de archivos csv.
- **Delta.tables :** Distintas operaciones de Delta Lake.

Python

```python
# Importar librerías de PySpark
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql.window import Window
from delta.tables import DeltaTable

# Importar librerías de visualización y análisis
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from datetime import datetime

# Configurar estilo de visualizaciones
sns.set_style('whitegrid')
sns.set_palette('husl')
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['font.size'] = 10

# Confirmar importación exitosa
print(" Librerías importadas correctamente")
print(f" Fecha de análisis: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
```

```
Librerías importadas correctamente
Fecha de análisis: 2025-11-22 10:43:04
```

También importamos datetime para la fecha y numpy para funciones matemáticas. Además configuramos los estilos de seaborn y matplotlib y finalmente imprimimos la fecha del análisis y que las librerías se importaron correctamente.

```python
# Crear SparkSession
spark = SparkSession.builder \
    .appName("TP_Analisis_Transacciones_Bancarias") \
    .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
    .config("spark.sql.catalog.spark_catalog",
"org.apache.spark.sql.delta.catalog.DeltaCatalog") \
    .getOrCreate()

print(" SPARK SESSION INICIADA")
```

```
SPARK SESSION INICIADA
```

Después creamos una SparkSession a la cual le damos configuraciones de Delta Lake y Sql Spark. (Spark ya está inicializado en databricks, pero esto nos permite configurar distintos aspectos de la sesión).

Carga del Dataset

Usando Unity Catalog cargamos el archivo CSV que habíamos importado anteriormente.

```python
# Cargar archivo CSV desde Unity Catalog Volume
df_raw = spark.read.csv(
    "/Volumes/workspace/transactions/transactions_volume/out.csv",
    header=True,
    sep=";",
    inferSchema=True
)

print(" DATOS CARGADOS CORRECTAMENTE")
print(f" Total de registros: {df_raw.count():,}")
print(f" Número de columnas: {len(df_raw.columns)}")
```

```
DATOS CARGADOS CORRECTAMENTE
Total de registros: 64,651
Número de columnas: 9
```

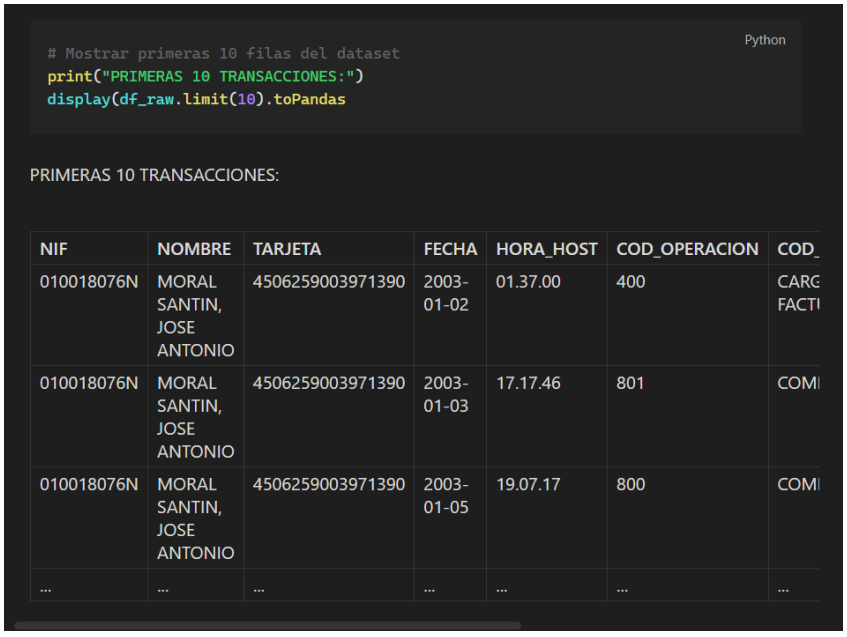
Mostramos que los datos se cargaron correctamente. La totalidad de registros y la totalidad de columnas. **Resultado:** 64,651 registros con 9 columnas cargadas correctamente.

Exploración y Limpieza de Datos

Análisis Exploratorio Inicial

Se realizó una inspección detallada del dataset que incluyó:

1. **Visualización de primeras filas** - Muestra las primeras 10 transacciones del dataset usando limit(10).toPandas() para inspección visual de la estructura y contenido antes del procesamiento.



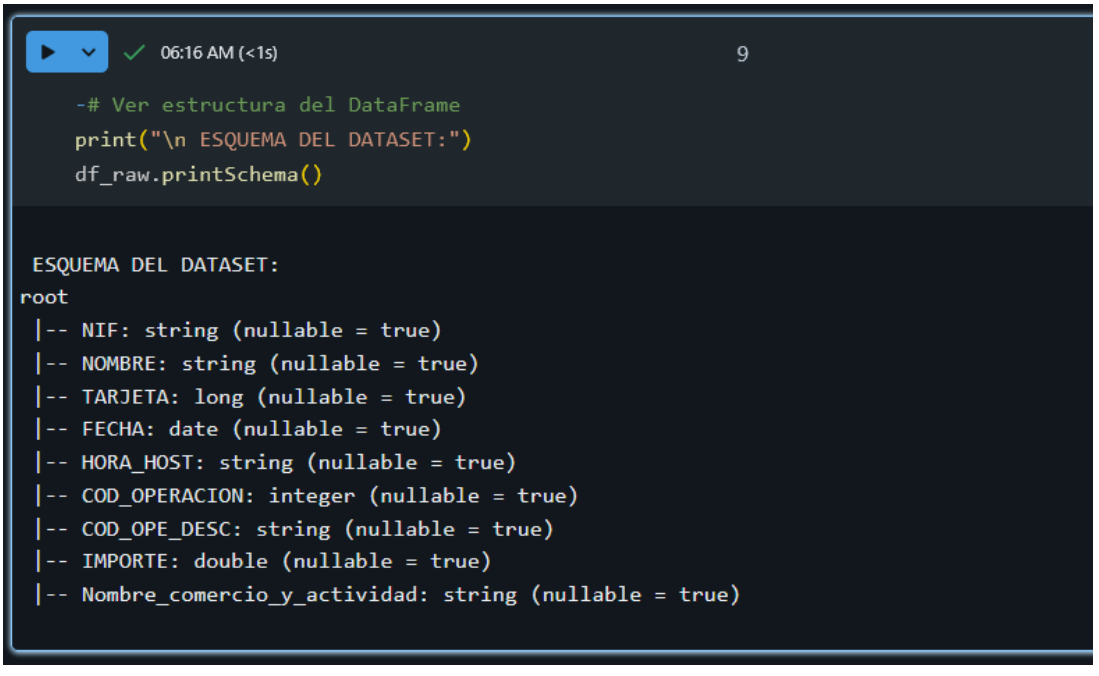
```python
# Mostrar primeras 10 filas del dataset
print("PRIMERAS 10 TRANSACCIONES:")
display(df_raw.limit(10).toPandas
```

Python

PRIMERAS 10 TRANSACCIONES:

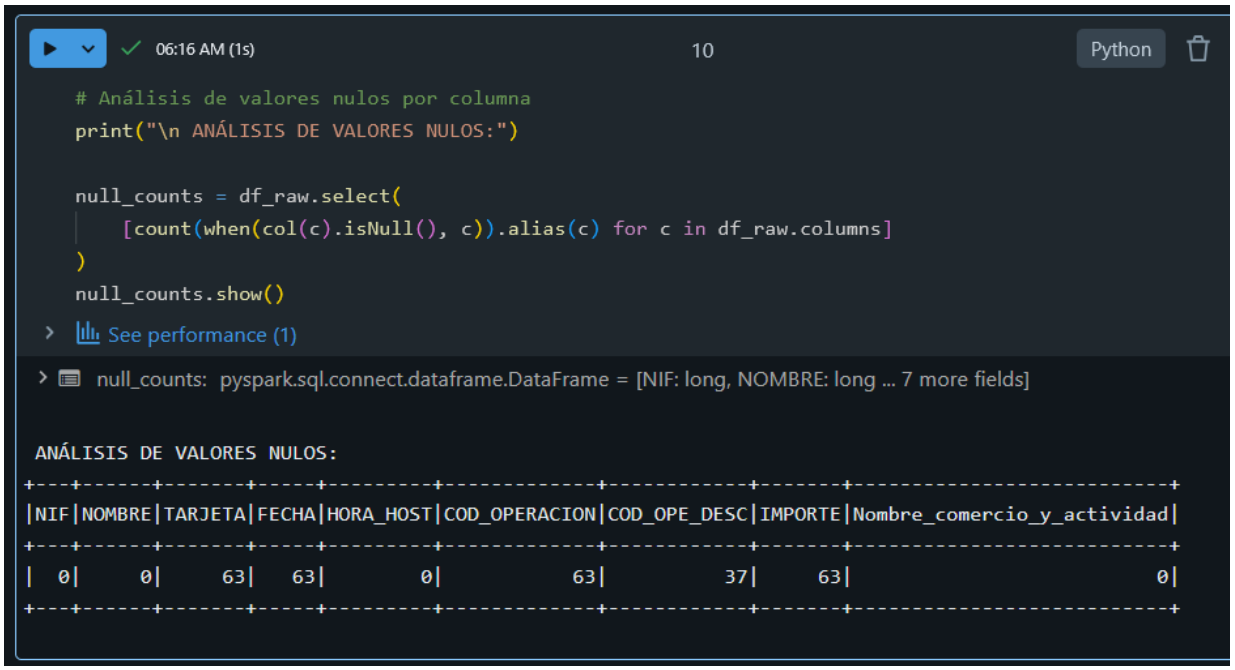
NIF	NOMBRE	TARJETA	FECHA	HORA_HOST	COD_OPERACION	COD_
010018076N	MORAL SANTIN, JOSE ANTONIO	4506259003971390	2003-01-02	01.37.00	400	CARG FACTI
010018076N	MORAL SANTIN, JOSE ANTONIO	4506259003971390	2003-01-03	17.17.46	801	COMI
010018076N	MORAL SANTIN, JOSE ANTONIO	4506259003971390	2003-01-05	19.07.17	800	COMI
...	...	...	...	...	...	...

2. **Análisis del esquema** - Ejecuta printSchema() para examinar la estructura completa del DataFrame.

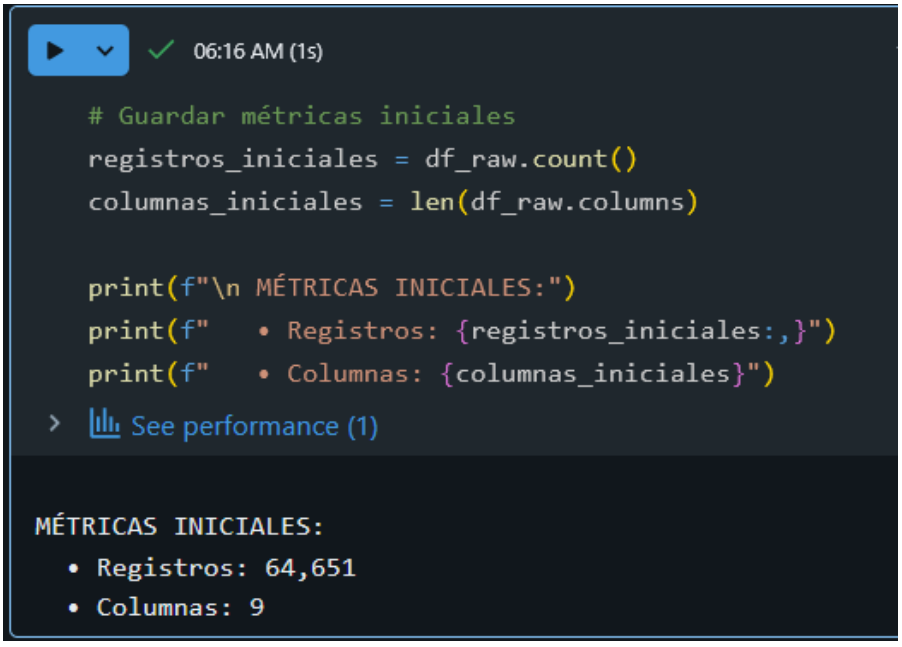


```
06:16 AM (<1s)                                    9
-# Ver estructura del DataFrame
print("\n ESQUEMA DEL DATASET:")
df_raw.printSchema()

 ESQUEMA DEL DATASET:
root
 |-- NIF: string (nullable = true)
 |-- NOMBRE: string (nullable = true)
 |-- TARJETA: long (nullable = true)
 |-- FECHA: date (nullable = true)
 |-- HORA_HOST: string (nullable = true)
 |-- COD_OPERACION: integer (nullable = true)
 |-- COD_OPE_DESC: string (nullable = true)
 |-- IMPORTE: double (nullable = true)
 |-- Nombre_comercio_y_actividad: string (nullable = true)
```

3. **Detección de valores nulos por coma** - Identificar cantidad de nulos por columna. Itera sobre cada columna calculando el número absoluto y porcentaje de valores nulos o vacíos usando isNull() y == '' para identificar información faltante.



4. **Guardado de Métricas Iniciales** - Captura y almacena las métricas fundamentales del dataset original antes de transformaciones: 64,651 registros totales y 9 columnas originales..

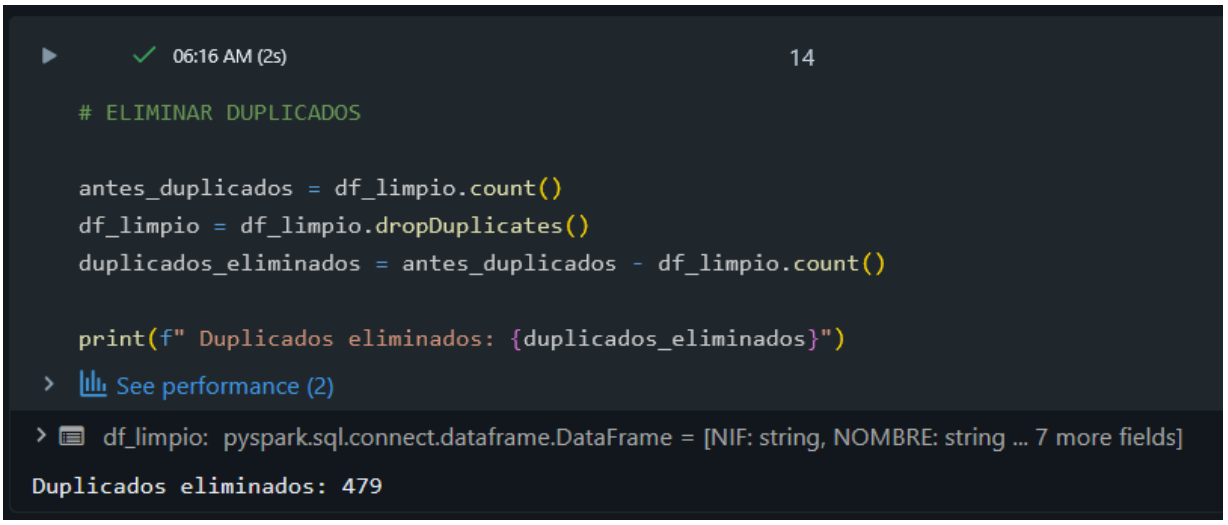


TRANSFORMACIÓN: LIMPIEZA Y PROCESAMIENTO DE DATOS

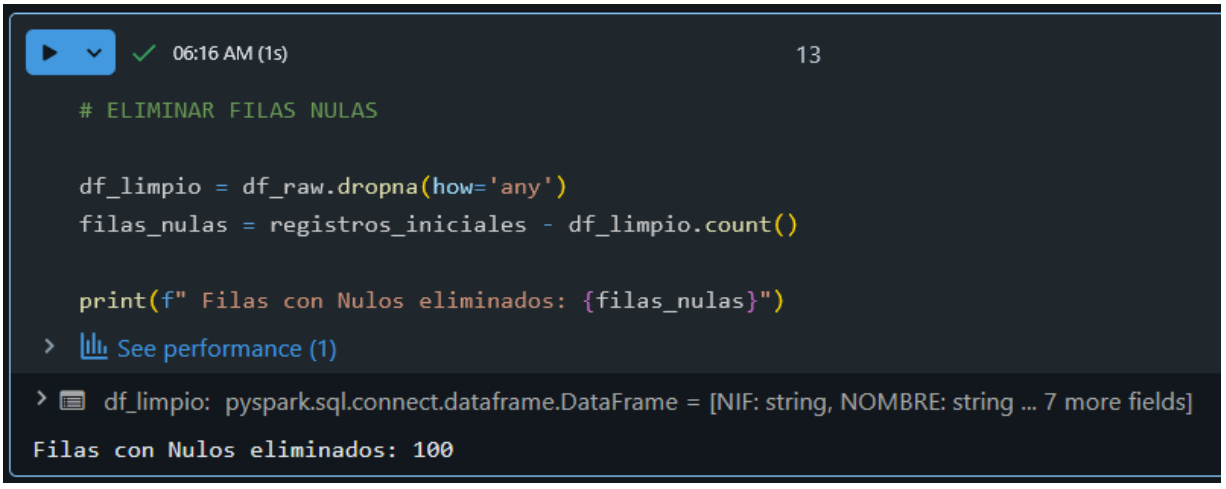
Se aplicaron las siguientes transformaciones para garantizar la calidad de los datos:

Eliminación de Registros Problemáticos

- **Remoción de duplicados** - Utiliza `dropDuplicates()` para identificar y eliminar 479 registros completamente idénticos en todas las columnas, reduciendo el dataset a 64,072 registros únicos.

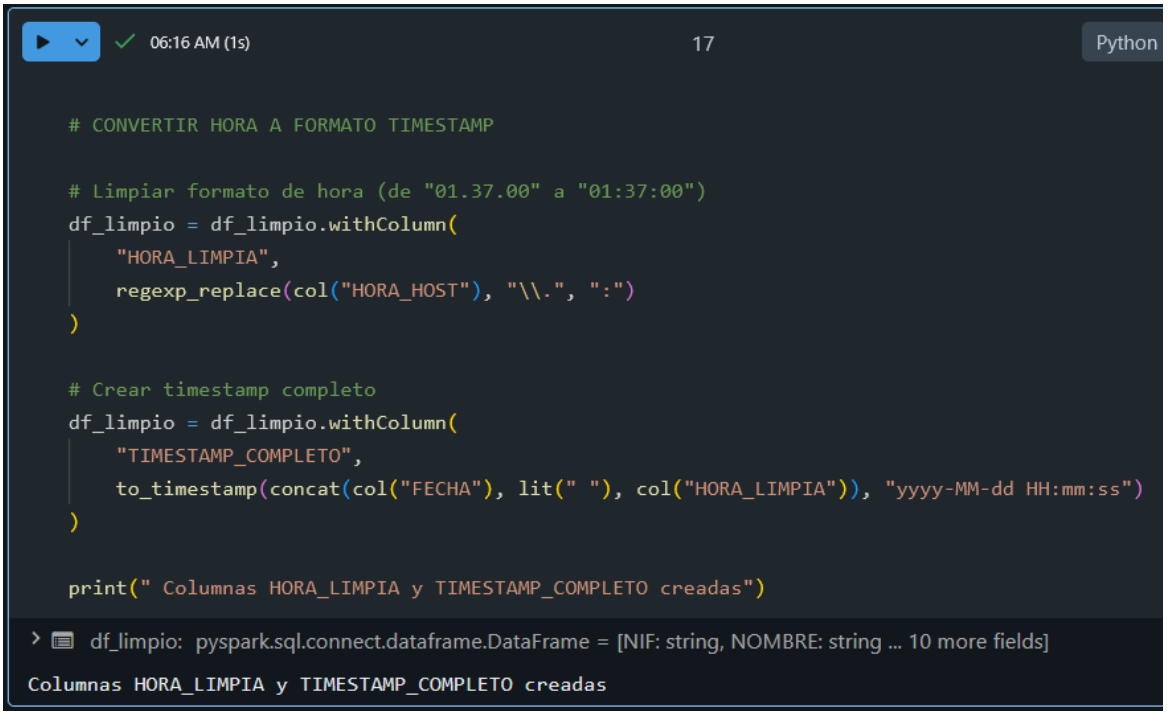


- **Remoción de nulos** - Aplica `dropna(how='any')` para eliminar todas las filas que contengan al menos un valor nulo en cualquier columna.



Conversión de Tipos de Datos

Se convirtió la hora a un formato tipo Timestamp con `to_timestamp()` para mejor manejo de las columnas de tiempo.



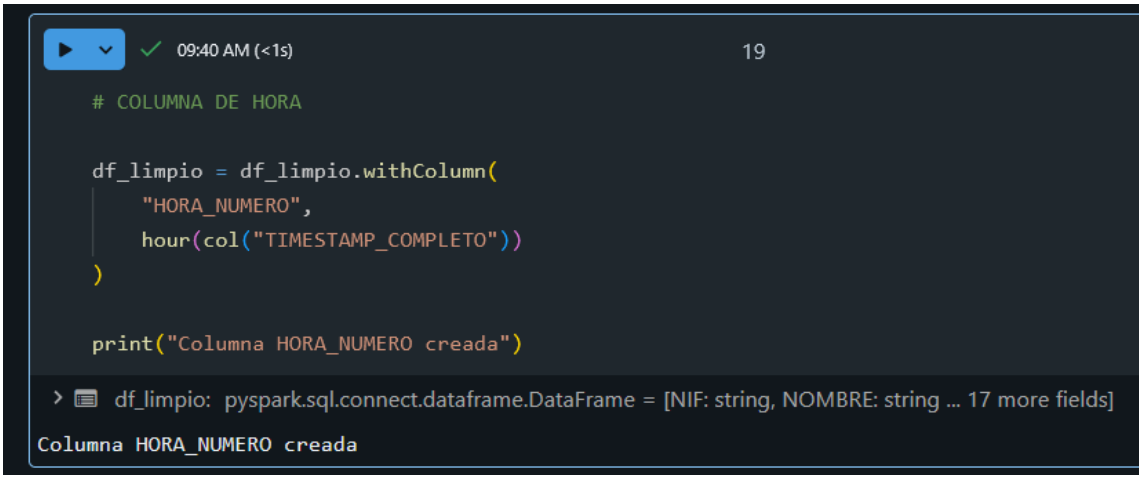
Creación de Columnas Calculadas

Columnas Temporales

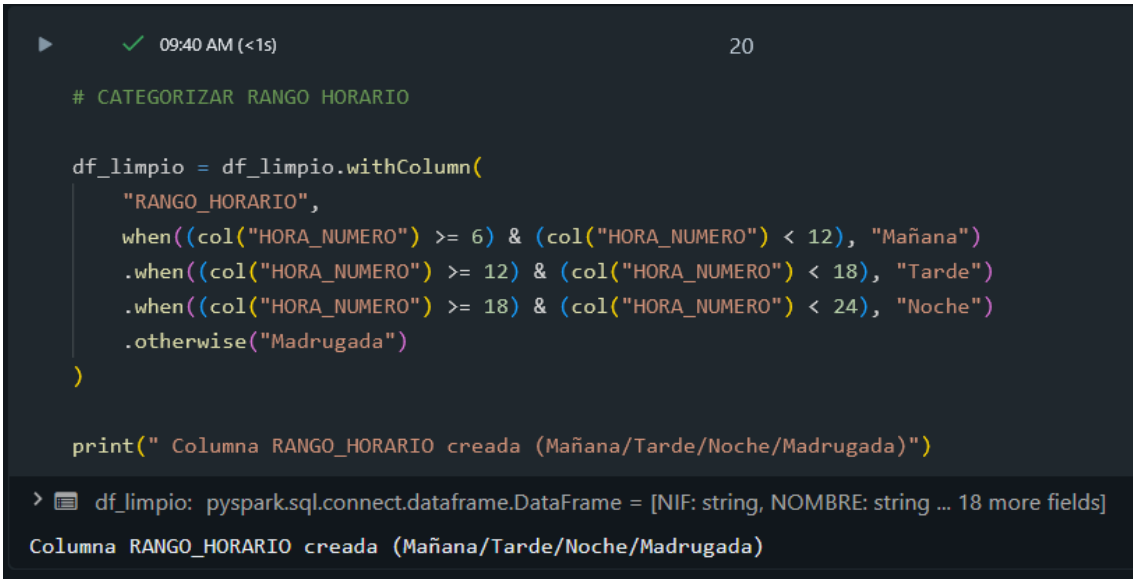
Se generarón las columnas `AÑO`, `MES`, `DIA`, `DIA_SEMANA`, `NOMBRE_DIA`, `NOMBRE_MES`, `TRISMESTRE`. Para poder hacer diferentes análisis de tiempo dentro del notebook.

Columnas Horarias

Se genera la columna `HORA_NUMERO` para solo tomar la hora desde el timestamp completo.



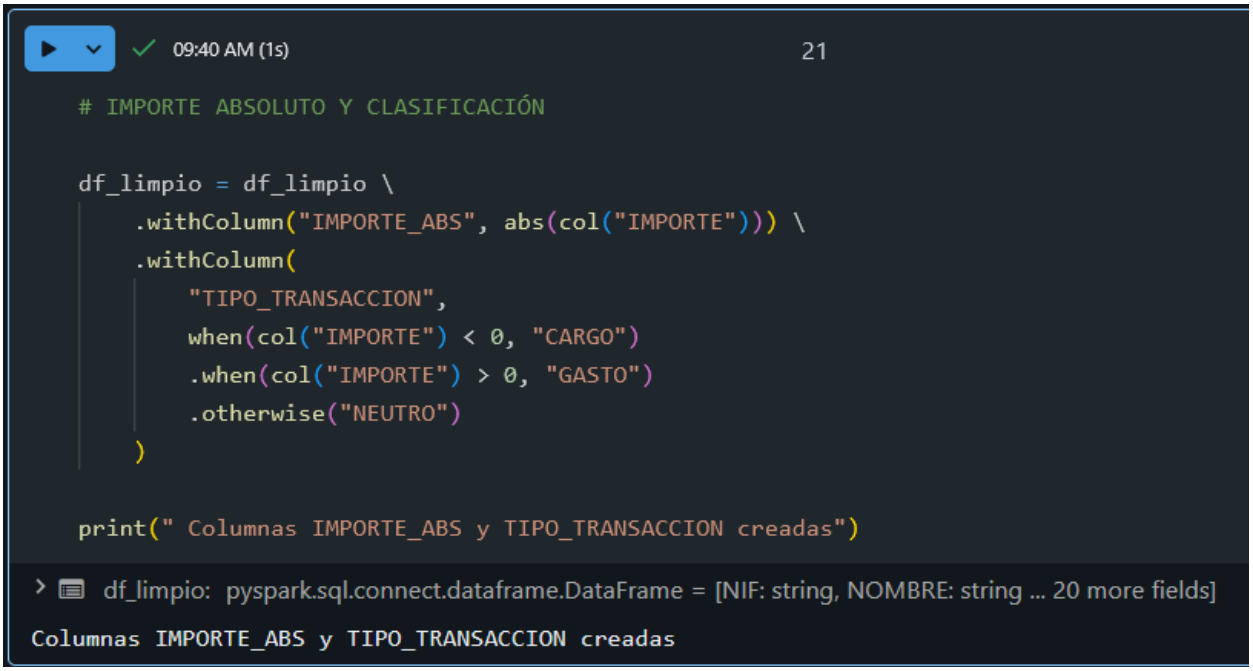
Se genera la columna `RANGO_HORARIO` para categorizar los distintos rangos horarios según la hora de la transacción siendo estos: Mañana, Tarde, Noche, Madrugada (Cada rango corresponde a un ciclo de 6 horas).



Clasificación de transacciones

Creamos las celdas TIPO TRANSACCIÓN e IMPORTE_ABS para clasificar por tipo de transacción (CARGO en caso de ser un importe menor a 0, GASTO en caso de ser un importe mayor a 0 y NEUTRO en cualquier otro caso) y guardar el Importe Absoluto.

¿Por qué IMPORTE_ABS? Para cálculos estadísticos sin signo (promedio de magnitud de transacciones)



Categorizar Montos

Definimos los rangos de los montos en una columna llamada RANGO_GASTO. Estos pueden ser: Bajos (Importes menores a 50), medio (Importes entre 50 y 100), Altos (Entre 150 y 300) y Muy altos o extremos (Mayores a 300).

Creamos la columna CATEGORIA_COMERCIO e NOMBRE_COMERCIO a partir de Nombre_comercio_y_actividad. En una obtendremos las diferentes categorías de comercio y por el otro lado tendremos los diferentes nombres de los comercios en los que fueron realizadas las transacciones.

```python
# EXTRAER CATEGORÍA Y NOMBRE DE COMERCIO

# Dividir el campo en partes separadas por "___"
df_limpio = df_limpio.withColumn(
    "array_parts",
    split(col("Nombre_comercio_y_actividad"), "___")
)

# Extraer categoría con validaciones múltiples
df_limpio = df_limpio.withColumn(
    "CATEGORIA_COMERCIO",
    when(
        (size(col("array_parts")) > 1) &
        (length(trim(col("array_parts").getItem(1))) > 0),  # Verificar que NO
esté vacía
        trim(col("array_parts").getItem(1))
    ).otherwise(lit("SIN CATEGORIA"))
)

# Extraer nombre del comercio
df_limpio = df_limpio.withColumn(
    "NOMBRE_COMERCIO",
    when(
        length(trim(col("array_parts").getItem(0))) > 0,
        trim(col("array_parts").getItem(0))
    ).otherwise(lit("COMERCIO DESCONOCIDO"))
)

# Limpiar columna temporal
df_limpio = df_limpio.drop("array_parts")

print("✅ Columnas CATEGORIA_COMERCIO y NOMBRE_COMERCIO creadas")
print("   - Categorías vacías → 'SIN CATEGORIA'")
print("   - Nombres vacíos → 'COMERCIO DESCONOCIDO'")
```

Resumen métricas finales:

Calcula y presenta un resumen completo comparando estado inicial versus final: pasamos de 64,651

registros a 64,072 (eliminados 579, conservando 99.10%), y de 9 columnas originales a 25 finales

(creadas 16 nuevas columnas).

09:40 AM (1s) 27

```python
# Calcular métricas finales
registros_finales = df_limpio.count()
registros_eliminados = registros_iniciales - registros_finales
porcentaje_eliminado = (registros_eliminados / registros_iniciales) * 100
columnas_finales = len(df_limpio.columns)
columnas_creadas = columnas_finales - columnas_iniciales

print("\n" + "="*70)
print(" RESUMEN DE TRANSFORMACIÓN")

print(f" Registros iniciales:     {registros_iniciales:,}")
print(f" Registros finales:       {registros_finales:,}")
print(f" Registros eliminados:    {registros_eliminados:,} ({porcentaje_eliminado:.2f}%)")
print(f" Columnas iniciales:      {columnas_iniciales}")
print(f" Columnas finales:        {columnas_finales}")
print(f" Columnas creadas:        {columnas_creadas}")
```

See performance (1)

```
======================================================================
RESUMEN DE TRANSFORMACIÓN
Registros iniciales:     64,651
Registros finales:       64,072
Registros eliminados:    579 (0.90%)
Columnas iniciales:      9
Columnas finales:        25
Columnas creadas:        16
```

Esquema después de las transformaciones:

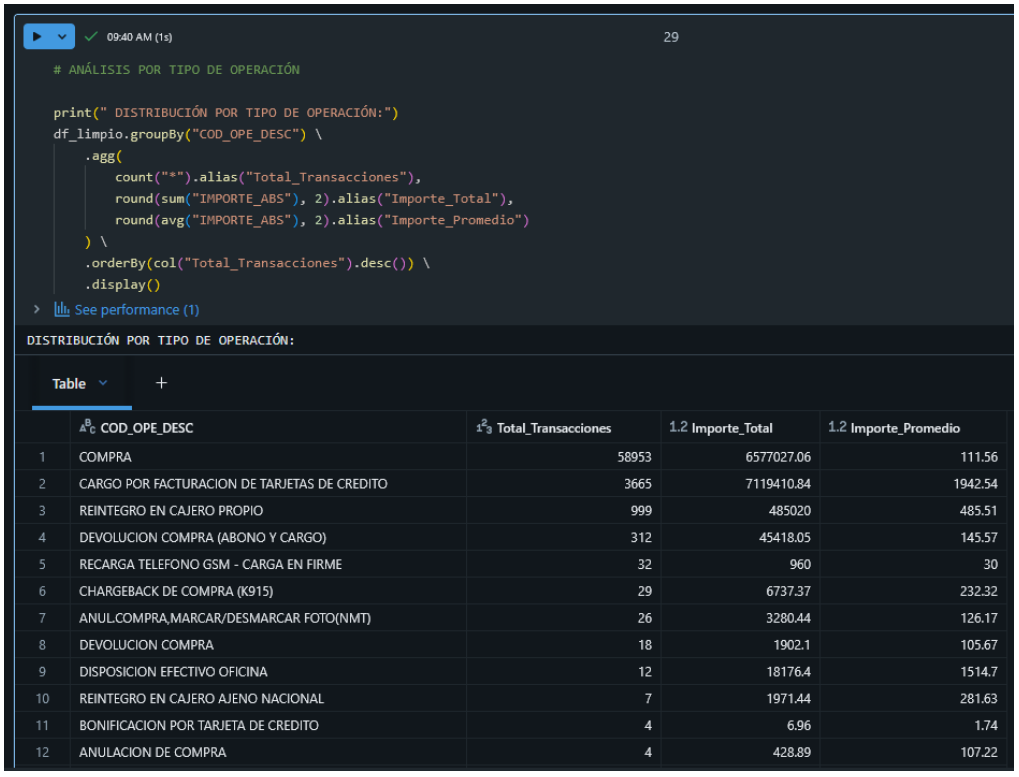
```
09:40 AM (<1s)

# Ver schema actualizado
print("\n ESQUEMA DESPUÉS DE TRANSFORMACIONES:")
df_limpio.printSchema()
```

```
 ESQUEMA DESPUÉS DE TRANSFORMACIONES:
root
 |-- NIF: string (nullable = true)
 |-- NOMBRE: string (nullable = true)
 |-- TARJETA: long (nullable = true)
 |-- FECHA: date (nullable = true)
 |-- HORA_HOST: string (nullable = true)
 |-- COD_OPERACION: integer (nullable = true)
 |-- COD_OPE_DESC: string (nullable = true)
 |-- IMPORTE: double (nullable = true)
 |-- Nombre_comercio_y_actividad: string (nullable = true)
 |-- HORA_LIMPIA: string (nullable = true)
 |-- TIMESTAMP_COMPLETO: timestamp (nullable = true)
 |-- AÑO: integer (nullable = true)
 |-- MES: integer (nullable = true)
 |-- DIA: integer (nullable = true)
 |-- DIA_SEMANA: integer (nullable = true)
 |-- NOMBRE_DIA: string (nullable = true)
 |-- NOMBRE_MES: string (nullable = true)
 |-- TRIMESTRE: integer (nullable = true)
 |-- HORA_NUMERO: integer (nullable = true)
 |-- RANGO_HORARIO: string (nullable = false)
 |-- IMPORTE_ABS: double (nullable = true)
 |-- TIPO_TRANSACCION: string (nullable = false)
 |-- CATEGORIA_COMERCIO: string (nullable = true)
 |-- NOMBRE_COMERCIO: string (nullable = true)
 |-- RANGO_GASTO: string (nullable = false)
```

AGREGACIONES Y ANÁLISIS CON PYSPARK

Análisis por tipo de distribución

Agrupa transacciones por COD_OPE_DESC calculando para cada tipo el número total de transacciones, importe total sumado, e importe promedio, ordenado descendentemente por frecuencia. Revela que 'COMPRA' domina con 58,953 transacciones (91% del total) sumando ~6.5 millones de euros con promedio de 111.56€ por transacción



```
09:40 AM (1s)                                                    29
# ANÁLISIS POR TIPO DE OPERACIÓN

print(" DISTRIBUCIÓN POR TIPO DE OPERACIÓN:")
df_limpio.groupBy("COD_OPE_DESC") \
    .agg(
        count("*").alias("Total_Transacciones"),
        round(sum("IMPORTE_ABS"), 2).alias("Importe_Total"),
        round(avg("IMPORTE_ABS"), 2).alias("Importe_Promedio")
    ) \
    .orderBy(col("Total_Transacciones").desc()) \
    .display()
See performance (1)
DISTRIBUCIÓN POR TIPO DE OPERACIÓN:
```

Table +

	COD_OPE_DESC	Total_Transacciones	Importe_Total	Importe_Promedio
1	COMPRA	58953	6577027.06	111.56
2	CARGO POR FACTURACION DE TARJETAS DE CREDITO	3665	7119410.84	1942.54
3	REINTEGRO EN CAJERO PROPIO	999	485020	485.51
4	DEVOLUCION COMPRA (ABONO Y CARGO)	312	45418.05	145.57
5	RECARGA TELEFONO GSM - CARGA EN FIRME	32	960	30
6	CHARGEBACK DE COMPRA (K915)	29	6737.37	232.32
7	ANUL.COMPRA,MARCAR/DESMARCAR FOTO(NMT)	26	3280.44	126.17
8	DEVOLUCION COMPRA	18	1902.1	105.67
9	DISPOSICION EFECTIVO OFICINA	12	18176.4	1514.7
10	REINTEGRO EN CAJERO AJENO NACIONAL	7	1971.44	281.63
11	BONIFICACION POR TARJETA DE CREDITO	4	6.96	1.74
12	ANULACION DE COMPRA	4	428.89	107.22

Análisis de gastos por año

Filtra solo transacciones tipo 'GASTO', agrupa por año y calcula transacciones totales, gasto total, y ticket promedio, revelando la evolución temporal de

```
12:12 PM (2s)                                                    30
# Analisis de gastos por año
df_limpio_pd = df_limpio.toPandas()
evolucion_anual_pd = df_limpio_pd[df_limpio_pd["TIPO_TRANSACCION"] == "GASTO"].groupby("AÑO").agg(
    Transacciones=("IMPORTE", "count"),
    Total_Gastado=("IMPORTE", "sum"),
    Ticket_Promedio=("IMPORTE", "mean")
).reset_index().sort_values("AÑO")
print(" EVOLUCIÓN DE GASTOS POR AÑO:")

display(evolucion_anual_pd)
See performance (2)
```

df_limpio_pd: pandas.core.frame.DataFrame = [NIF: object, NOMBRE: object ... 23 more fields]
evolucion_anual_pd: pandas.core.frame.DataFrame = [AÑO: int64, Transacciones: int64 ... 2 more fields]

EVOLUCIÓN DE GASTOS POR AÑO:

Table +

	AÑO	Transacciones	Total_Gastado	Ticket_Promedio
1	2003	4804	547250.6396279335	113.91562023895368
2	2004	5193	492087.9301340468	94.75985560062523
3	2005	6058	611938.2799081504	101.01325188315458
4	2006	6639	730557.2200964391	110.04025005218243
5	2007	7274	874235.5000800937	120.18634864999915
6	2008	7442	962215.8400614634	129.29532922083627
7	2009	7060	879309.0398889035	124.54802264715347
8	2010	8228	993179.919894129	120.7073310517901
9	2011	7310	993926.4701186754	135.96805336780784

9 rows | 1.79s runtime

- **Durante la crisis 2008-2009 el promedio no bajó y se mantuvieron gastando de la misma manera.**

Análisis de los gastos de alto valor.

Analizamos los gastos mayores a 500 € y los gastos mayores a 1000 €. De los cuales se identificaron 1,082 transacciones superiores a 500 €, junto con 488 gastos extremos de más de 1,000 €. Todas estas transacciones deberían haber requerido una aprobación especial. Las principales categorías de estos gastos elevados muestran una clara concentración en consumos personales de lujo.

```
df_limpio_pd = df_limpio.toPandas()

# Gastos superiores a 500€
gastos_altos_pd = df_limpio_pd[
    (df_limpio_pd["IMPORTE"] > 500) &
    (df_limpio_pd["TIPO_TRANSACCION"] == "GASTO")
]

total_gastos_altos = len(gastos_altos_pd)
monto_total_alto = gastos_altos_pd["IMPORTE"].sum()
porcentaje = (total_gastos_altos / len(df_limpio_pd[df_limpio_pd["IMPORTE"] > 0])) * 100

print(f"\n GASTOS SUPERIORES A 500€:")
print(f"   • Transacciones: {total_gastos_altos:,} ({porcentaje:.2f}% del total)")
print(f"   • Monto total: {monto_total_alto:,.2f}€")
print(f"   • Promedio por transacción: {monto_total_alto/total_gastos_altos:,.2f}€")

# Top categorías de gastos altos
print(f"\n CATEGORÍAS CON GASTOS >500€:")
top_categorias = gastos_altos_pd.groupby("CATEGORIA_COMERCIO").agg(
    Transacciones=("IMPORTE", "count"),
    Total_Gastado=("IMPORTE", "sum"),
    Ticket_Promedio=("IMPORTE", "mean")
).reset_index().sort_values("Total_Gastado", ascending=False).head(10)
display(top_categorias)

# Gastos extremos (>1000€)
print(f"\n GASTOS EXTREMOS (>1,000€):")
gastos_extremos_pd = df_limpio_pd[df_limpio_pd["IMPORTE"] > 1000]
print(f"   • Transacciones: {len(gastos_extremos_pd):,}")
print(f"   • Monto total: {gastos_extremos_pd['IMPORTE'].sum():,.2f}€")

# Muestra de gastos extremos
print(f"\n EJEMPLOS DE GASTOS EXTREMOS:")
ejemplos_extremos = gastos_extremos_pd[[
    "FECHA", "NOMBRE", "COD_OPE_DESC", "IMPORTE",
    "NOMBRE_COMERCIO", "CATEGORIA_COMERCIO"
]].sort_values("IMPORTE", ascending=False).head(15)
display(ejemplos_extremos)
```

See performance (3)

```
GASTOS SUPERIORES A 500€:
  • Transacciones: 1,902 (3.17% del total)
  • Monto total: 1,817,929.89€
  • Promedio por transacción: 955.80€

CATEGORÍAS CON GASTOS >500€:
```

Table +

	CATEGORIA_COMERCIO	Transacciones	Total_Gastado	Ticket_Promedio
1	CAJEROS AUTOMATICOS	460	276000	600
2	AGENCIAS DE VIAJES	117	181398.56982421875	1550.4151267027244
3	RESTAURANTES RESTO	195	175223.15991210938	898.5803072415865
4	EL CORTE INGLES	138	171092.78979492188	1239.8028246008832
5	CONFECCION TEXTIL EN GENERAL	115	103344.66998291016	898.6493041992187
6	HIPERCOR SUPERMERCADOS EL CORTE INGLES	55	87185.61004638672	1585.192909934304
7	HOTELES 4 Y 5 ESTRELLAS,BALNEARIOS,CAMPI	79	85235.91998291016	1078.9356959862046
8	HOTELES,MOTELES,BALNEARIOS,CAMPINGS REST	54	57345.88018798828	1061.9607442220051
9	AUTOM.Y MOTOCICLETAS (VENTAS Y REPARAC)	42	45002.5500793457	1071.4892876034692
10	JOYERIAS Y RELOJERIAS	33	43888.75	1329.9621212121212

10 rows | 2.64s runtime

```
GASTOS EXTREMOS (>1,000€):
  • Transacciones: 488
  • Monto total: 873,176.61€

EJEMPLOS DE GASTOS EXTREMOS:
```

Table +

	FECHA	NOMBRE	COD_OPE_DESC	IMPORTE	NOMBRE_COMERCIO	CATEGORIA_COMERCIO
1	2007-11-26	ROMERO DE TEJADA Y PICATOSTE, RICARDO	DISPOSICION EFECTIVO OFICI...	11930	COMERCIO DESCONOCIDO	AGENCIAS BANCARIAS(ANTICIPO VENTANILLA)
2	2011-10-05	RODRIGUEZ-PONGA SALAMANCA, ESTANISLAO	COMPRA	8000	EL CORTE INGLES	HIPERCOR SUPERMERCADOS EL CORTE INGLES
3	2010-12-14	RODRIGUEZ-PONGA SALAMANCA, ESTANISLAO	COMPRA	5500	EL CORTE INGLES	HIPERCOR SUPERMERCADOS EL CORTE INGLES
4	2008-12-29	RODRIGUEZ-PONGA SALAMANCA, ESTANISLAO	COMPRA	5500	VIAJES ECI	EL CORTE INGLES
5	2011-02-15	CAFRANGA CAVESTANY, MARIA CARMEN	COMPRA	5500	ICE TRAVEL SPAIN, S.L.	V.DIST.VIAJES Y TRANSPORTE DE VIAJEROS
6	2008-12-27	RODRIGUEZ-PONGA SALAMANCA, ESTANISLAO	COMPRA	5500	VIAJES ECI	EL CORTE INGLES
7	2008-12-30	RODRIGUEZ-PONGA SALAMANCA, ESTANISLAO	COMPRA	5266.02978515625	VIAJES ECI	EL CORTE INGLES
8	2009-12-28	RODRIGUEZ-PONGA SALAMANCA, ESTANISLAO	COMPRA	5000	VIAJES ECI	EL CORTE INGLES
9	2008-12-16	RODRIGUEZ-PONGA SALAMANCA, ESTANISLAO	COMPRA	5000	VIAJES ECI	EL CORTE INGLES
10	2009-12-19	RODRIGUEZ-PONGA SALAMANCA, ESTANISLAO	COMPRA	5000	VIAJES ECI	EL CORTE INGLES
11	2009-12-26	RODRIGUEZ-PONGA SALAMANCA, ESTANISLAO	COMPRA	5000	VIAJES ECI	EL CORTE INGLES
12	2010-12-21	RODRIGUEZ-PONGA SALAMANCA, ESTANISLAO	COMPRA	4985	EL CORTE INGLES	HIPERCOR SUPERMERCADOS EL CORTE INGLES

Análisis de las categorías de comercio con más transacciones:

Filtramos solo gastos excluyendo transacciones sin categoría, agrupa por CATEGORIA_COMERCIO calculando número de transacciones y total gastado, mostrando las 15 categorías con mayor cantidad de transacciones.

12:12 PM (2s)

```python
# TOP CATEGORÍAS DE COMERCIO

print("\n TOP 15 CATEGORÍAS DE COMERCIO:")
display(
    df_limpio.filter(col("CATEGORIA_COMERCIO").isNotNull() &  (col("IMPORTE") > 0))
    .groupBy("CATEGORIA_COMERCIO")
    .agg(
        count("*").alias("Num_Transacciones"),
        round(sum("IMPORTE"), 2).alias("Total_Gastado")
    )
    .orderBy(col("Num_Transacciones").desc())
    .limit(15)
    .toPandas()
)
```

See performance (2)

```
TOP 15 CATEGORÍAS DE COMERCIO:
```

	CATEGORIA_COMERCIO	Num_Transacciones	Total_Gastado
1	RESTAURANTES RESTO	16454	1939216.22
2	GASOLINERAS	4974	259710.28
3	EL CORTE INGLES	3289	489738.6
4	C.A.M.P.S.A.	2836	159418.03
5	RESTAURANTES DE 4 Y 5 TENEDORES	2201	341862.64
6	CAFETERIAS,SNACKS	1977	130246.27
7	GARAJES Y APARCAMIENTOS	1587	14435.63
8	HIPERCOR SUPERMERCADOS EL CORTE INGLES	1584	308599.55
9	SUPERMERCADOS,ULTRAMARINOS, ECONOMATOS	1381	94981.25
10	TAXIS	1242	21612.86
11	LIBRERIAS, PAPELERIAS Y DISCOS	1171	95455.76
12	HOTELES 4 Y 5 ESTRELLAS,BALNEARIOS,CAMPI	1166	215710.56
13	AUTOPISTAS (TERMINALES)	1134	5577.43
14	CAJEROS AUTOMATICOS	1038	487951.44
15	CONFECCION TEXTIL EN GENERAL	1021	246046.58

15 rows | 1.75s runtime

Análisis temporal: Gastos por mes

Filtra solo gastos, agrupa por año y mes calculando transacciones, total gastado, y promedio mensual, generando 108 registros (9 años × 12 meses) que revelan patrones estacionales y períodos de gasto intensivo.

12:12 PM (2s)

```
# ANÁLISIS TEMPORAL: GASTOS POR MES

print("\n GASTOS POR MES:")
df_limpio_pd = df_limpio.toPandas()
gastos_mes = df_limpio_pd[df_limpio_pd["TIPO_TRANSACCION"] == "GASTO"].groupby(["AÑO", "MES"]).agg(
    Transacciones=("IMPORTE", "count"),
    Total_Gastado=("IMPORTE", "sum"),
    Promedio_Gasto=("IMPORTE", "mean")
).reset_index().sort_values(["AÑO", "MES"])
display(gastos_mes)
```

See performance (2)

df_limpio_pd: pandas.core.frame.DataFrame = [NIF: object, NOMBRE: object ... 23 more fields]

gastos_mes: pandas.core.frame.DataFrame = [AÑO: int64, MES: int64 ... 3 more fields]

GASTOS POR MES:

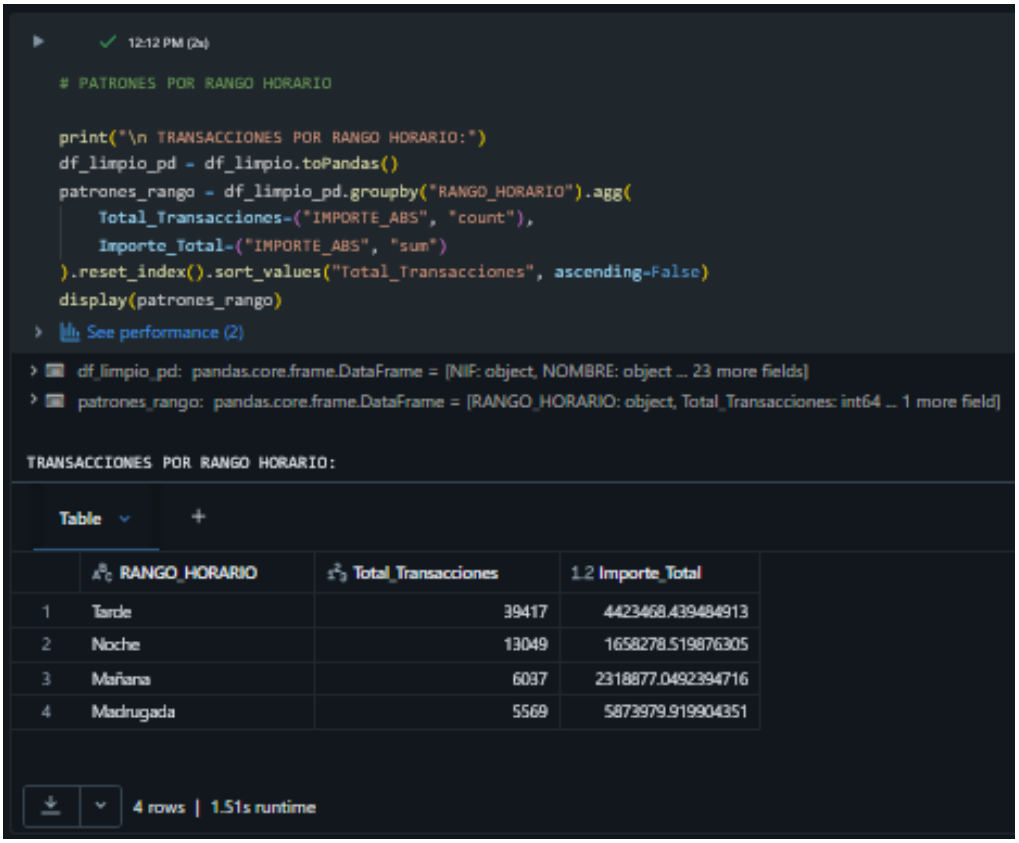
Table

	AÑO	MES	Transacciones	Total_Gastado	Promedio_Gasto
1	2003	1	380	49429.90986084938	130.07871016012996
2	2003	2	336	40667.33999347687	121.03374998058591
3	2003	3	413	49721.99010515213	120.39222785751122
4	2003	4	430	46367.15997123718	107.83060458427252
5	2003	5	372	42462.97984790802	114.14779529007532
6	2003	6	359	42185.2700316906	117.50771596571197
7	2003	7	420	43914.6699783802	104.55873804376239
8	2003	8	310	36458.699915885925	117.60870940608363
9	2003	9	397	46437.420083999634	116.9708314458429
10	2003	10	468	50054.23994612694	106.9535041583909
11	2003	11	441	45612.32994186878	103.42931959607434
12	2003	12	478	53938.62995135784	112.84232207397038
13	2004	1	435	43471.84003710747	99.93526445312061
14	2004	2	421	38177.46998858452	90.68282657621026
15	2004	3	426	37737.870024859905	88.58654935413124

107 rows | 1.79s runtime

Análisis de Patrones por Rango Horario

Se identificaron cuatro rangos horarios de gasto: Tarde (39,417 transacciones; ~4.4 M€), Noche (13,049; 1.6 M€), Mañana (6,037; 2.3 M€) y Madrugada (5,569; 587 K€), siendo la Tarde el período de mayor actividad. Los más de 18,000 gastos registrados entre Noche y Madrugada superan los 2 M€, mostrando un volumen significativo de operaciones fuera del horario laboral.



```
12:12 PM (2s)

# PATRONES POR RANGO HORARIO

print("\n TRANSACCIONES POR RANGO HORARIO:")
df_limpio_pd = df_limpio.toPandas()
patrones_rango = df_limpio_pd.groupby("RANGO_HORARIO").agg(
    Total_Transacciones=("IMPORTE_ABS", "count"),
    Importe_Total=("IMPORTE_ABS", "sum")
).reset_index().sort_values("Total_Transacciones", ascending=False)
display(patrones_rango)
```

See performance (2)

df_limpio_pd: pandas.core.frame.DataFrame = [NIF: object, NOMBRE: object ... 23 more fields]

patrones_rango: pandas.core.frame.DataFrame = [RANGO_HORARIO: object, Total_Transacciones: int64 ... 1 more field]

TRANSACCIONES POR RANGO HORARIO:

Table +

	RANGO_HORARIO	Total_Transacciones	1.2 Importe_Total
1	Tarde	39417	4423468.439484913
2	Noche	13049	1658278.519876305
3	Mañana	6037	2318877.0492394716
4	Madrugada	5569	5873979.919904351

4 rows | 1.51s runtime

Análisis de Comercio con más visitas

Filtra transacciones excluyendo registros sin nombre de comercio ni categoría, agrupa por establecimiento específico calculando visitas, total gastado, y ticket promedio, limitado a top 20 más frecuentados

```
# TOP COMERCIOS MÁS FRECUENTADOS

print("\n TOP 20 COMERCIOS MÁS FRECUENTADOS:")
df_limpio.filter(
    (col("NOMBRE_COMERCIO").isNotNull()) &
    (col("NOMBRE_COMERCIO") != "") &
    (col("IMPORTE") > 0)
) \
    .groupBy("NOMBRE_COMERCIO", "CATEGORIA_COMERCIO") \
    .agg(
        count("*").alias("Visitas"),
        round(sum("IMPORTE"), 2).alias("Total_Gastado"),
        round(avg("IMPORTE"), 2).alias("Ticket_Promedio")
    ) \
    .orderBy(col("Visitas").desc()) \
    .show(20, truncate=False)
```

```
 TOP 20 COMERCIOS MÁS FRECUENTADOS:
+----------------------+----------------------------------------+-------+-------------+--------------+
|NOMBRE_COMERCIO       |CATEGORIA_COMERCIO                      |Visitas|Total_Gastado|Ticket_Promedio|
+----------------------+----------------------------------------+-------+-------------+--------------+
|EL CORTE INGLES       |EL CORTE INGLES                         |2589   |311719.43    |120.4         |
|COMERCIO DESCONOCIDO  |CAJEROS AUTOMATICOS                     |1036   |487351.44    |470.42        |
|EL CORTE INGLES       |HIPERCOR SUPERMERCADOS EL CORTE INGLES  |754    |180648.88    |239.59        |
|COMERCIO DESCONOCIDO  |RESTAURANTES RESTO                      |551    |63556.55     |115.35        |
|ASADOR CRUZ NEVADA    |RESTAURANTES RESTO                      |406    |13619.61     |33.55         |
|E.S.VEGONES DER       |C.A.M.P.S.A.                            |371    |20182.85     |54.4          |
|RENFE 001             |R.E.N.F.E.                              |361    |44113.3      |122.2         |
|PARKING EGUISA C/SEVILLA|GARAJES Y APARCAMIENTOS                 |356    |2408.2       |6.76          |
|EL CORTE INGLES S.A.  |EL CORTE INGLES                         |347    |41935.49     |120.85        |
|PK VELAZQUEZ JORGE JUAN|GARAJES Y APARCAMIENTOS                 |302    |1907.05      |6.31          |
|APARCAMIENTO CORTES   |GARAJES,RESTO DE APARCAMIENTOS          |294    |2741.05      |9.32          |
|ALCAMPO-PIO XII       |ALCAMPO                                 |274    |10524.27     |38.41         |
|FNAC CALLAO           |FNAC                                    |254    |23341.15     |91.89         |
|LA TABERNA DE CHANA   |CAFETERIAS,SNACKS                       |250    |6917.4       |27.67         |
|COMERCIO DESCONOCIDO  |HOTELES 4 Y 5 ESTRELLAS,BALNEARIOS,CAMPI|247    |70826.84     |286.75        |
```

Análisis de Gastos (Top 5 más Gastadores - Concentración del gasto general)

Análisis para saber quienes fueron los que más gastaron dentro del dataset y además datos relevantes para entender el contexto del caso. Como pueden ser el total de personas, el gasto total del periodo, el gasto del top 5 unificado y el promedio por persona.

```
12:12 PM (2s)

print(f"\n ANALISIS DE GASTOS")
print(100*"-")

# Convertir a pandas y filtrar gastos
df_limpio_pd = df_limpio.toPandas()
gastos_por_persona_pd = df_limpio_pd[df_limpio_pd["IMPORTE"] > 0].groupby(["NIF", "NOMBRE"]).agg(
    Total_Transacciones=("IMPORTE", "count"),
    Total_Gastado=("IMPORTE", "sum"),
    Ticket_Promedio=("IMPORTE", "mean"),
    Primera_Transaccion=("FECHA", "min"),
    Ultima_Transaccion=("FECHA", "max")
).reset_index().sort_values("Total_Gastado", ascending=False)

# Top 5 gastadores
print(f"\n TOP 5 MAYORES GASTADORES:")
print(f"{'Posición':<10}{'Nombre':<35}{'Total Gastado':<18}{'Transacciones':<15}{'Promedio':<12}")


top_5 = gastos_por_persona_pd.head(5)
for i, row in enumerate(top_5.itertuples(index=False), 1):
    nombre = row.NOMBRE[:33]  # Truncar si es muy largo
    print(f"{i:<10}{nombre:<35}{row.Total_Gastado:>15,.2f}€  {row.Total_Transacciones:>12,}  {row.Ticket_Promedio:>10,.2f}€")

# Estadísticas generales
total_personas = gastos_por_persona_pd.shape[0]
gasto_total_general = df_limpio_pd[df_limpio_pd["IMPORTE"] > 0]["IMPORTE"].sum()
top_5_gasto = top_5["Total_Gastado"].sum()
porcentaje_top5 = (top_5_gasto / gasto_total_general) * 100

print(f"\n CONCENTRACIÓN DEL GASTO GENERAL:")
print(f"   • Total de personas: {total_personas}")
print(f"   • Gasto total del periodo: {gasto_total_general:,.2f}€")
print(f"   • Top 5 concentran: {top_5_gasto:,.2f}€ ({porcentaje_top5:.1f}% del total)")
print(f"   • Promedio por persona: {gasto_total_general/total_personas:,.2f}€")
```

```
See performance (1)
df_limpio_pd: pandas.core.frame.DataFrame = [NIF: object, NOMBRE: object ... 23 more fields]
gastos_por_persona_pd: pandas.core.frame.DataFrame = [NIF: object, NOMBRE: object ... 5 more fields]
top_5: pandas.core.frame.DataFrame = [NIF: object, NOMBRE: object ... 5 more fields]

ANALISIS DE GASTOS
--------------------------------------------------------------------------------

TOP 5 MAYORES GASTADORES:
Posición  Nombre                          Total Gastado     Transacciones  Promedio
1         MORAL SANTIN, JOSE ANTONIO          439,990.55€         1,197      367.58€
2         DE LA MERCED MONGE, MARIA MERCEDE   285,654.65€         1,596      178.98€
3         BAQUERO NORIEGA, FRANCISCO          265,111.32€         1,072      247.31€
4         ABEJAS JUAREZ, PABLO                253,259.30€         1,711      148.02€
5         ROMERO LAZARO, ANTONIO              246,704.99€         2,086      118.27€

CONCENTRACIÓN DEL GASTO GENERAL:
  • Total de personas: 91
  • Gasto total del período: 7,884,700.84€
  • Top 5 concentran: 1,490,720.81€ (21.0% del total)
  • Promedio por persona: 77,853.86€
```

SPARK SQL: CONSULTAS CON SQL

Vista Temporal y Resumen general del Dataset

Registra df_limpio como vista temporal llamada 'transacciones' usando createOrReplaceTempView(), que funciona exactamente como tabla de base de datos relacional pero existe solo durante la sesión Spark actual y después con la función spark.sql( ) podemos ejecutar sentencias sql usando la vista transacciones como la tabla relacional. Mostramos datos relevantes como total de transacciones, total de clientes, total de categorías, etc.

```
12:12 PM (<1s)
# REGISTRAR DATAFRAME COMO VISTA TEMPORAL

df_limpio.createOrReplaceTempView("transacciones")
print(" Vista temporal 'transacciones' creada")
See performance (1)
Vista temporal 'transacciones' creada
```

```
print(" RESUMEN GENERAL DEL DATASET")

spark.sql("""
    SELECT
        COUNT(*) as Total_Transacciones,
        COUNT(DISTINCT NIF) as Clientes_Unicos,
        COUNT(DISTINCT TARJETA) as Tarjetas_Unicas,
        COUNT(DISTINCT CATEGORIA_COMERCIO) as Categorias_Unicas,
        ROUND(SUM(IMPORTE_ABS), 2) as Suma_Total_Importes,
        ROUND(AVG(IMPORTE_ABS), 2) as Importe_Promedio,
        MIN(FECHA) as Fecha_Inicial,
        MAX(FECHA) as Fecha_Final
    FROM transacciones
""").show(truncate=False)
See performance (1)
RESUMEN GENERAL DEL DATASET
+------------------+--------------+--------------+----------------+------------------+---------------+-------------+-----------+
|Total_Transacciones|Clientes_Unicos|Tarjetas_Unicas|Categorias_Unicas|Suma_Total_Importes|Importe_Promedio|Fecha_Inicial|Fecha_Final|
+------------------+--------------+--------------+----------------+------------------+---------------+-------------+-----------+
|64072             |64            |148           |372             |1.427460393E7     |222.79         |2003-01-01   |2012-01-01 |
+------------------+--------------+--------------+----------------+------------------+---------------+-------------+-----------+
```

Análisis del uso de Cajeros Automáticos

Utiliza Spark SQL con LIKE '%CAJERO%' para filtrar transacciones de retiro de efectivo en cajeros automáticos, agrupadas por RANGO_HORARIO calculando número de retiros y monto total retirado. Las transacciones de los “Cajeros” son particularmente importantes porque el efectivo es mucho más difícil de rastrear que las compras con tarjeta.

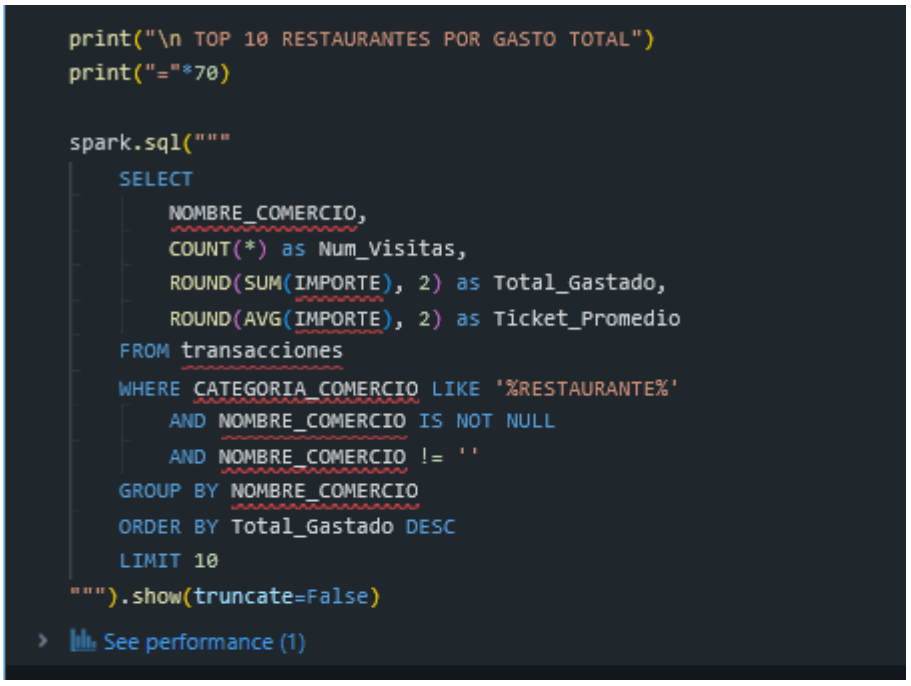
```
print(" ANÁLISIS DE USO DE CAJEROS")

spark.sql("""
    SELECT
        RANGO_HORARIO,
        COUNT(*) as Total_Retiros,
        ROUND(SUM(IMPORTE_ABS), 2) as Monto_Total_Retirado,
        ROUND(AVG(IMPORTE_ABS), 2) as Promedio_Por_Retiro
    FROM transacciones
    WHERE COD_OPE_DESC LIKE '%CAJERO%'
    GROUP BY RANGO_HORARIO
    ORDER BY Total_Retiros DESC
""").show()
```

```
 ANÁLISIS DE USO DE CAJEROS
+-------------+-------------+--------------------+------------------+
|RANGO_HORARIO|Total_Retiros|Monto_Total_Retirado|Promedio_Por_Retiro|
+-------------+-------------+--------------------+------------------+
|        Noche|          464|            242317.2|             522.24|
|        Tarde|          348|           156914.24|              450.9|
|      Mañana|          179|             79950.0|             446.65|
|    Madrugada|           15|              7810.0|             520.67|
+-------------+-------------+--------------------+------------------+
```

Analisis 10 restaurantes donde más se gastó

Podemos ver los gastos dentro de restaurantes, la cantidad de visitas y el promedio de estos. Filtrando los comercios con CATEGORIA_COMERCIO LIKE '%RESTAURANTE%' excluyendo registros sin nombre de comercio, agrupa por establecimiento calculando visitas, total gastado, y ticket promedio, ordenado descendentemente por gasto total limitado a top 10.

```
print("\n TOP 10 RESTAURANTES POR GASTO TOTAL")
print("="*70)

spark.sql("""
    SELECT
        NOMBRE_COMERCIO,
        COUNT(*) as Num_Visitas,
        ROUND(SUM(IMPORTE), 2) as Total_Gastado,
        ROUND(AVG(IMPORTE), 2) as Ticket_Promedio
    FROM transacciones
    WHERE CATEGORIA_COMERCIO LIKE '%RESTAURANTE%'
        AND NOMBRE_COMERCIO IS NOT NULL
        AND NOMBRE_COMERCIO != ''
    GROUP BY NOMBRE_COMERCIO
    ORDER BY Total_Gastado DESC
    LIMIT 10
""").show(truncate=False)
```



```
 TOP 10 RESTAURANTES POR GASTO TOTAL
======================================================================
+----------------------+-----------+-------------+---------------+
|NOMBRE_COMERCIO       |Num_Visitas|Total_Gastado|Ticket_Promedio|
+----------------------+-----------+-------------+---------------+
|COMERCIO DESCONOCIDO  |555        |63894.69     |115.13         |
|RESTAURANTE CANTOBLANCO|49        |45345.61     |925.42         |
|RESTAURANTE ERROTA ZAR|179        |37418.04     |209.04         |
|REST EL ESPIGON       |219        |27162.52     |124.03         |
|ADOC GOURMET, S.A.    |89         |26671.37     |299.68         |
|REST. MOA#A           |129        |24051.52     |186.45         |
|CASA MUNDI            |149        |22740.83     |152.62         |
|EL BALCON             |38         |20260.0      |533.16         |
|REST AINOHA           |96         |18323.39     |190.87         |
|HOTEL R. CURRITO      |91         |17958.88     |197.35         |
+----------------------+-----------+-------------+---------------+
```

Gastos por día de la semana

Podemos ver los gastos de cada día de la semana y analizar en qué día de la semana solían gastar más. Usa CASE WHEN para convertir DIA_SEMANA numérico (1-7) en nombres legibles en español ('Lunes', 'Martes', etc.), filtra solo gastos, agrupa por día calculando transacciones totales, gasto total, y promedio, ordenado cronológicamente.

```
Just now (4s)

# Gastos por día de la semana

print("\n PATRONES DE GASTO POR DÍA DE LA SEMANA")

spark.sql("""
    SELECT
        CASE DIA_SEMANA
            WHEN 1 THEN 'Lunes'
            WHEN 2 THEN 'Martes'
            WHEN 3 THEN 'Miércoles'
            WHEN 4 THEN 'Jueves'
            WHEN 5 THEN 'Viernes'
            WHEN 6 THEN 'Sábado'
            WHEN 7 THEN 'Domingo'
            ELSE NOMBRE_DIA
        END AS DIA_ES,
        COUNT(*) as Total_Transacciones,
        ROUND(SUM(CASE WHEN TIPO_TRANSACCION = 'GASTO' THEN IMPORTE ELSE 0 END), 2) as Total_Gastado,
        ROUND(AVG(CASE WHEN TIPO_TRANSACCION = 'GASTO' THEN IMPORTE END), 2) as Gasto_Promedio
    FROM transacciones
    GROUP BY DIA_ES, DIA_SEMANA
    ORDER BY DIA_SEMANA
""").show()

See performance (1)
```

```
 PATRONES DE GASTO POR DÍA DE LA SEMANA
+---------+------------------+-------------+-------------+
|   DIA_ES|Total_Transacciones|Total_Gastado|Gasto_Promedio|
+---------+------------------+-------------+-------------+
|    Lunes|              5862|    640401.75|       112.71|
|   Martes|              9364|   1010443.05|       122.66|
|Miércoles|             10757|    1148972.8|       113.27|
|   Jueves|              9801|   1055328.67|       114.77|
|  Viernes|              9402|   1054423.14|       119.08|
|   Sábado|             10096|   1140356.59|       121.11|
|  Domingo|              8790|   1034774.84|       122.05|
+---------+------------------+-------------+-------------+
```

Gastos mayores a 500€

Obtenemos todos los gastos mayores a 500 € y los ordenamos por el importe de manera descendente. Las transacciones mayores de 500 € incluyen importes muy elevados —hasta 11,930 €— .Se repiten especialmente en categorías como agencias de viajes y grandes almacenes, mostrando un patrón de gastos de alto valor.

```
## Gastos de alto monto

print("\n GASTOS DE ALTO MONTO (>500€)")
print("-"*70)

spark.sql("""
    SELECT
        FECHA,
        COD_OPE_DESC,
        NOMBRE_COMERCIO,
        CATEGORIA_COMERCIO,
        ROUND(IMPORTE, 2) as Importe
    FROM transacciones
    WHERE IMPORTE > 500 AND TIPO_TRANSACCION = 'GASTO'
    ORDER BY IMPORTE DESC
    LIMIT 20
```

```
 GASTOS DE ALTO MONTO (>500€)
----------------------------------------------------------------------
+----------+--------------------------+--------------------+-----------------------------------+-------+
|FECHA     |COD_OPE_DESC              |NOMBRE_COMERCIO     |CATEGORIA_COMERCIO                 |Importe|
+----------+--------------------------+--------------------+-----------------------------------+-------+
|2007-11-26|DISPOSICION EFECTIVO OFICINA|COMERCIO DESCONOCIDO|AGENCIAS BANCARIAS(ANTICIPO VENTANILLA)|11930.0|
|2011-10-05|COMPRA                    |EL CORTE INGLES     |HIPERCOR SUPERMERCADOS EL CORTE INGLES |8000.0 |
|2010-12-14|COMPRA                    |EL CORTE INGLES     |HIPERCOR SUPERMERCADOS EL CORTE INGLES |5500.0 |
|2011-02-15|COMPRA                    |ICE TRAVEL SPAIN, S.L.|V.DIST.VIAJES Y TRANSPORTE DE VIAJEROS|5500.0 |
|2008-12-27|COMPRA                    |VIAJES ECI          |EL CORTE INGLES                    |5500.0 |
|2008-12-29|COMPRA                    |VIAJES ECI          |EL CORTE INGLES                    |5500.0 |
|2008-12-30|COMPRA                    |VIAJES ECI          |EL CORTE INGLES                    |5266.03|
|2009-12-28|COMPRA                    |VIAJES ECI          |EL CORTE INGLES                    |5000.0 |
|2009-12-26|COMPRA                    |VIAJES ECI          |EL CORTE INGLES                    |5000.0 |
|2008-12-16|COMPRA                    |VIAJES ECI          |EL CORTE INGLES                    |5000.0 |
|2009-12-19|COMPRA                    |VIAJES ECI          |EL CORTE INGLES                    |5000.0 |
|2010-12-21|COMPRA                    |EL CORTE INGLES     |HIPERCOR SUPERMERCADOS EL CORTE INGLES |4985.0 |
|2008-07-26|COMPRA                    |VIAJES N L S L      |AGENCIAS DE VIAJES                 |4986.0 |
|2011-03-02|COMPRA                    |VIAJES PERFECT DAY  |AGENCIAS DE VIAJES                 |4800.06|
|2008-04-08|COMPRA                    |AEROMEXICO          |LINEAS AEREAS                      |4708.41|
```

DELTA LAKE

Guardamos el Data Frame como archivo Delta

Guardamos los datos limpios en formato Delta dentro del volumen de Unity Catalog, generando archivos Parquet con transacciones ACID y sobrescribiendo la ubicación especificada.

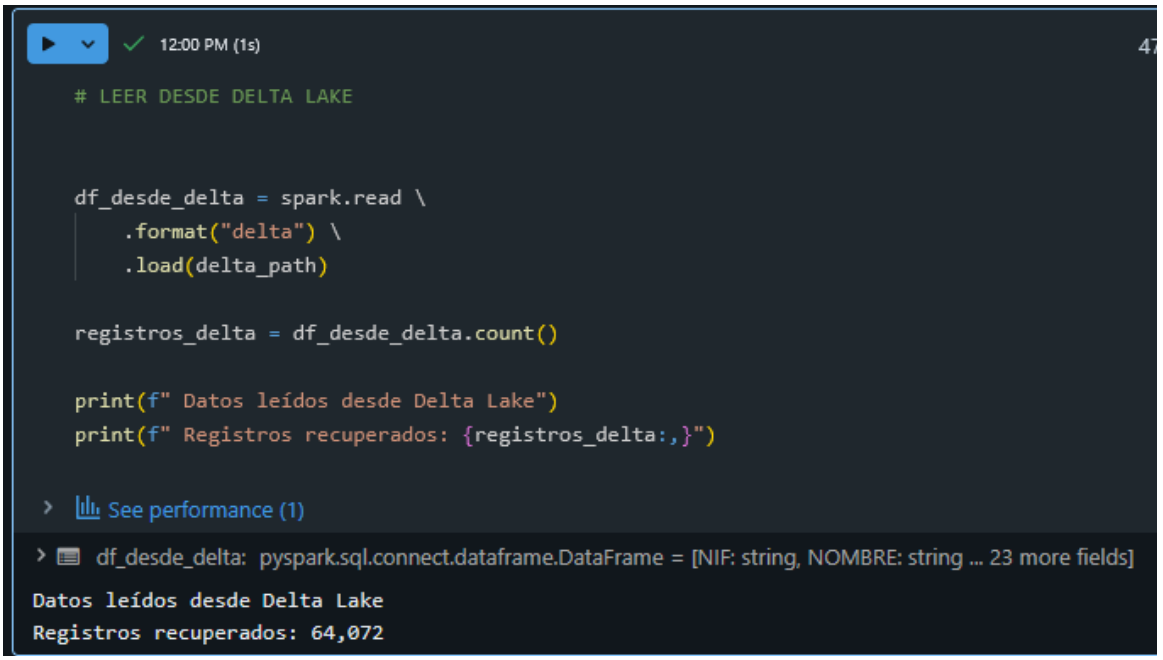
```
print("GUARDANDO DATOS EN FORMATO DELTA")

# Ruta donde guardaremos los archivos Delta
delta_path = "/Volumes/workspace/transactions/transactions_volume/transacciones_delta"

# Guardar como archivos Delta
df_limpio.write.format("delta") \
    .mode("overwrite") \
    .option("overwriteSchema", "true") \
    .save(delta_path)

print(f"Archivos Delta guardados en: {delta_path}")
print(f"Registros guardados: {df_limpio.count():,}")
print(f"Columnas guardadas: {len(df_limpio.columns)}")

# Verificar los archivos creados
print("\ncARCHIVOS CREADOS:")
dbutils.fs.ls(delta_path)
```

See performance (3)

```
GUARDANDO DATOS EN FORMATO DELTA
Archivos Delta guardados en: /Volumes/workspace/transactions/transactions_volume/transacciones_delta
Registros guardados: 64,072
Columnas guardadas: 25

cARCHIVOS CREADOS:
[FileInfo(path='dbfs:/Volumes/workspace/transactions/transactions_volume/transacciones_delta/_delta_log/', name='_delta_log/', size=0, modificationTime=1763910018272),
 FileInfo(path='dbfs:/Volumes/workspace/transactions/transactions_volume/transacciones_delta/part-00000-2a81c88d-6e2d-4ee5-b973-592051d1681c.c000.snappy.parquet', name='part-00000-2a81c88
d-6e2d-4ee5-b973-592051d1681c.c000.snappy.parquet', size=2992564, modificationTime=1763910017000),
 FileInfo(path='dbfs:/Volumes/workspace/transactions/transactions_volume/transacciones_delta/part-00000-65c93aac-5aff-4998-96de-73c34c8d4bb4.c000.snappy.parquet', name='part-00000-65c93aa
c-5aff-4998-96de-73c34c8d4bb4.c000.snappy.parquet', size=2992564, modificationTime=1763909745000)]
```

Leemos el Data Frame como archivo Delta

Luego creamos una tabla Delta que apunta a estos archivos usando spark.read \
dandonos la lectura del archivo y devolviendonos el conteo de los registros recuperados.



12:00 PM (1s)

```
# LEER DESDE DELTA LAKE


df_desde_delta = spark.read \
    .format("delta") \
    .load(delta_path)

registros_delta = df_desde_delta.count()

print(f" Datos leídos desde Delta Lake")
print(f" Registros recuperados: {registros_delta:,}")
```

See performance (1)

df_desde_delta: pyspark.sql.connect.dataframe.DataFrame = [NIF: string, NOMBRE: string ... 23 more fields]

```
Datos leídos desde Delta Lake
Registros recuperados: 64,072
```

Validamos Integridad

Se valida la integridad del archivo Delta comparándolo con el dataset, verificando que el conteo de registros, los valores clave y la estructura coincidan exactamente. Este proceso garantiza que la conversión a formato Delta no haya introducido pérdidas ni alteraciones en los datos originales

```
12:00 PM (<1s)                                                    48

# VALIDAR INTEGRIDAD

print(f" Registros originales (df_limpio):  {registros_finales:,}")
print(f" Registros desde Delta Lake:        {registros_delta:,}")
print(f" Columnas originales:                {len(df_limpio.columns)}")
print(f" Columnas desde Delta:               {len(df_desde_delta.columns)}")

if registros_finales == registros_delta and len(df_limpio.columns) == len(df_desde_delta.columns):
    print("\n INTEGRIDAD VALIDADA: Los datos son consistentes")
else:
    print("\n ADVERTENCIA: Hay diferencias en los datos")
```

```
Registros originales (df_limpio):  64,072
Registros desde Delta Lake:        64,072
Columnas originales:                25
Columnas desde Delta:               25

INTEGRIDAD VALIDADA: Los datos son consistentes
```

Muestreo de Datos

Mostramos 10 filas de datos para hacer una última verificación de la integridad.

```
12:00 PM (1s)                                                     49

# Muestra de datos desde Delta

print("\n MUESTRA DE DATOS DESDE DELTA LAKE:")
df_desde_delta.select(
    "FECHA", "COD_OPE_DESC", "IMPORTE", "CATEGORIA_COMERCIO", "RANGO_HORARIO"
).show(10, truncate=False)

> See performance (1)
```

```
 MUESTRA DE DATOS DESDE DELTA LAKE:
+----------+----------------------------------------------+-----------------+---------------------------+-------------+
|FECHA     |COD_OPE_DESC                                  |IMPORTE          |CATEGORIA_COMERCIO         |RANGO_HORARIO|
+----------+----------------------------------------------+-----------------+---------------------------+-------------+
|2004-11-03|COMPRA                                        |4.650000095367432|AUTOPISTAS (TERMINALES)    |Tarde        |
|2006-12-01|COMPRA                                        |5.400000095367432|GARAJES Y APARCAMIENTOS    |Tarde        |
|2011-09-28|COMPRA                                        |68.4000015258789 |C.A.M.P.S.A.               |Mañana       |
|2004-03-06|COMPRA                                        |91.77999877929688|VIPS                       |Madrugada    |
|2005-11-11|COMPRA                                        |73.7300033569336 |GASOLINERAS                |Noche        |
|2009-09-22|REINTEGRO EN CAJERO PROPIO                    |600.0            |CAJEROS AUTOMATICOS        |Noche        |
|2009-04-27|COMPRA                                        |595.0            |CONFECCION TEXTIL EN GENERAL|Tarde       |
|2009-11-01|CARGO POR FACTURACION DE TARJETAS DE CREDITO  |-2710.8798828125 |SIN CATEGORIA              |Madrugada    |
|2007-05-14|COMPRA                                        |40.0             |C.A.M.P.S.A.               |Mañana       |
|2006-07-19|REINTEGRO EN CAJERO PROPIO                    |600.0            |CAJEROS AUTOMATICOS        |Noche        |
+----------+----------------------------------------------+-----------------+---------------------------+-------------+
only showing top 10 rows
```

TODAS LAS VISUALIZACIONES UTILIZAN LA TABLA CREADA A PARTIR DE LA RECUPERACIÓN DE LOS ARCHIVOS DELTA

VISUALIZACIONES

Gráfico 1: Evolución Temporal Por Mes

Muestra dos gráficos complementarios: un gráfico de barras horizontales representando la cantidad de transacciones por mes (eje izquierdo) y un gráfico de líneas mostrando el importe total gastado por mes (eje derecho), cubriendo el período completo de 108 meses (2003-2011). Permite identificar visualmente la tendencia ascendente de gasto desde ~40K€/mes hasta picos de ~140K€/mes a través del tiempo.

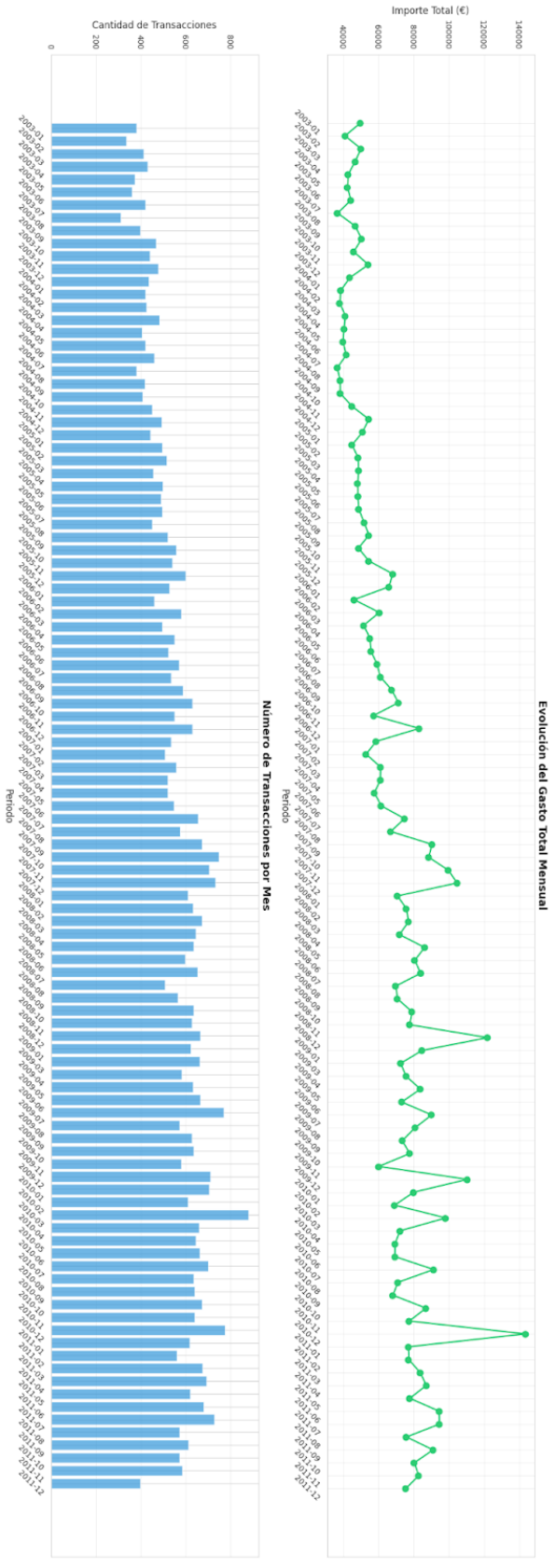


Gráfico 2: Top Categorías

Gráfico de barras horizontales mostrando las 12 categorías de comercio más frecuentadas ordenadas descendentemente por número de transacciones, con valores absolutos etiquetados al final de cada barra.

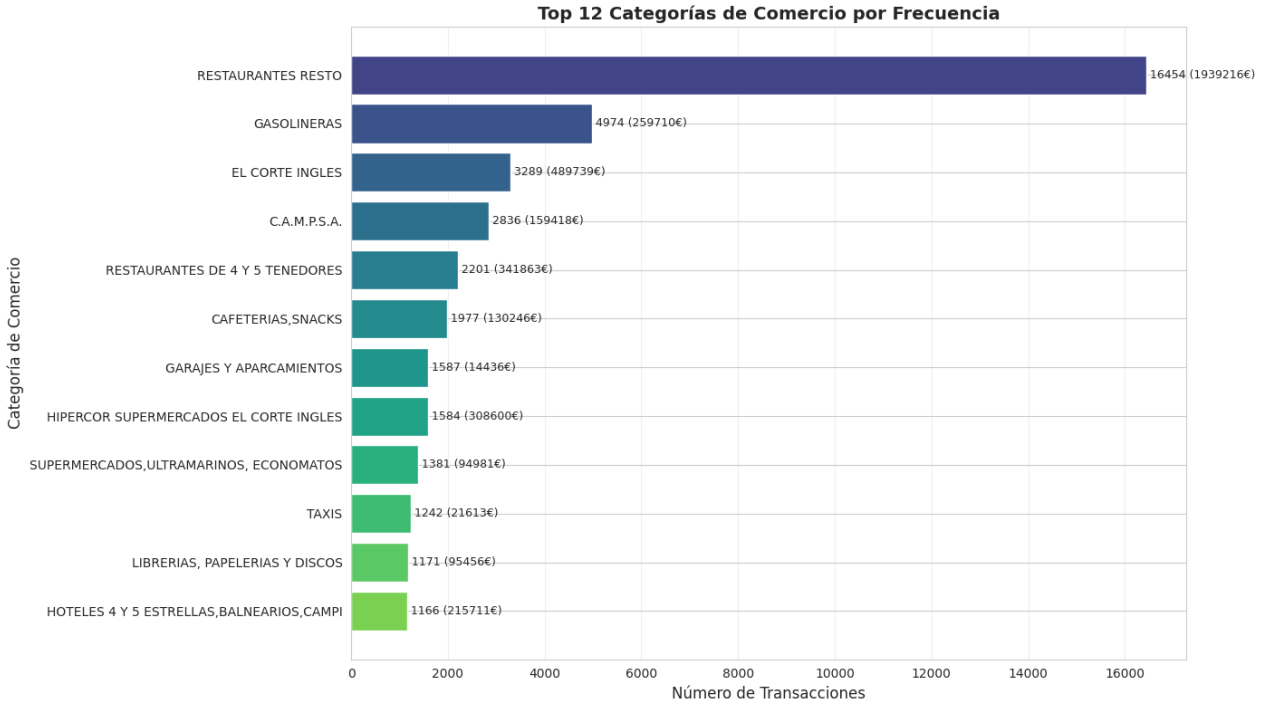


Gráfico 3: Distribución por Tipo de Operación

Gráfico circular (pie chart) mostrando la distribución porcentual, dejando ver que la operación *COMPRA* concentra el 92.1% de todas las transacciones (58,953), seguida a gran distancia por *Cargo por facturación de tarjetas de crédito* con 0.06% (3,665 transacciones) y *Reintegro en cajero propio* con 1.56% (999), mientras que el resto de categorías representan proporciones muy reducidas.

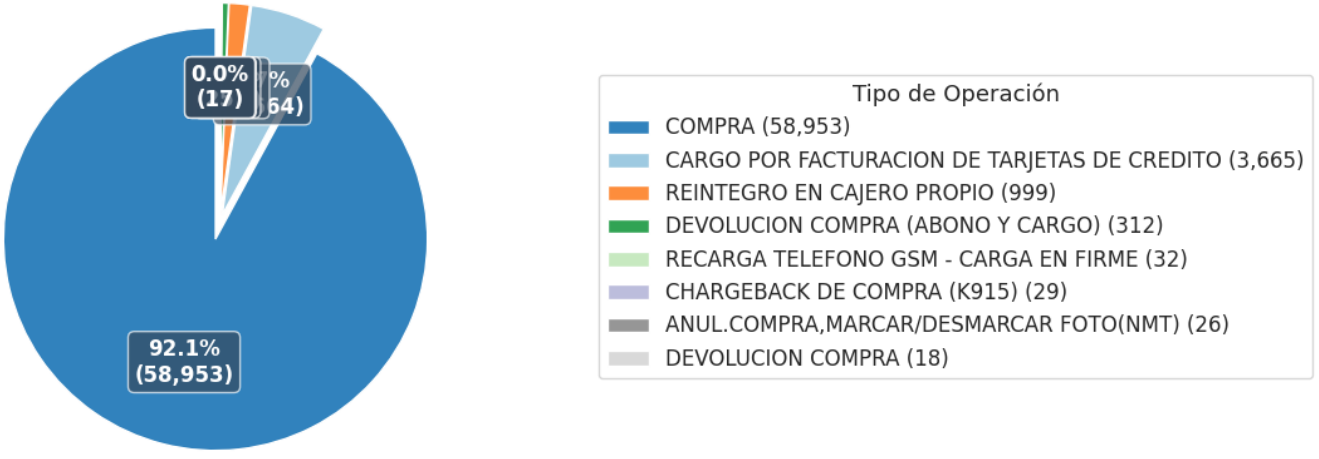
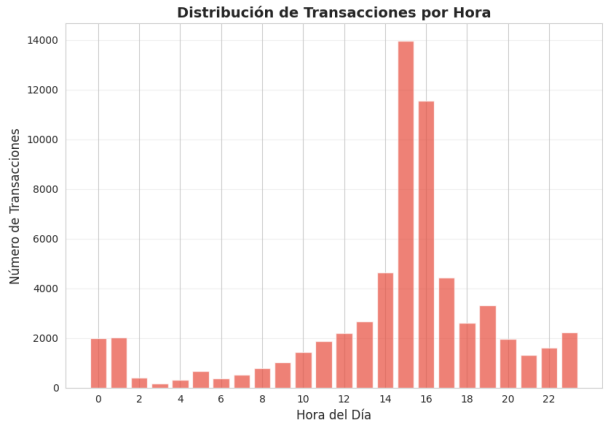


Gráfico 4: Patrones Horarios

Dos gráficos complementarios: un histograma de barras mostrando distribución de transacciones por hora del día (0-23h) revelando picos entre las 13-15h (horario de comida) con ~13,000 transacciones, y un gráfico circular mostrando distribución por rango horario con 'Tarde' dominando 61.3% (39,417), 'Noche' 20.4% (13,049), 'Madrugada' 8.7% (5,569), y 'Mañana' 9.4% (6,037).



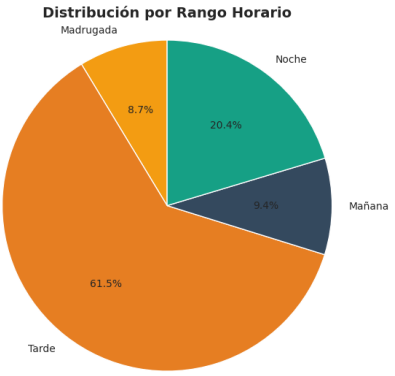
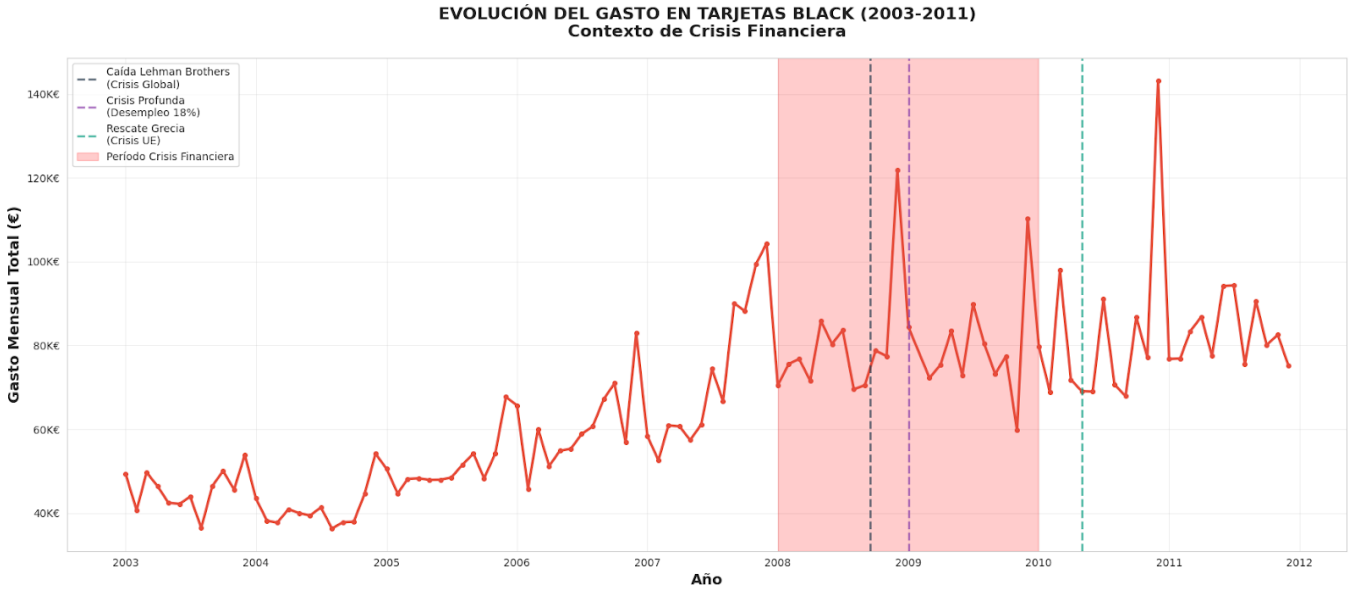


Gráfico 5: Evolución de los gastos en contexto Crisis Financiera

Gráfico de línea temporal (2003-2012) mostrando gasto mensual total con área sombreada en rojo marcando el período de crisis financiera (Sep 2008 - Dic 2009), líneas verticales punteadas indicando eventos clave: inicio de crisis (Lehman Brothers, Sep 2008), peor momento (Mar 2009), y rescate de Caja Madrid (Jun 2010). Durante la zona roja de crisis, el gasto NO disminuyó sino que alcanzó uno de sus niveles más altos (~120,000€/mes), con picos extremos de volatilidad demostrando que mientras el país estaba en crisis los directivos de Caja Madrid continuaron gastando dinero público sin reducción alguna.



REFERENCIAS Y RECURSOS UTILIZADOS

Dataset y Contexto Histórico

- Dataset Github: https://github.com/rafadelascuevas/limpieza-analisis-basico/tree/master/datasets/hoja_calculo_tarjetas_black/03_csv_para_limpiar
- Audiencia Nacional. (2017). Sentencia del Caso Tarjetas Black - Procedimiento Abreviado
- 63/2013. Tribunal Penal, Sección Segunda. Madrid, España.
- El País. (2014-2017). Especial: El Caso de las Tarjetas Black: https://elpais.com/especiales/2014/tarjetas-opacas-caja-madrid/

Documentación Técnica Consultada

- Databricks Documentation: https://docs.databricks.com/
- Apache Spark Documentation: https://spark.apache.org/docs/latest/
- Delta Lake Documentation: https://docs.delta.io/
- PySpark API Reference: https://spark.apache.org/docs/latest/api/python/
- Databricks Free Edition Setup: https://docs.databricks.com/getting-started/free-edition.html
- Spark SQL Guide: https://spark.apache.org/docs/latest/sql-programming-guide.html
- Delta Lake Quickstart: https://docs.delta.io/latest/quick-start.html