SIMULADOR DE INVERSIÓN DOMÉSICA Y MERCADOS FINANCIEROS

Un enfoque Big Data para la planificación Domésica con AWS y PowerBI mediante la proyección de metas financieras alcanzables y simulación de MonteCarlo en los mercados S&P500 y Bitcoin

Autor: Pablo Vidal Vidal
Reto 5 Big Data & IA
pablo2vbnddaw@gmail.com

Índice del Proyecto

1. INTRODUCCIÓN ..... [Pág. 2]

1.1. Problema a Resolver y Solución Propuesta

1.2. Tecnologías Utilizadas y Diagrama de Arquitectura

2. PASO 1: PREPARACIÓN DEL ENTORNO Y DATOS EN S3 ..... [Pág. 3]

2.1. Creación de Bucket y Estructura de Carpetas

2.2. Generación y Carga de Datos (Extracto, S&P 500, Bitcoin)

2.3. Resumen del Paso 1

3. PASO 2: AWS GLUE - ETL DEL EXTRACTO BANCARIO ..... [Pág. 7]

3.1. Objetivo y Acciones (Crawler, Job, Transformaciones SQL)

3.2. Salidas: Detalle Mensual, Promedios de Gasto, Aporte para Simulación

3.3. Resumen del Paso 2

4. PASO 3: AWS GLUE - ETL DE DATOS DE MERCADO ..... [Pág. 14]

4.1. Objetivo y Acciones (Crawlers, Job, Transformaciones SQL)

4.2. Salida: Parámetros de Mercado (μ y σ para S&P 500 y Bitcoin)

4.3. Resumen del Paso 3

5. PASO 4: SIMULACIÓN MONTE CARLO CON AWS EMR ..... [Pág. 21]

5.1. Objetivo y Configuración del Entorno EMR (Clúster, Studio, Notebook)

5.2. Desarrollo en Notebook PySpark:

5.2.1. Configuración y Carga de Datos (Celda 1)

5.2.2. Ejecución de Simulaciones (Celda 2)

5.2.3. Análisis y Almacenamiento de Resultados (Celda 3)

6. PASO 5: COMPROBACIÓN DE RESULTADOS EN S3 ..... [Pág. 28]

(Archivos: Distribución, Proyección Mediana, Sumario)

7. PASO 6: VISUALIZACIÓN CON POWER BI ..... [Pág. 29]

7.1. Conexión, Importación y Transformación de Datos

7.2. Diseño del Dashboard (KPIs, Gráficos de Simulación e Históricos)

8. CONCLUSIÓN DEL PROYECTO ..... [Pág. 32]

8.1. Recorrido por la Solución, Valor y Aprendizaje

8.2. Agradecimientos y Cierre

9. APÉNDICES ..... [Pág. 33]

9.1. Scripts Python de Preparación de Ingesta

9.2. Scripts PySpark del Notebook EMR

9.3. Scripts SQL de Transformación en AWS Glue

🚀 INTRODUCCIÓN 🎯

- 💡 PROBLEMA A RESOLVER:

  Muchas personas tienen metas financieras 🏡💰✈️ (comprar una casa, jubilación, un viaje) pero les cuesta visualizar si sus hábitos de ahorro e inversión actuales les permitirán alcanzarlas, especialmente considerando la volatilidad de los mercados 📉📈.

- ✨ SOLUCIÓN:

  Un simulador innovador que:

  - 🔍 Analiza el comportamiento de ahorro del usuario a partir de un extracto bancario.
  - 🎯 Permite al usuario definir una meta financiera, un plazo y una estrategia de inversión (distribuyendo su ahorro mensual entre efectivo 💵, S&P 500 📊 y Bitcoin ₿).
  - 🎰 Utiliza datos históricos de mercado para simular miles de posibles escenarios futuros mediante la potente técnica de Monte Carlo.
  - 🧮 Calcula la probabilidad de alcanzar la meta financiera definida.
  - 📊 Presenta los resultados de forma clara, visual e interactiva en Microsoft Power BI.

- ⚙️ TECNOLOGÍAS UTILIZADAS:

  - ☁️ Almacenamiento de Datos: Amazon S3
  - 🛠️ ETL (Extracción, Transformación, Carga): AWS Glue
  - 💻 Procesamiento y Simulación: Amazon EMR (con Apache Spark)
  - 📈 Visualización: Microsoft Power BI

- 🔗 DIAGRAMA DE ARQUITECTURA:

  (Se incluirá un diagrama de flujo visualizando ➡ ☁️ S3 -> 🛠️ Glue -> ☁️ S3 -> ▯ EMR -> ☁️ S3 -> 📊 Power BI, detallando los tipos de datos en cada transición).

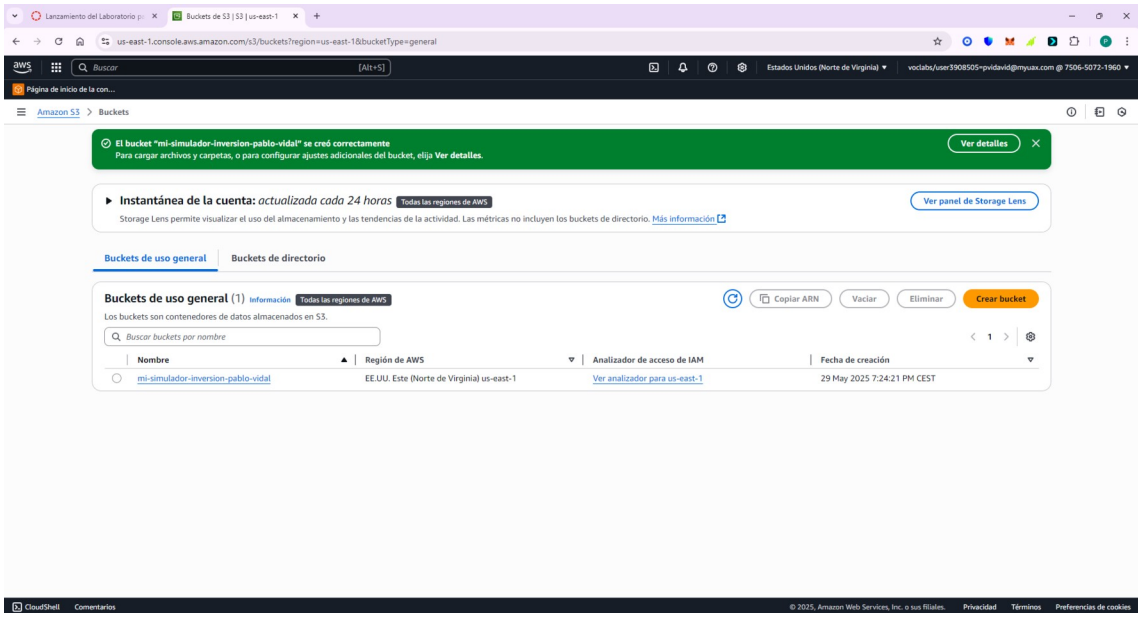
❓ PASO 1: CARGA DE DATOS INICIALES EN S3 Y CREACIÓN DEL ENTORNO ☁️📂

🎯 OBJETIVO DEL PASO:

Crear la estructura de carpetas en Amazon S3 y subir los datasets iniciales (extracto bancario del usuario y datos históricos de mercado).

- 🛠️ ACCIONES DETALLADAS:

1. Crear un Bucket S3 (🖼️ Imagen1.jpg):
    - Nombre del bucket: mi-simulador-inversion-pablo-vidal

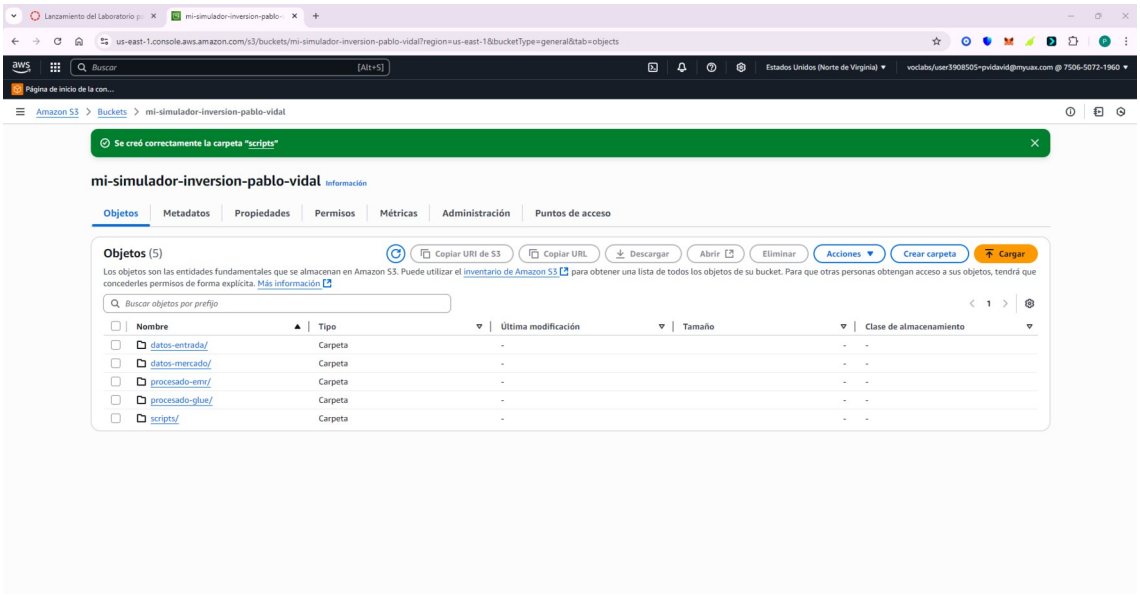


2. Crear Estructura de Carpetas dentro del Bucket (🖼️ imagen2.jpg):

- Creamos las siguientes carpetas (en S3, las "carpetas" son prefijos de objeto):
    - 📁 datos-entrada/
    - 📊 datos-mercado/
    - ⚙️ procesado-glue/ (Los datos procesados tienen subcarpetas):
        - 📄 detalle_movimientos_mensual/: Desglose mensual detallado de ingresos y gastos.

- promedios_gastos_categorias/: Promedios mensuales por categoría de gasto.
- aporte_simulacion_base/: Un solo valor con el aporte mensual promedio para la inversión (ej. promedio de "Transferencia Ahorro").
- parametros_mercado/: Los valores μ y σ para S&P 500 y Bitcoin.

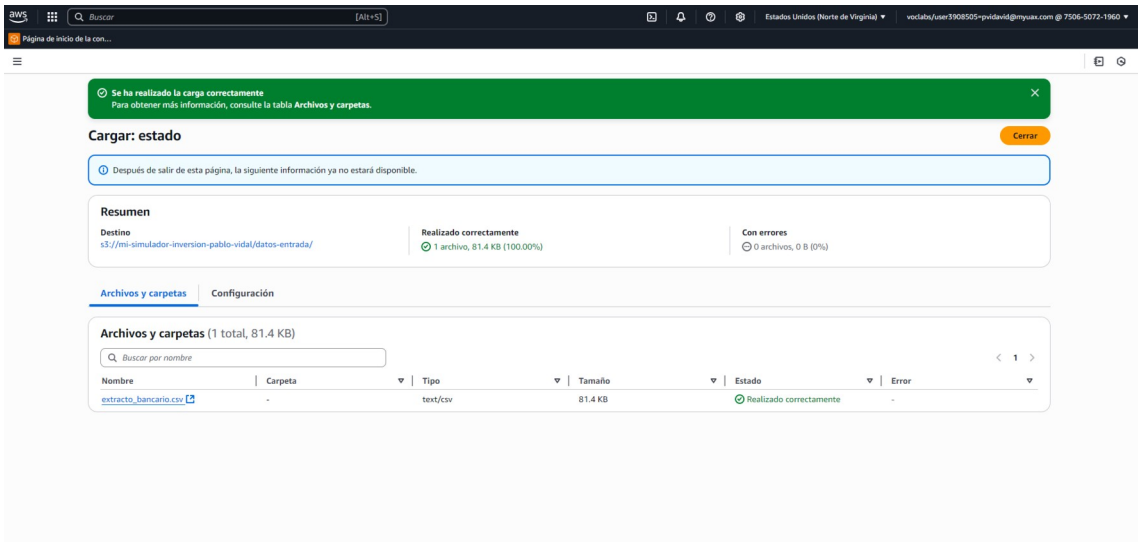
- procesado-emr/
- scripts/ (Opcional, pero bueno para guardar los scripts).



3. Preparar los Datos Locales:
   - Extracto Bancario (extracto_bancario.csv) (imagen3.jpg):
   - Creamos o pedimos al banco un CSV (extracto_bancario.csv). Columnas importantes: Fecha, Descripcion, Importe.(*Yo lo he creado con código Python "generarextractobancario.py" y lo he guardado en la carpeta scripts/ de S3.)*
   - Conceptos bancarios a utilizar: Nómina, Supermercado, alquiler-hipoteca, restauración, transferencia-ahorro, gastos-varios, vacaciones, ingreso-extraordinario.
4. Datos Históricos S&P 500 (sp500_historico.csv):

- Fuente: Yahoo Finance (u otra fuente fiable). *He creado un script Python (generarcsvsp500.py), adjuntado, para llamar directamente a Yahoo Finance y crear el CSV de los últimos 10 años.*
- Descargamos el CSV (sp500_historico.csv). Columnas necesarias: Fecha y cierre. Renombrar si es necesario a Fecha y PrecioCierre.
- Ubicación en S3: datos-mercado/

5. Datos Históricos Bitcoin (bitcoin_historico.csv):
    - Fuente: Similar al S&P 500, ticker BTC-USD en Yahoo Finance. *He creado un script Python (generarcsvbitcoin.py), adjuntado, para llamar directamente a Yahoo Finance y crear el CSV de los últimos 10 años.*
    - Descargamos el CSV (bitcoin_historico.csv) mensual con Date y Adj Close. Renombrar si es necesario a Fecha y PrecioCierre.
    - Ubicación en S3: datos-mercado/



1. Subir los Archivos a S3 (Referencia general a 🖼️ imagen3.jpg para la apariencia de S3 con archivos):
    - ➡ Subimos extracto_bancario.csv a la carpeta s3://mi-simulador-inversion-pablo-vidal/datos-entrada/.

- ➡ Subimos sp500_historico.csv a la carpeta s3://mi-simulador-inversion-pablo-vidal/datos-mercado/.
- ➡ Subimos bitcoin_historico.csv a la carpeta s3://mi-simulador-inversion-pablo-vidal/datos-mercado/.

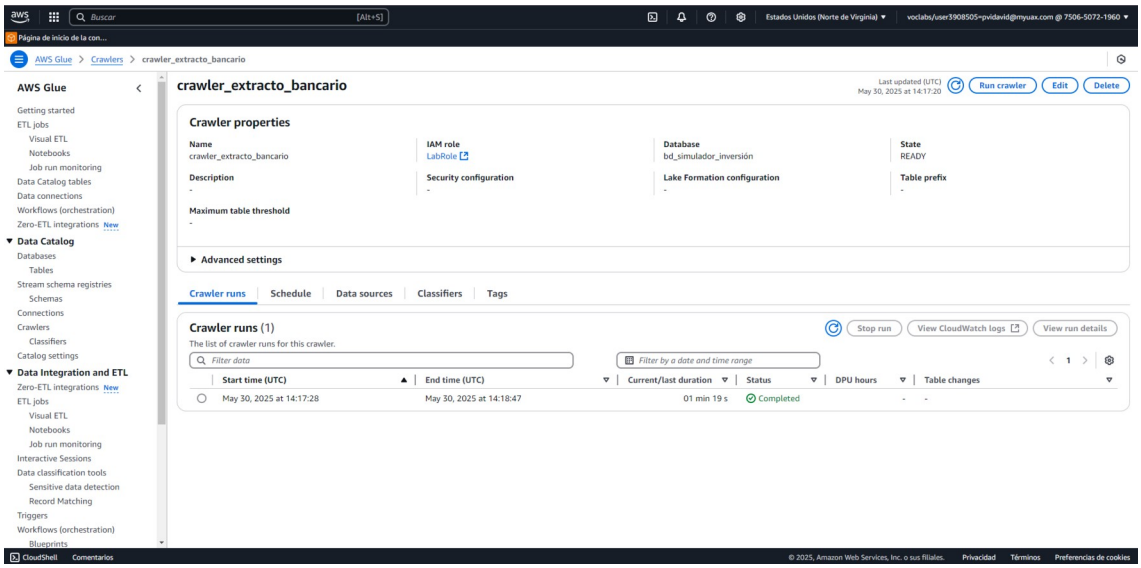
📝 RESUMEN PASO 1:

- ✅ Iniciamos creando un bucket en Amazon S3, que servirá como nuestro data lake centralizado para todos los datos del proyecto.
- ✅ Hemos organizado el bucket con una estructura de carpetas lógica para separar los datos de entrada crudos, los datos de mercado, y los resultados de las etapas de procesamiento posteriores (Glue y EMR).
- ✅ Se han preparado y cargado tres conjuntos de datos iniciales:
  1. Un extracto bancario (extracto_bancario.csv) generado por el usuario que contiene transacciones de ingresos y gastos.
  2. Datos históricos mensuales del índice S&P 500 (sp500_historico.csv) obtenidos de Yahoo Finance mediante un script Python.
  3. Datos históricos mensuales de Bitcoin (bitcoin_historico.csv) obtenidos de Yahoo Finance mediante un script Python.

  Estos archivos se almacenan en sus respectivas carpetas dentro de S3.
- ✅ Hemos guardado los dos scripts de Python para obtener información de Yahoo Finance en la carpeta scripts/ de S3.

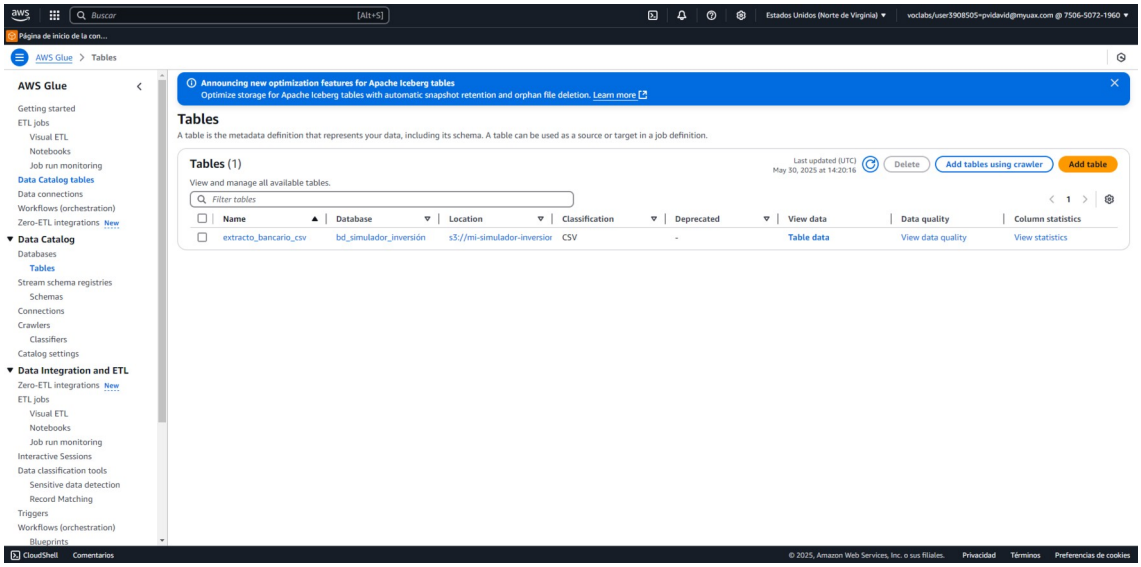
❓ PASO 2: AWS GLUE - ETL PARA CALCULAR EL AHORRO MENSUAL DEL USUARIO 🛠️⚙️

- 🎯 OBJETIVO PRINCIPAL DEL PASO:
  - ⚙️ Procesar el archivo extracto_bancario.csv.
  - 🏷️ Categorizar cada transacción de forma más granular.
  - ➕ Agregar los montos mensuales para cada categoría de ingreso y gasto.
  - 🧮 Calcular totales mensuales de ingresos, gastos operativos (excluyendo la transferencia de ahorro), el ahorro potencial (ingresos - gastos operativos) y el saldo neto final de la cuenta corriente.
  - 📄 Generar un CSV detallado con esta información mensual (detalle_movimientos_categorizados_mensual.csv).
  - 📄 Generar un CSV con los promedios mensuales para cada categoría de gasto (promedios_gastos_categorias.csv).
  - 💰 Generar un CSV con el "aporte mensual base para la simulación" (ej. promedio de "Transferencia Ahorro") que usará EMR (aporte_mensual_simulacion_base.csv).
- 🛠️ ACCIONES DETALLADAS:
  1. Creamos y arrancamos un Crawler de Glue para el Extracto Bancario (🖼️ Imagen4.jpg):
     - Nombre del crawler: crawler_extracto_bancario
     - Base de datos de salida del crawler: bd_simulador_inversion
     - ➡️ Acción: Arrancamos el crawler.



2. Verificamos la Tabla en el Data Catalog (`Imagen5.jpg`):

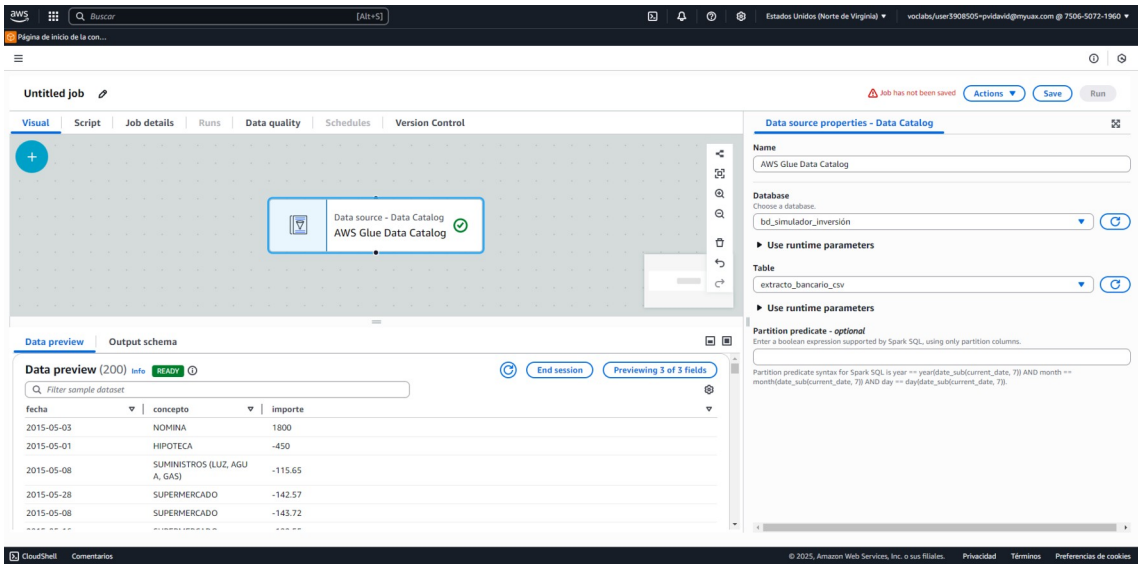
a. 🔍 Revisamos el esquema inferido. Nos aseguramos de que las columnas `Fecha`, `Concepto` e `Importe` estén presentes y con tipos de datos razonables (ej. fecha, texto, número).



3. Creamos un Job de AWS Glue Studio para Procesar el Extracto(`Imagen6.jpg` - Diseño del Job):

a. **Nodo Origen (Source):**

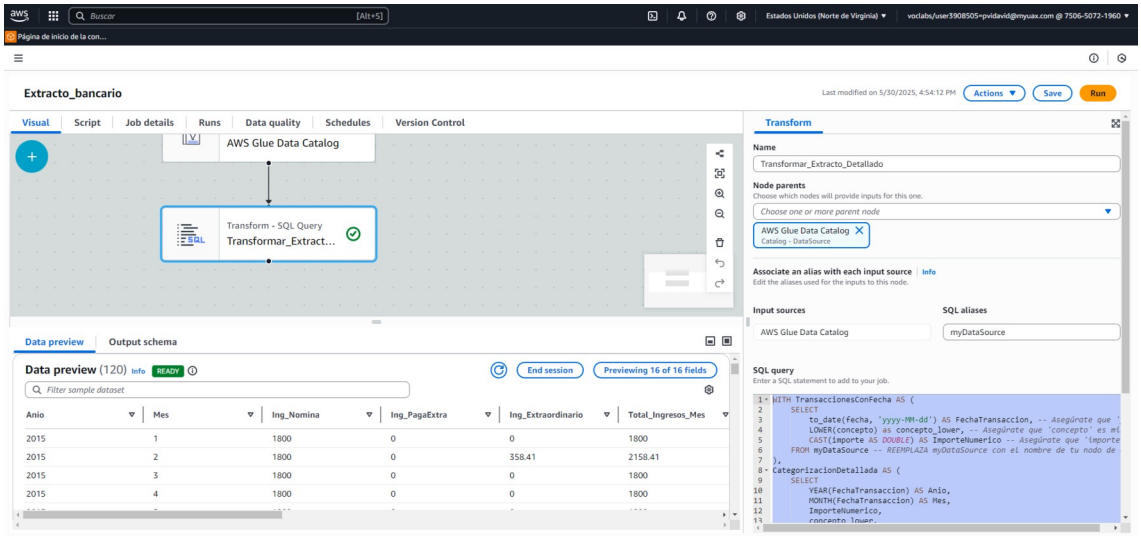
Fuente: `bd_simulador_inversion` y la tabla del extracto bancario.



b. **Nodo Transformación - SQL (imagen7.jpg)**

Nombre del nodo: Transformar_Extracto_detallado

En la pestaña "Transformar", introducimos el script SQL correspondiente.

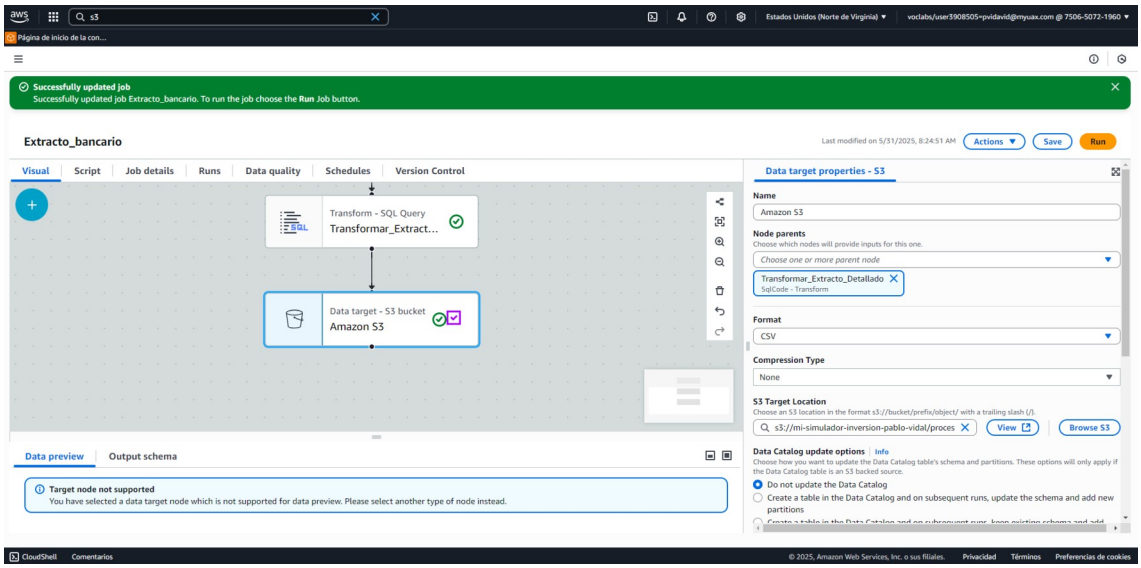


c. **Nodo Destino (Target) (`imagen8.jpg`)**

Nombre del nodo: `Detalle_Movimientos`

Acción: Vamos a guardar en `s3://mi-simulador-inversion-pablo-vidal/procesado-glue/detalle_movimientos_mensual/` el archivo `.csv` resultante con el nombre

`detalle_movimientos_categorizados_mensual.csv`.

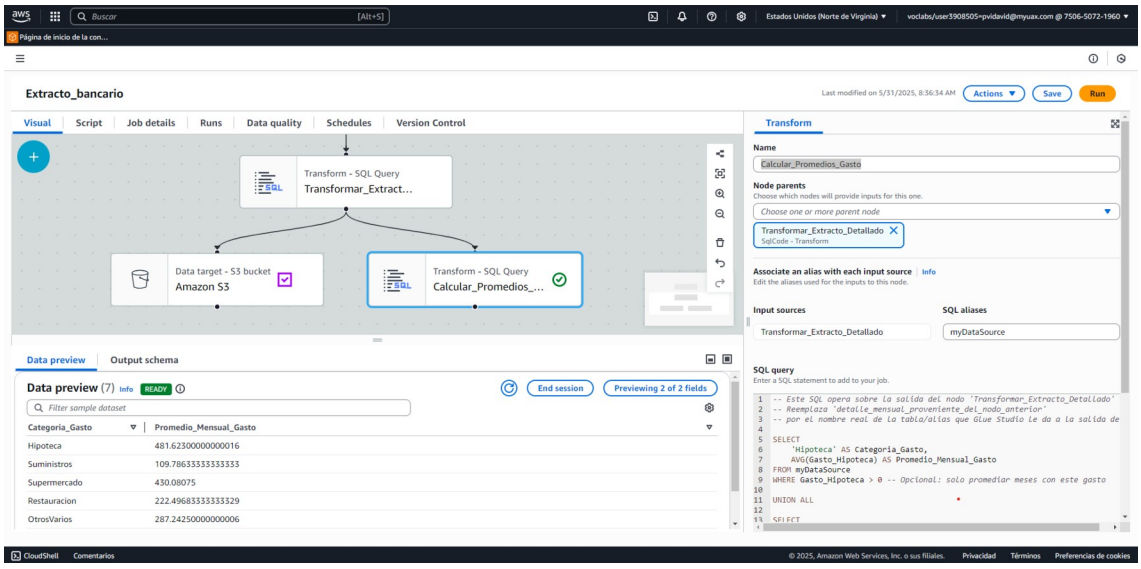


d. **Añadimos un Nuevo Nodo de Transformación SQL (🖼️ imagen9.jpg)**

Flujo: A partir del nodo Transformar_Extracto_Detallado creamos una nueva rama de flujo.

e. **Configurar el Nodo SQL para promedios_gastos_categorias**

Nombre del nodo: Calcular_Promedios_Gasto



f. **Volcamos la información en S3 (🖼️ imagen10.jpg)**

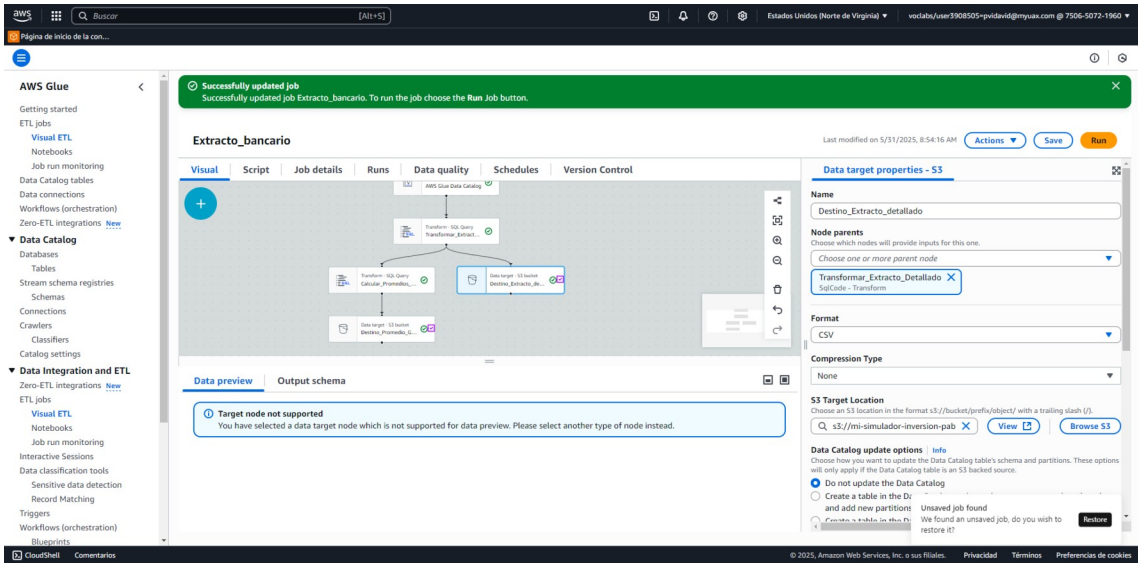
Nombre del nodo (Opcional): Destino_Promedios_Gasto_S3

Configuración:

Formato: "CSV"

Tipo de compresión: "Ninguna"

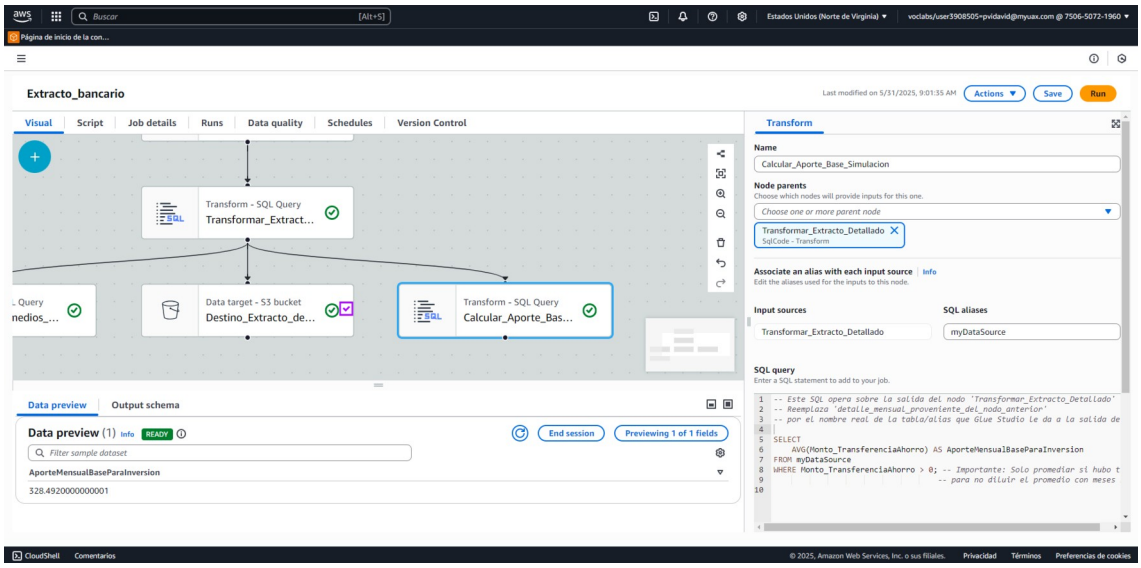
Ruta S3: s3://mi-simulador-inversion-pablo-vidal/procesado-glue/promedios_gastos_categorias/



g. **Añadimos un nuevo nodo transformador para Calcular y guardar el Aporte Mensual Base para la Simulación de EMR (🖼️ `imagen11.jpg`)**

Entrada: Nuevamente, este nodo tomará como entrada la salida del Nodo SQL 1 (`Transformar_Extracto_Detallado`).

Nombre del nodo: `Calcular_Aporte_Base_Simulacion`



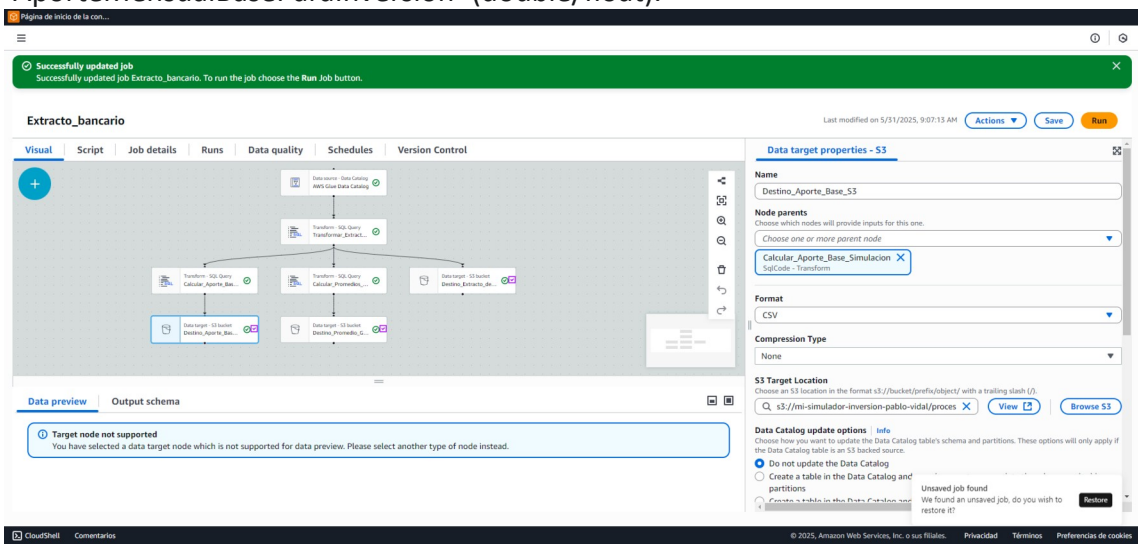
h. Añadir un Nodo Destino S3 para `aporte_mensual_simulacion_base` (`imagen12.jpg`)

Flujo: Desde la salida del nodo `Calcular_Aporte_Base_Simulacion`.

Nombre del nodo: `Destino_Aporte_Base_S3`

`s3://mi-simulador-inversion-pablo-vidal/procesado-glue/aporte_simulacion_base/`

Esquema de salida: Debería mostrar una única columna: `AporteMensualBaseParaInversion` (double/float).



📝 RESUMEN PASO 2: PROCESAMIENTO ETL CON AWS GLUE

(Formato sugerido en Word: Título 2 o ej. 14pt, Negrita)

- ✅ **Ingesta y Transformación Principal:** Se crea y ejecuta un job de AWS Glue que ingiere el archivo extracto_bancario.csv desde Amazon S3.

- ⚙️ **Procesamiento Detallado:** Un primer nodo de transformación SQL procesa los datos del extracto, realizando una categorización granular de ingresos y gastos. Calcula totales mensuales, el ahorro potencial bruto (ingresos menos gastos operativos) y el saldo neto resultante en la cuenta corriente. Este análisis mensual detallado se guarda en S3, en la carpeta detalle_movimientos_mensual/.
- 📊 **Análisis de Promedios:** Un segundo nodo de transformación SQL utiliza la salida detallada anterior para calcular los promedios mensuales para cada categoría principal de gasto. Estos promedios se almacenan en S3, dentro de la carpeta promedios_gastos_categorias/.
- 💰 **Cálculo del Aporte para Simulación:** Un tercer nodo de transformación SQL, también partiendo del detalle mensual, se encarga de calcular el "Aporte Mensual Base para la Simulación". Este valor (por ejemplo, el promedio de las transferencias destinadas a ahorro) es crucial para la siguiente etapa y se guarda en un archivo CSV en S3, en la carpeta aporte_simulacion_base/.
- 🖼️ **Evidencia Visual:** Se incluyen capturas de pantalla que muestran el diseño del job de ETL en AWS Glue Studio, los scripts SQL clave utilizados en las transformaciones, y ejemplos de los archivos CSV de salida generados y almacenados en Amazon S3.

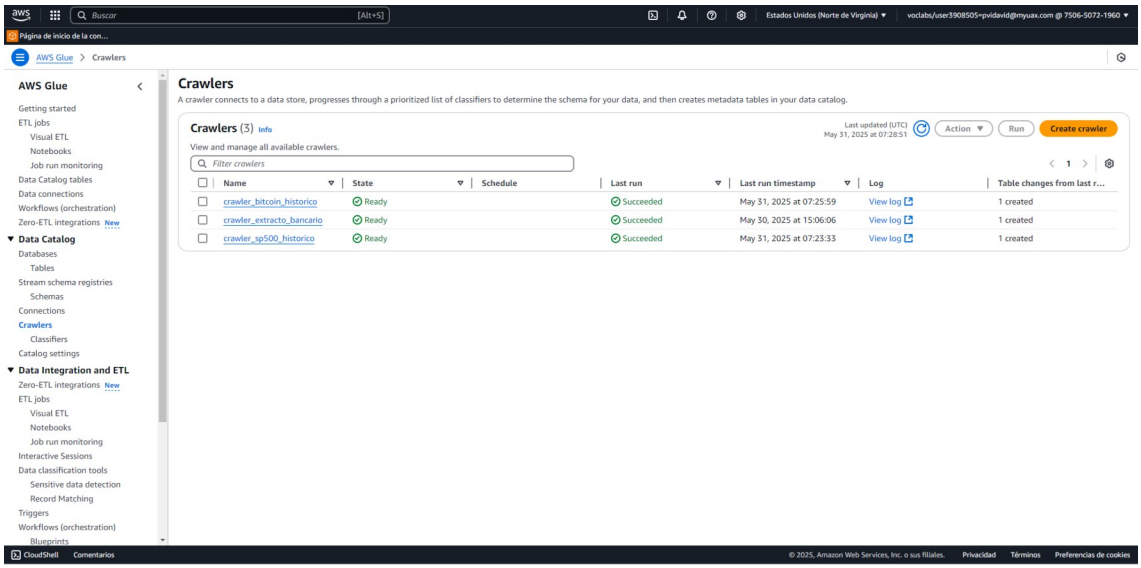
⁉️ PASO 3: AWS GLUE - ETL PARA DATOS DE MERCADO (S&P 500 Y BITCOIN) 📈⚙️

🎯 OBJETIVO DEL PASO:

Procesar los archivos CSV con los datos históricos del S&P 500 (sp500_historico.csv) y Bitcoin (bitcoin_historico.csv) para calcular los parámetros estadísticos clave que alimentarán nuestro modelo de simulación de Monte Carlo. Específicamente, necesitamos:
1. El retorno mensual promedio (μ) para cada activo.
2. La volatilidad mensual (σ) (desviación estándar de los retornos mensuales) para cada activo.
Estos cuatro valores (μ_sp500, σ_sp500, μ_bitcoin, σ_bitcoin) se guardarán en un archivo para ser utilizados por Amazon EMR.

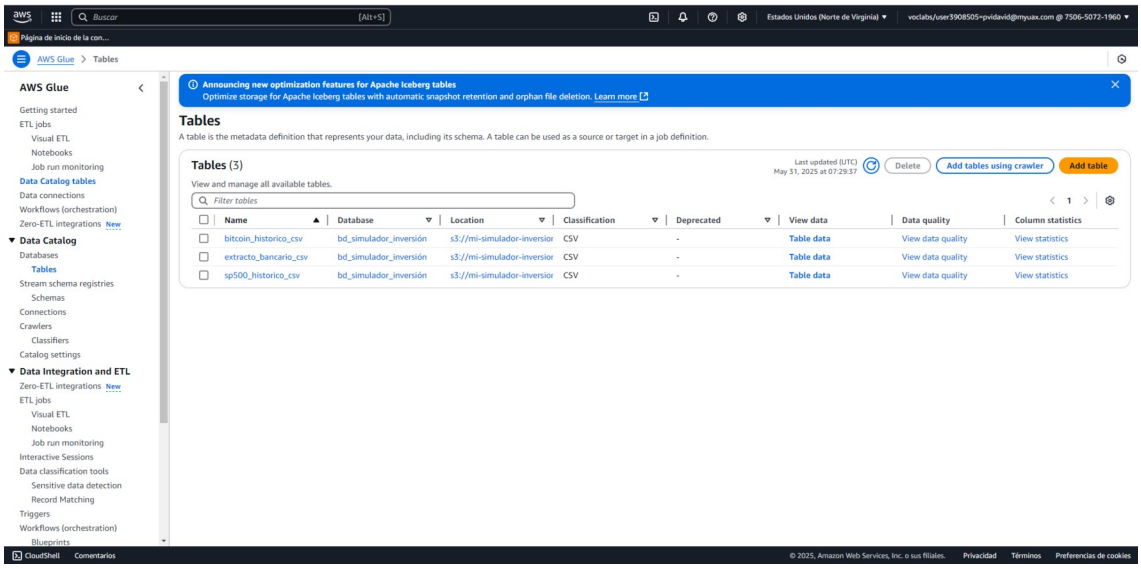
🛠️ ACCIONES DETALLADAS:

1. **Crear Crawlers de Glue para los Datos de Mercado (🖼️ `imagen13.jpg`)**
   a. **Crawler S&P 500:**
      Nombre del crawler (S&P 500): `crawler_sp500_historico`
      Origen de datos: S3, ruta `s3://mi-simulador-inversion-pablo-vidal/datos-mercado/sp500_historico.csv` (Corregido para apuntar al archivo CSV directamente para mayor precisión del crawler)
      Base de datos de salida: `bd_simulador_inversion` (la misma que antes).
   b. **Crawler Bitcoin:**
      Nombre del crawler (Bitcoin): `crawler_bitcoin_historico`
      Origen de datos: S3, ruta `s3://mi-simulador-inversion-pablo-vidal/datos-mercado/bitcoin_historico.csv` (Corregido para apuntar al archivo CSV y no al del S&P500)
      Base de datos de salida: `bd_simulador_inversion` (la misma que antes).



2. **Verificamos Tablas en el Data Catalog** 🔍

Confirmamos en el AWS Glue Data Catalog que las tablas `sp500_historico` y `bitcoin_historico` existen y que sus esquemas han sido inferidos correctamente, conteniendo principalmente las columnas `Fecha` y `PrecioCierre` con los tipos de datos adecuados.



3. **Crear un Nuevo Job de AWS Glue Studio para Procesar los Datos de Mercado(🏞️ `imagen15.jpg`)**

Nombre del Job: `job_procesar_datos_mercado`

Estrategia del Job: Este job implementará dos flujos de procesamiento paralelos (uno dedicado al S&P 500 y otro a Bitcoin). Las salidas de estos flujos se unirán posteriormente para calcular y guardar los parámetros estadísticos combinados en un único archivo.

a. **Flujo para S&P 500:**

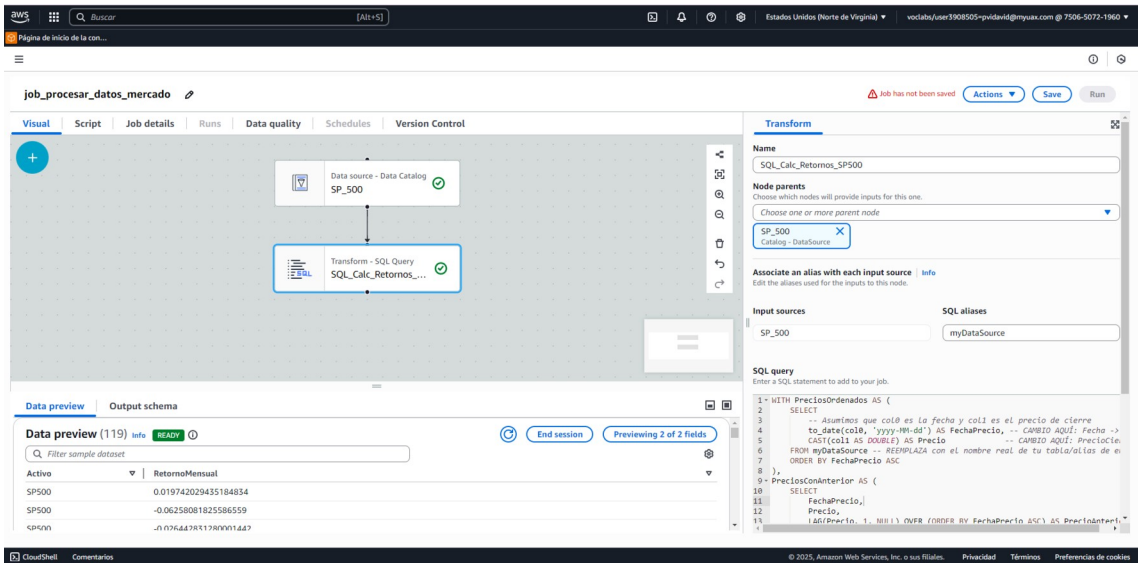
i. Nodo Origen 1 (SP500):

Fuente: AWS Glue Data Catalog, utilizando la base de datos `bd_simulador_inversion` y la tabla `sp500_historico`.

ii. Nodo Transformación SQL 1 (`SQL_Calc_Retornos_SP500`):

Entrada: Toma como entrada los datos del nodo `Source_SP500`.

(Propósito: Calcular los retornos mensuales para el S&P 500)



b. **Flujo para Bitcoin (`imagen16.jpg`):**

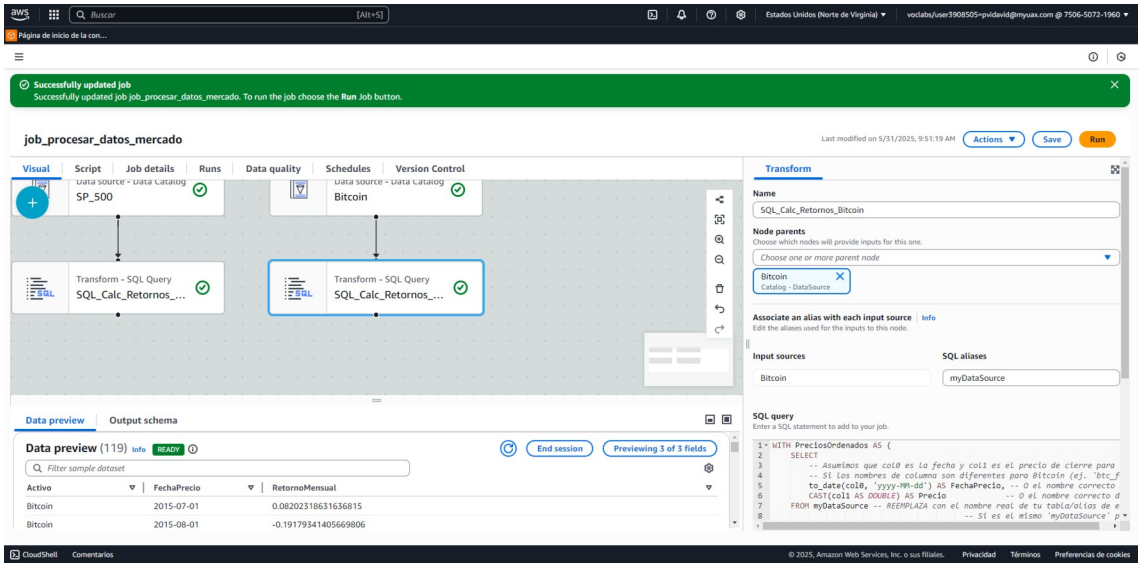
i. Nodo Origen 2 (Bitcoin):

Fuente: AWS Glue Data Catalog, utilizando la base de datos `bd_simulador_inversion` y la tabla `bitcoin_historico`. (Corregido el nombre de la tabla a `bitcoin_historico`)

ii. Nodo Transformación SQL 2 (`SQL_Calc_Retornos_Bitcoin`):

Entrada: Toma como entrada los datos del nodo `Source_Bitcoin`.

(Propósito: Calcular los retornos mensuales para Bitcoin).



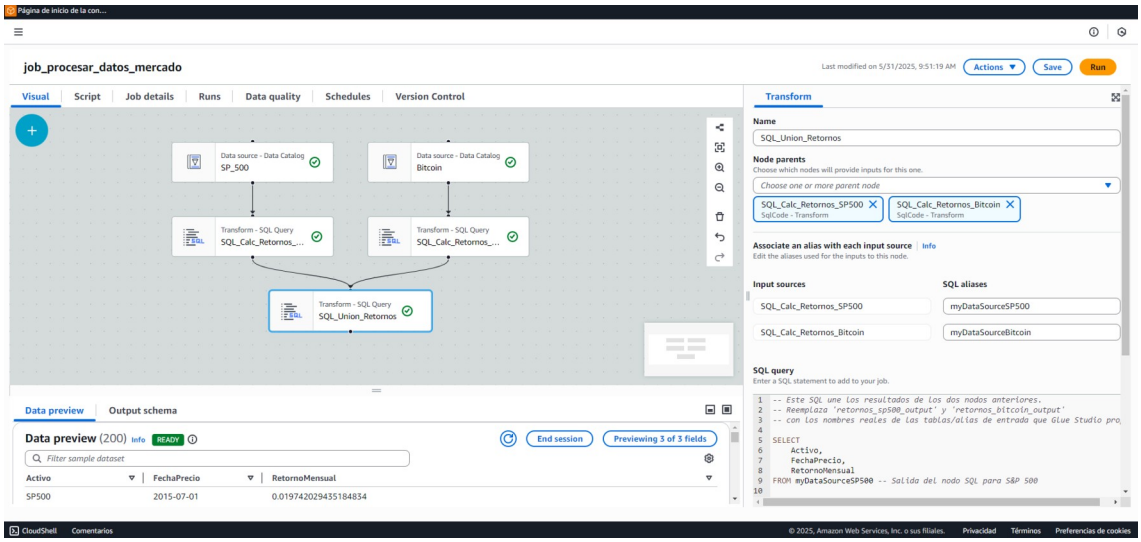
4. **Configurar el Nodo SQL para Unir los Retornos (🖼️ imagen17.jpg)**

Nombre del nodo: SQL_Union_Retornos

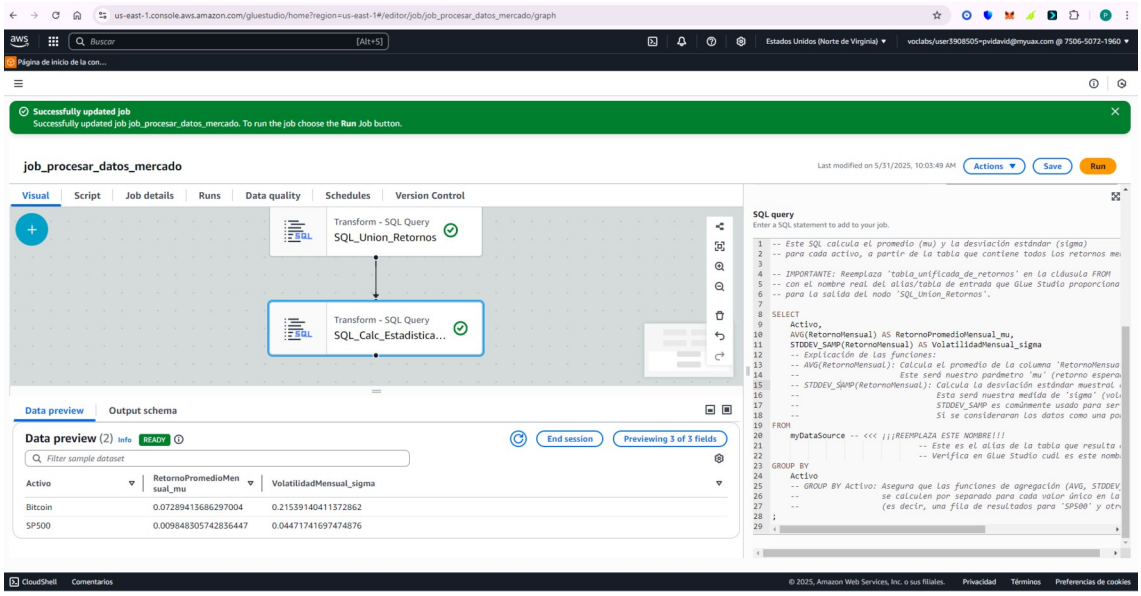
Entradas: Las salidas de los nodos SQL_Calc_Retornos_SP500 y SQL_Calc_Retornos_Bitcoin.

(Nota Importante: Es crucial identificar los nombres exactos o alias que AWS Glue Studio asigna a las tablas de salida de los dos nodos de transformación SQL anteriores, ya que estos serán los nombres de las tablas de entrada para este nodo de unión).

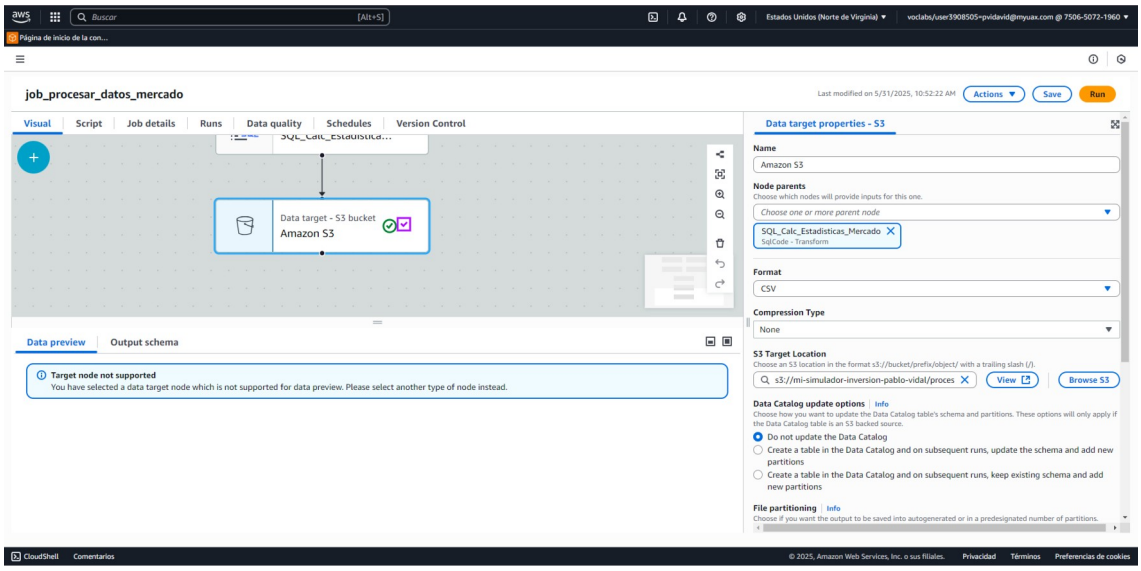
Operación: Los retornos de ambos activos se unen utilizando una consulta UNION ALL (como se detalla en el script SQL adjunto), no mediante un JOIN, para apilar los conjuntos de datos.



1. **Añadimos el Nodo Final de Transformación SQL(imagen18.jpg):**
    1. Nombre del nodo: SQL_Calc_Estadisticas_Mercado
    2. **Pestaña "sql_union_retornos_alias`**



1. **Guardamos en S3 (imagen19.jpg):**
    1. Nombre del nodo: Amazon S3



Resumen del Paso 3:

1. 🔍 Lectura de Datos:
   - •Dos Crawlers de AWS Glue 🕷️ se lanzaron a S3.
   - •Uno para sp500_historico.csv 📈 y otro para bitcoin_historico.csv ₿.
   - •Misión cumplida: ¡Registraron las tablas sp500_historico y bitcoin_historico en nuestro AWS Glue Data Catalog! 📖
2. ⚙️ Job ETL en Glue Studio (job_procesar_datos_mercado):
   - • 🧠 Cálculo de Retornos Mensuales:
     - •Para el S&P 500: Un nodo SQL 📐 tomó sus precios históricos y calculó sus subidas y bajadas mensuales (RetornoMensual).
     - •Para Bitcoin: ¡Otro nodo SQL hizo lo mismo con los precios de Bitcoin! 💸
     - •Salida de cada uno: Activo, FechaPrecio, RetornoMensual.
   - • 🤝 Unión de Retornos:
     - •Un nodo SQL UNION ALL juntó todos los retornos (S&P 500 y Bitcoin) en una sola gran tabla. ¡Todos juntos ahora!
   - • 📊 Cálculo de Estadísticas (μ y σ):
     - •El gran final: Un nodo SQL 🤓 procesó la tabla unida.

•Calculó el Retorno Mensual
Promedio (AVG como media_retorno_mensual o μ) para CADA activo.

•Calculó la Volatilidad
Mensual (STDDEV_SAMP como volatilidad_mensual o σ) para CADA activo.

3. 💾 Salida a S3:

•Los cuatro parámetros estrella ✨ (μ_sp500, σ_sp500, μ_bitcoin, σ_bitcoin) se guardaron bien ordenaditos.

•Destino: Un archivo CSV (ej. parametros_mercado.csv) en la carpeta s3://mi-simulador-inversion-pablo-vidal/procesado-glue/parametros_mercado/.

• 🔑 Resultado Clave: Ahora tenemos en S3 un archivo con los parámetros estadísticos promedio de retorno y volatilidad para el S&P 500 y Bitcoin. ¡Listos para alimentar nuestro motor de simulación en EMR! 🚀

🎰 Paso 4: Simulación de Monte Carlo con AWS EMR Notebooks 🚀

• 🎯 Objetivo: ¡Poner a trabajar toda nuestra preparación! Usaremos la potencia de Amazon EMR y Apache Spark para:

1.Lanzar miles de "futuros posibles" 🎲 (simulación de Monte Carlo) para ver cómo podría crecer el dinero del usuario.

2.Alimentar la simulación con:

•El aporte mensual que el usuario puede invertir (calculado por Glue y guardado en s3://.../procesado-glue/ahorro_promedio_usuario/).

•Los parámetros mágicos del mercado (μ y σ para S&P 500 y Bitcoin, desde s3://.../procesado-glue/parametros_mercado/).

•La estrategia de inversión del usuario (% a S&P 500, % a Bitcoin, % a efectivo).

•La meta financiera 💰 y el plazo ⏳ del usuario.

3.Obtener un abanico de posibles capitales finales después de todas esas simulaciones.

4.Calcular la gran pregunta: ¿Qué probabilidad (%) tiene el usuario de alcanzar su meta? 🌟

5.Guardar los jugosos resultados en S3, listos para que Power BI los haga brillar ✨.

• 🛠️ Herramientas AWS: Amazon EMR (con Apache Spark, ejecutado desde un EMR Notebook).

•Resumen del Proceso (en el EMR Notebook 📓):

1. 🚀 Configuración Inicial:

•Conectarse a Spark.

•Cargar los datos necesarios desde S3:

•El archivo ahorro_promedio_usuario.csv (para saber cuánto se invierte cada mes).

•El archivo parametros_mercado.csv (con las μ y σ del S&P 500 y Bitcoin).

•Definir los inputs del usuario:

•meta_financiera = XXXXX €

•plazo_meses = XX

•porcentaje_sp500 = 0.XX

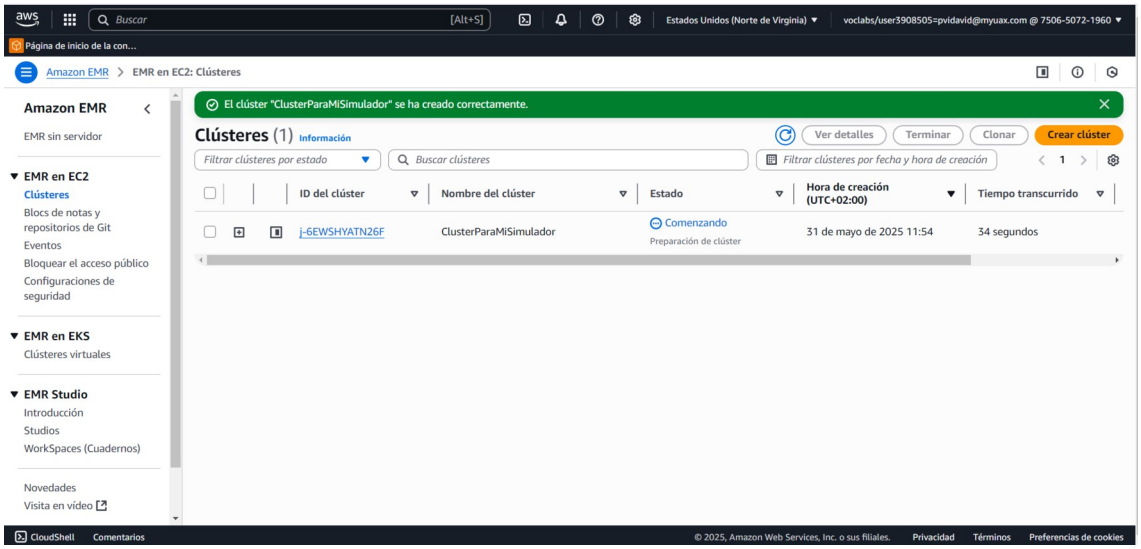
•porcentaje_bitcoin = 0.XX

- porcentaje_cash = 0.XX (lo que queda)
- numero_simulaciones = 10000 (o el número que elijas)

2. 🎲 El Corazón de la Simulación (Bucle de Monte Carlo):
   - Se repite numero_simulaciones veces:
     - Para cada simulación, se empieza con capital = 0.
     - Se simula mes a mes durante el plazo_meses:
       - Se calcula cuánto se invierte este mes en cada activo (S&P 500, Bitcoin, cash) según los porcentajes.
       - Se genera un retorno aleatorio para el S&P 500 (usando su μ y σ) 📈.
       - Se genera un retorno aleatorio para Bitcoin (usando su μ y σ) ₿.
       - Se actualiza el valor de la inversión en S&P 500 y Bitcoin, y se suma el aporte al cash.
     - Al final de los meses, se guarda el capital total final de ESTA simulación.
3. 📊 Análisis de Resultados:
   - Una vez tenemos los numero_simulaciones capitales finales:
     - Se calcula la probabilidad de éxito: (Número de simulaciones donde Capital Final >= Meta Financiera) / Total de Simulaciones.
     - Se identifica la simulación mediana (la que queda justo en medio si ordenamos todos los resultados) para tener una proyección "típica" del crecimiento.
4. 💾 Guardado de Resultados en S3 (para Power BI):
   - resultados_simulacion_distribucion.csv: Contiene los 10.000 capitales finales.
   - resultados_simulacion_proyeccion_mediana.csv: Detalla el crecimiento mes a mes (cash, S&P 500, Bitcoin) de la simulación mediana.
   - sumario_simulacion.json (o .csv): Un resumen con los inputs de la simulación y la probabilidad de éxito calculada.
   - Destino:
     Carpeta s3://mi-simulador-inversion-pablo-vidal/procesado-emr/.

- 🔑 Resultado Clave: ¡Archivos listos en S3 con toda la información que Power BI necesita para contar la historia financiera del usuario! 📉📈✅

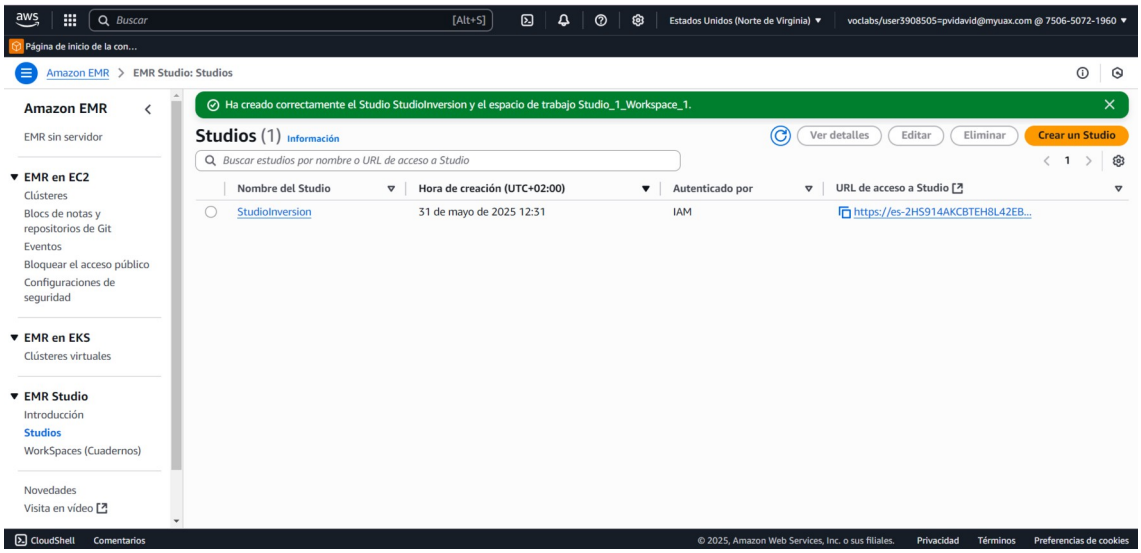
1 Vamos a crear un clúster Clusterparamisimulador (imagen20.jpg)



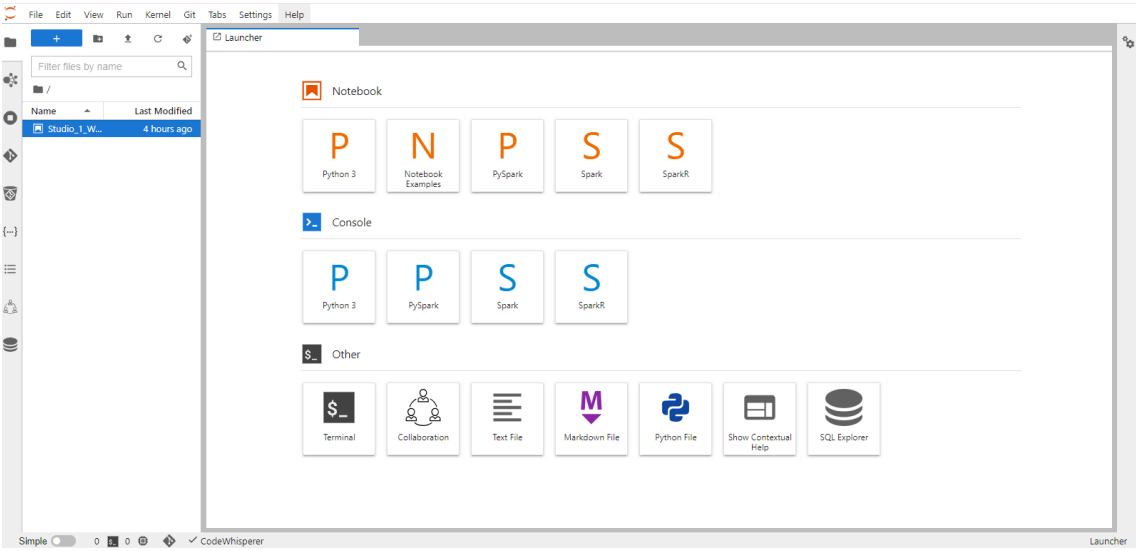
Ojo a los parámetros de creación del clúster. Las máquinas permitidas son m4.large y los roles son diferentes hasta el momento. Necesitas JupiterEnterpriseGAtegway y Jupiterhub, así como Spark 3.5.4

2 **Creamos un studio y un Workspace para ejecutar notebook de jupiter y los scripts**

Cuando creemos el studio (ha de ser personalizado) creamos también el workspace



3.**Asociamos el clúster al workspace (tiene que estar inactivo), lanzamos el notebook pyspark y empezamos a programar (imagen22.jpg)**

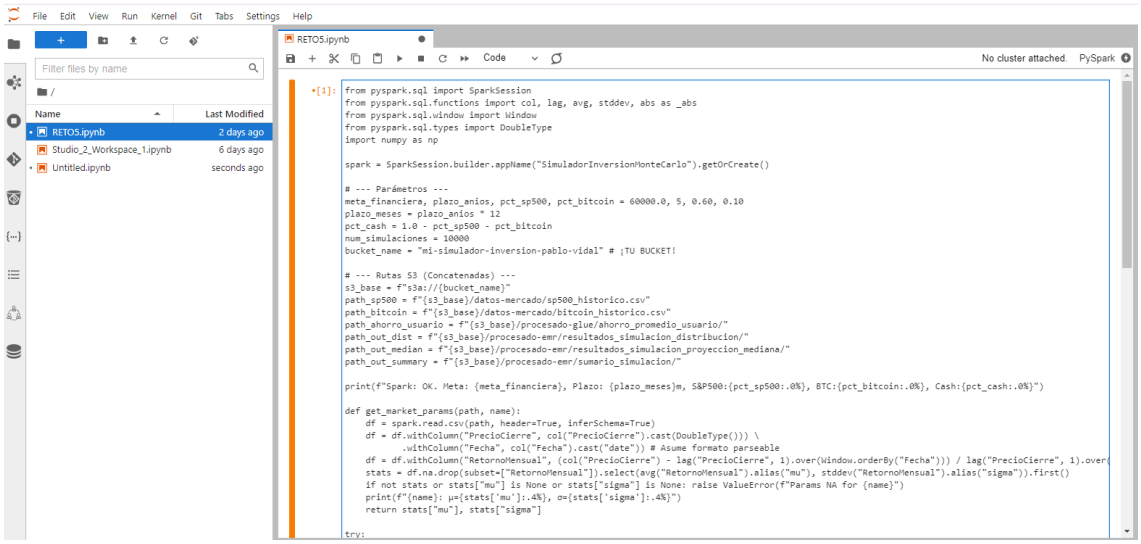


Paso 4.1 (EMR Notebook): ¡Encendiendo Motores y Cargando Munición! 🚀🔧

• 🎯 Objetivo: Poner a punto nuestro entorno de Spark en el EMR Notebook y cargar toda la "materia prima" que necesitamos para la simulación:

1. Arrancar la sesión de Spark ✨.
2. Definir las reglas del juego: meta financiera 🥅, plazo 🗓️, estrategia de inversión (%, %, %) y cuántas veces vamos a simular 🔁.
3. Leer los datos históricos del S&P 500 📈 y Bitcoin ₿ desde S3.
4. Calcular *in situ* (o cargar si ya los tenemos de Glue) sus motores de comportamiento: el retorno mensual promedio (μ) y la volatilidad (σ) para cada uno.
5. Cargar el dato crucial: el ahorro mensual promedio del usuario 💸 (¡gracias al trabajo previo de Glue!).

• 📸 (imagen23.jpg): Una captura del inicio del Notebook mostrando las celdas de configuración de Spark, la definición de los parámetros de entrada (meta, plazo, etc.) y las funciones o código para cargar y procesar los datos de mercado y el ahorro del usuario.



Paso 4.2 (EMR Notebook): ¡Lanzando los Dados del Destino Financiero! 💸🎰

• 🎯 Objetivo: ¡Aquí ocurre la magia! Ejecutar miles de escenarios de inversión simulados para ver cómo podría evolucionar el capital del usuario. Queremos:

1.Simular, en paralelo (gracias a Spark 💪), miles de "vidas financieras" diferentes.

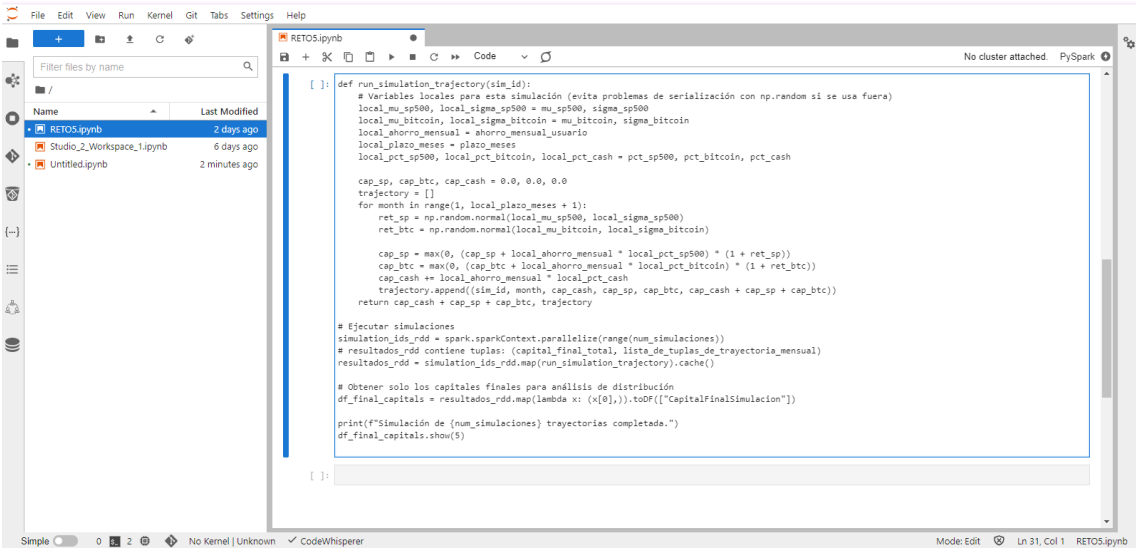
2.Para cada "vida", rastrear mes a mes cómo crece (¡o decrece!) el dinero invertido en S&P 500 📈, Bitcoin ₿ y lo que queda en efectivo 💵.

3.Obtener el capital total final para cada una de estas miles de simulaciones.

4.Guardar también la trayectoria mensual de cada componente de la cartera para, al menos, una simulación representativa (la mediana, por ejemplo).

• 📸 (imagen24.jpg): Una captura del Notebook mostrando la función principal de simulación (simular_una_trayectoria o similar) que proyecta el capital mes a mes, y la sección de código donde se invoca esta función repetidamente (por ejemplo, usando spark.sparkContext.parallelize o un bucle que genera un RDD/DataFrame de resultados) para las numero_simulaciones iteraciones. Podría mostrarse también una pequeña muestra de los resultados parciales.

- **Código PySpark (Celda 2)**

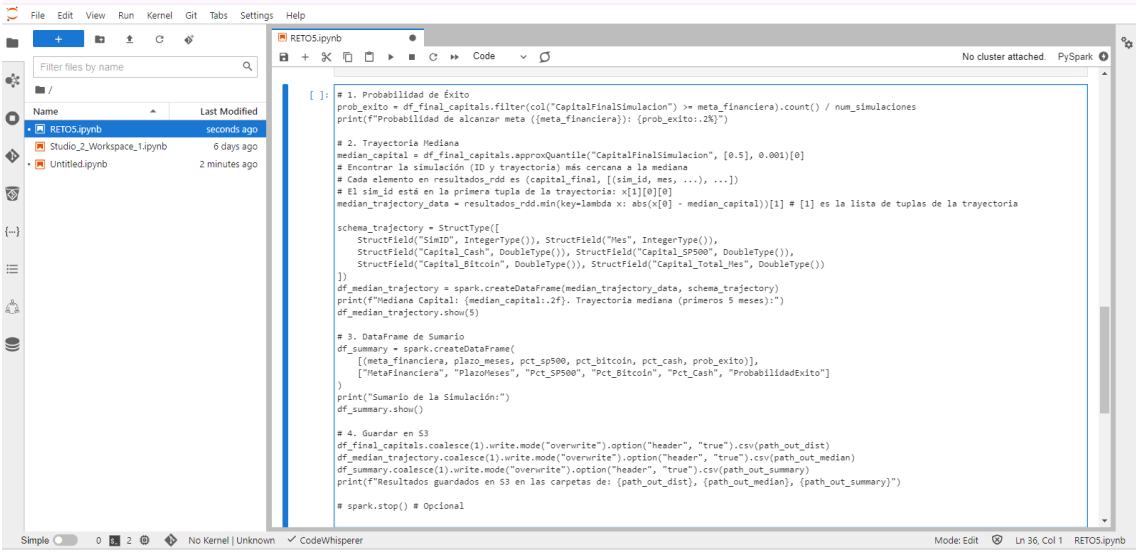


Paso 3 del Notebook: Análisis y Almacenamiento de Resultados (imagen25.jpg)

🎲 Paso 4.2 (EMR Notebook): ¡Lanzando los Dados del Destino Financiero! 💸🎰

- 🎯 Objetivo: ¡Aquí ocurre la magia! Ejecutar miles de escenarios de inversión simulados para ver cómo podría evolucionar el capital del usuario. Queremos:
  1. Simular, en paralelo (gracias a Spark 💪), miles de "vidas financieras" diferentes.
  2. Para cada "vida", rastrear mes a mes cómo crece (¡o decrece!) el dinero invertido en S&P 500 📈, Bitcoin ₿ y lo que queda en efectivo 💵.
  3. Obtener el capital total final para cada una de estas miles de simulaciones.
  4. Guardar también la trayectoria mensual de cada componente de la cartera para, al menos, una simulación representativa (la mediana, por ejemplo).
- 📸 (imagen24.jpg): Una captura del Notebook mostrando la función principal de simulación (simular_una_trayectoria o similar) que proyecta el capital mes a mes, y la sección de código donde se invoca esta función repetidamente (por ejemplo, usando spark.sparkContext.parallelize o un bucle que genera un RDD/DataFrame de resultados) para las numero_simulaciones iteraciones. Podría mostrarse también una pequeña muestra de los resultados parciales.

- **Código PySpark (Celda 3):**


File Edit View Run Kernel Git Tabs Settings Help
Filter files by name
/
Name
Last Modified
RETO5.ipynb
seconds ago
Studio_2_Workspace_1.ipynb
6 days ago
Untitled.ipynb
2 minutes ago
RETO5.ipynb
Code
No cluster attached
PySpark
[ ]:
# 1. Probabilidad de Éxito
prob_exito = df_final_capitals.filter(col("CapitalFinalSimulacion") >= meta_financiera).count() / num_simulaciones
print(f"Probabilidad de alcanzar meta ({meta_financiera}): {prob_exito:.2%}")
# 2. Trayectoria Mediana
median_capital = df_final_capitals.approxQuantile("CapitalFinalSimulacion", [0.5], 0.001)[0]
# Encontrar la simulación (ID y trayectoria) más cercana a la mediana
# Cada elemento en resultados_rdd es (capital_final, [(sim_id, mes, ...), ...])
# El sim_id está en la primera tupla de la trayectoria: x[1][0][0]
median_trajectory_data = resultados_rdd.min(key=lambda x: abs(x[0] - median_capital))[1] # [1] es la lista de tuplas de la trayectoria
schema_trajectory = StructType([
StructField("SimID", IntegerType()), StructField("Mes", IntegerType()),
StructField("Capital_Cash", DoubleType()), StructField("Capital_SP500", DoubleType()),
StructField("Capital_Bitcoin", DoubleType()), StructField("Capital_Total_Mes", DoubleType())
])
df_median_trajectory = spark.createDataFrame(median_trajectory_data, schema_trajectory)
print(f"Mediana Capital: {median_capital:.2f}. Trayectoria mediana (primeros 5 meses):")
df_median_trajectory.show(5)
# 3. DataFrame de Sumario
df_summary = spark.createDataFrame(
[(meta_financiera, plazo_meses, pct_sp500, pct_bitcoin, pct_cash, prob_exito)],
["MetaFinanciera", "PlazoMeses", "Pct_SP500", "Pct_Bitcoin", "Pct_Cash", "ProbabilidadExito"]
)
print("Sumario de la Simulación:")
df_summary.show()
# 4. Guardar en S3
df_final_capitals.coalesce(1).write.mode("overwrite").option("header", "true").csv(path_out_dist)
df_median_trajectory.coalesce(1).write.mode("overwrite").option("header", "true").csv(path_out_median)
df_summary.coalesce(1).write.mode("overwrite").option("header", "true").csv(path_out_summary)
print(f"Resultados guardados en S3 en las carpetas de: {path_out_dist}, {path_out_median}, {path_out_summary}")
# spark.stop() # Opcional
Simple
No Kernel | Unknown
CodeWhisperer
Mode: Edit
Ln 36, Col 1
RETO5.ipynb

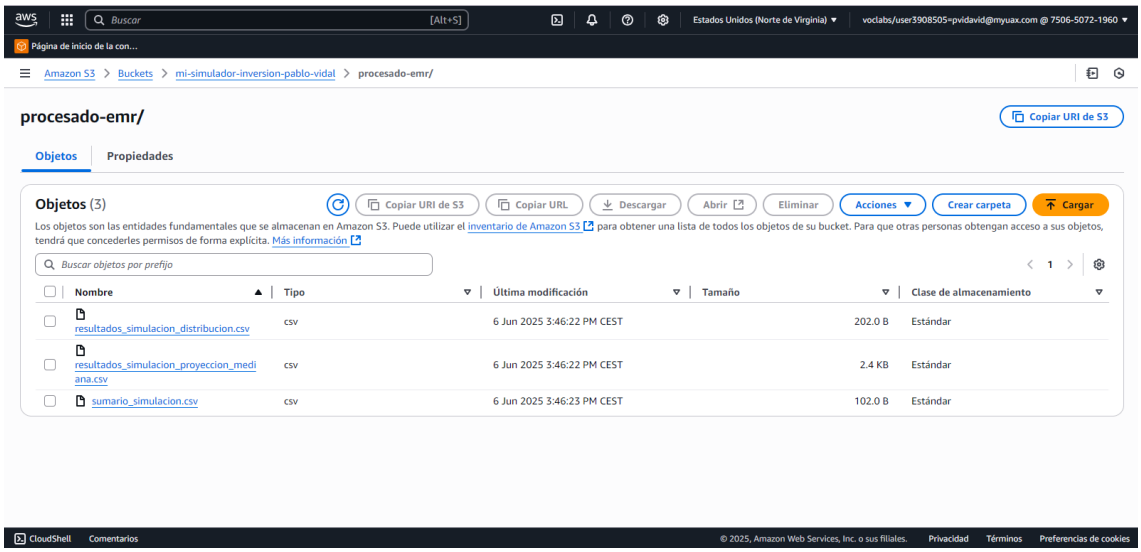
Paso 5: Comprabación de resultados .csv en bucket s3

Archivo 1: resultados_simulacion_distribucion.csv

Archivo 2: resultados_simulacion_proyeccion_mediana.csv

Archivo 3: sumario_simulacion.csv

Imagen26.jpg



🎨 Paso 6 ¡Dando Vida a los Datos! Conexión y Visualización con Power BI ✨📊

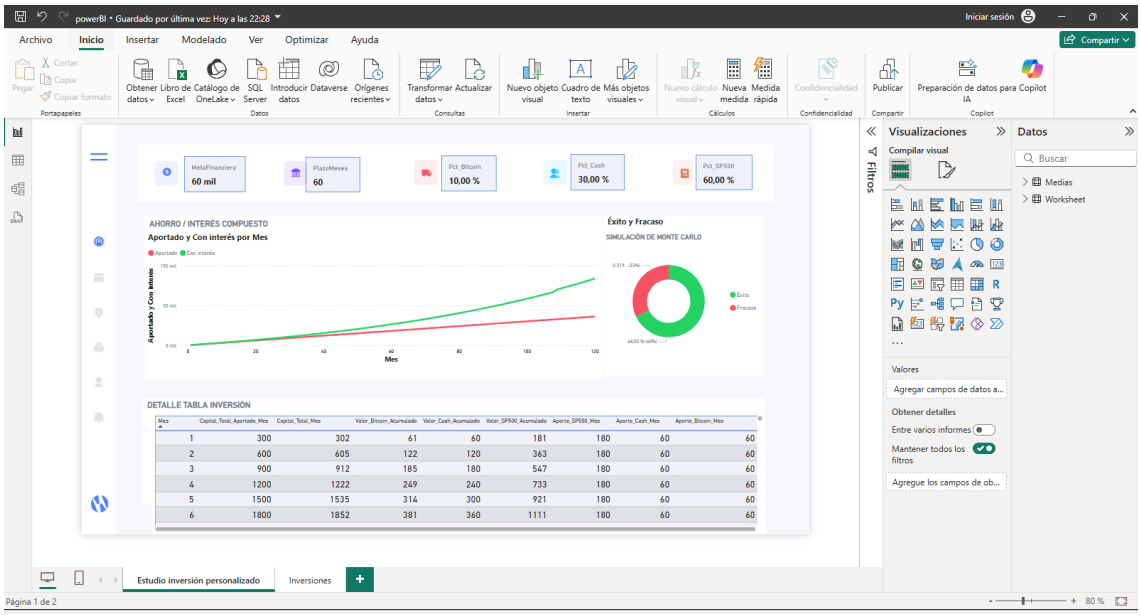
- 🎯 Objetivo del Paso:
    1. Establecer un puente 🌉 entre Power BI Desktop y nuestros resultados de simulación almacenados en Amazon S3.
    2. Importar los datos (los CSVs que EMR generó) a Power BI.
    3. Darles un pequeño "masaje" 🧘 a los datos si es necesario (ej. ajustar formatos de números, tipos de datos) para que Power BI los entienda perfectamente.
    4. ¡Construir el dashboard! 🏗️ Crear los gráficos y tarjetas (KPIs) que contarán la historia financiera de forma clara e impactante.
- 📝 Explicación para el PDF/Presentación (Parte General):

"Con los resultados de nuestra simulación de Monte Carlo cuidadosamente procesados y almacenados en Amazon S3, el siguiente paso es llevar estos datos a Microsoft Power BI para crear un dashboard interactivo y visualmente atractivo. Power BI nos permitirá presentar la información de una manera que sea fácil de entender para el usuario final."

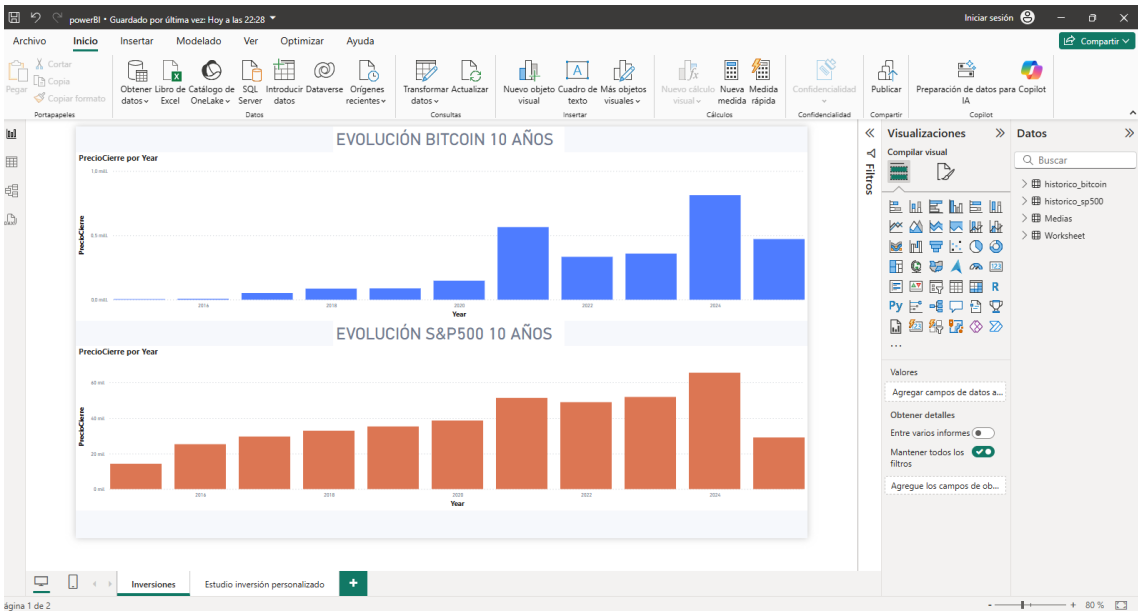
"El proceso implica:"

1."Conectar Power BI a Amazon S3: Utilizaremos el conector de S3 en Power BI (o una conexión a través de 'Web' si usamos URLs pre-firmadas, o cargando los archivos si es una demo puntual) para acceder a los archivos CSV generados por EMR (resultados_simulacion_distribucion.csv, resultados_simulacion_proyeccion_mediana.csv, sumario_simulacion.csv)."

2."Importar y Transformar Datos: Una vez conectados, importaremos los datos a Power BI. Es común que en esta etapa se requieran pequeñas transformaciones usando Power Query Editor. Por ejemplo, asegurar que los números se interpreten correctamente (ej. cambiando comas por puntos como delimitadores decimales si es necesario, dependiendo de la configuración regional del sistema que generó el CSV y la de Power BI), convertir columnas al tipo de dato adecuado (número, texto, fecha), o renombrar columnas para mayor claridad."



En la seguna parte de la representación en Power BI aprovechamos los datos de Yahoo Fiance para mostrar la evolución de nuestras dos inversiones (Bitcoin y S&P 500) en los últimos 10 años. Aprovechamos los datos del bucket Amazon S3/Buckets/mi-simulador-inversion-pablo-vidal/**datos-mercado/ imagen28.jpg**



🏁 Conclusión del Proyecto: Simulador de Inversión Personalizado 🚀✨

Este proyecto ha sido un viaje fascinante a través del mundo del Big Data y la analítica financiera, culminando en la creación de un Simulador de Inversión

Partiendo de la necesidad real de visualizar el camino hacia metas financieras, hemos construido una solución integral que aprovecha la potencia y flexibilidad del ecosistema AWS, junto con la capacidad de visualización de Microsoft Power BI.

Un Recorrido por la Solución:

1.☁️ Cimientos Sólidos en S3: Establecimos nuestro data lake en Amazon S3, organizando los datos de entrada del usuario (extractos bancarios generados programáticamente) y los datos históricos de mercado (S&P 500 y Bitcoin, también obtenidos mediante scripts Python que interactúan con Yahoo Finance).

2. 🛠️ Transformación con AWS Glue:

- •Procesamos los extractos bancarios para obtener un análisis detallado de ingresos, gastos por categoría, y, crucialmente, el aporte mensual base para la inversión.
- •Analizamos los datos históricos de mercado para calcular los parámetros estadísticos esenciales (retorno promedio mensual μ y volatilidad mensual σ) para el S&P 500 y Bitcoin.

Estos procesos de ETL, diseñados en AWS Glue Studio, prepararon los datos de manera eficiente para la siguiente etapa.

3. 🎰 Simulación Potente con Amazon EMR y Spark:

- •Configuramos un clúster de EMR y, utilizando un Notebook de Jupyter (PySpark), implementamos una simulación de Monte Carlo.
- •Este motor tomó el aporte mensual del usuario, los parámetros de mercado (μ y σ), y la estrategia de inversión definida para proyectar miles de posibles escenarios futuros de crecimiento del capital.
- •Calculamos la probabilidad de alcanzar la meta financiera y identificamos la trayectoria mediana para una visualización representativa. Los resultados se almacenaron de nuevo en S3, listos para el análisis.

4. 📊 Visualización Impactante con Power BI:

- •Conectamos Power BI a los resultados en S3.
- •Creamos un dashboard que presenta de forma clara:
    - •Los KPIs clave (meta, plazo, probabilidad de éxito, capital mediano).

•Un gráfico de proyección de la cartera mediana, ilustrando el poder del interés compuesto.

•Una vista de la evolución histórica de las inversiones elegidas (S&P 500 y Bitcoin).

Valor y Aprendizaje:

Este proyecto no solo demuestra la aplicación práctica de tecnologías de Big Data en un caso de uso financiero relevante, sino que también subraya la importancia de un flujo de datos bien orquestado, desde la ingesta y transformación hasta el procesamiento avanzado y la visualización final. La capacidad de integrar servicios como S3, Glue, EMR y Power BI abre un abanico de posibilidades para análisis complejos y toma de decisiones informadas.

Agradecimientos y Cierre :

Ha sido un placer aprender durante el curso. Se nota que David tiene pasión por o que hace y así lo transmite. Desarrollar este simulador, profundizando en cada una de las herramientas y superando los desafíos técnicos que surgieron. Ha sido difícil y he obviado alguna cosa pero la oportunidad de construir algo tan tangible y con potencial de ayudar a las personas a planificar su futuro financiero ha sido increíblemente gratificante.

Espero sinceramente que este proyecto y su presentación sean de su agrado y cumplan con las expectativas. Muchísimas gracias por la oportunidad de trabajar en este reto y por su guía a lo largo del proceso.

🚀 ¡Gracias! 🚀

SCRIPTS PREPARACIÓN INGESTA

generarextractobanario.py

```
import pandas as pd

import random
from datetime import datetime, timedelta
import os # <--- IMPORTACIÓN OS AÑADIDA AQUÍ

def generar_fecha_aleatoria_mes(year, month, dia_inicio=1, dia_fin=28):
  """Genera una fecha aleatoria dentro de un mes y rango de días."""
  start_date = datetime(year, month, dia_inicio)
  # Asegurarse de que dia_fin no exceda el último día del mes
  try:
    # Intentar crear la fecha de fin directamente
    datetime(year, month, dia_fin)
    dia_fin_real = dia_fin
  except ValueError: # Si dia_fin es inválido (ej. 30 de febrero)
    # Usar el último día del mes
    end_of_month = (start_date.replace(day=1) +
timedelta(days=32)).replace(day=1) - timedelta(days=1)
    dia_fin_real = end_of_month.day

  # Asegurarse de que dia_inicio no sea mayor que dia_fin_real
  dia_inicio_real = min(dia_inicio, dia_fin_real)

  random_day = random.randint(dia_inicio_real, dia_fin_real)
  return datetime(year, month, random_day)

def
generar_extracto_simulado(nombre_archivo_salida="extracto_bancario.csv",
                anos_a_generar=10,
                objetivo_saldo_final_mes_cuenta_corriente=20.0):
  """
  Genera un archivo CSV de extracto bancario simulado desde cero.
  Los gastos variables se ajustan para que el saldo de la cuenta
corriente quede equilibrado.
  """

  # Parámetros base de simulación
  nomina_base = 1800.00
  incremento_anual_nomina = 0.02 # 2%
  paga_extra_base_ratio = 0.6 # Como ratio de la nómina base del año
```

```
  hipoteca_base = -450.00
  incremento_anual_hipoteca = 0.015 # 1.5%

  suministros_base = -110.00
  variacion_suministros = 20.00 # +/-

  transferencia_ahorro_base = -300.00
  incremento_anual_transferencia = 0.02 # 2%

  # Rangos para gastos variables (por transacción)
  supermercado_rango = (-150, -90)
  restauracion_rango = (-60, -25)
  gastos_varios_rango = (-150, -40)
  vacaciones_gasto_base = -1500

  transacciones = []
  # Empezar X años atrás desde el inicio del año actual para consistencia
  ano_inicio_sim = datetime.now().year - anos_a_generar

  for anio_offset in range(anos_a_generar):
    ano_actual_sim = ano_inicio_sim + anio_offset

    nomina_anual = nomina_base * ((1 + incremento_anual_nomina) **
anio_offset)
    paga_extra_anual = nomina_anual * paga_extra_base_ratio
    hipoteca_anual = hipoteca_base * ((1 + incremento_anual_hipoteca) **
anio_offset)
    transferencia_ahorro_anual = transferencia_ahorro_base * ((1 +
incremento_anual_transferencia) ** anio_offset)

    for mes_idx in range(1, 13):
      ingresos_este_mes = 0
      gastos_fijos_este_mes = 0
      gastos_variables_este_mes_lista = []

      # --- INGRESOS ---
      fecha_nomina = generar_fecha_aleatoria_mes(ano_actual_sim, mes_idx,
1, 5)
      transacciones.append([fecha_nomina.strftime('%Y-%m-%d'), "NOMINA",
round(nomina_anual, 2)])
      ingresos_este_mes += nomina_anual

      if mes_idx == 6 or mes_idx == 12:
        fecha_paga_extra = generar_fecha_aleatoria_mes(ano_actual_sim,
mes_idx, 15, 25)
```

```
            transacciones.append([fecha_paga_extra.strftime('%Y-%m-%d'),
f"PAGA EXTRA {'VERANO' if mes_idx == 6 else 'NAVIDAD'}",
round(paga_extra_anual, 2)])
            ingresos_este_mes += paga_extra_anual

        if random.random() < 0.1:
            monto_ing_extra = round(random.uniform(50, 400), 2)
            fecha_ing_extra = generar_fecha_aleatoria_mes(ano_actual_sim,
mes_idx, 10, 20)
            transacciones.append([fecha_ing_extra.strftime('%Y-%m-%d'),
"INGRESO EXTRAORDINARIO", monto_ing_extra])
            ingresos_este_mes += monto_ing_extra

        # --- GASTOS FIJOS ---
        fecha_hipoteca = generar_fecha_aleatoria_mes(ano_actual_sim,
mes_idx, 1, 5)
        monto_hipoteca_actual = round(hipoteca_anual, 2)
        transacciones.append([fecha_hipoteca.strftime('%Y-%m-%d'),
"HIPOTECA", monto_hipoteca_actual])
        gastos_fijos_este_mes += abs(monto_hipoteca_actual)

        fecha_suministros = generar_fecha_aleatoria_mes(ano_actual_sim,
mes_idx, 5, 10)
        monto_suministros = round(suministros_base + random.uniform(-
variacion_suministros, variacion_suministros), 2)
        # Asegurarse que suministros sea negativo
        monto_suministros = -abs(monto_suministros) if monto_suministros >
0 else monto_suministros
        transacciones.append([fecha_suministros.strftime('%Y-%m-%d'),
"SUMINISTROS (LUZ, AGUA, GAS)", monto_suministros])
        gastos_fijos_este_mes += abs(monto_suministros)

        fecha_transferencia = generar_fecha_aleatoria_mes(ano_actual_sim,
mes_idx, 25, 28)
        monto_transferencia_actual = round(transferencia_ahorro_anual, 2)
        # Asegurarse que transferencia sea negativa
        monto_transferencia_actual = -abs(monto_transferencia_actual) if
monto_transferencia_actual > 0 else monto_transferencia_actual
        transacciones.append([fecha_transferencia.strftime('%Y-%m-%d'),
"TRANSFERENCIA AHORRO", monto_transferencia_actual])
        gastos_fijos_este_mes += abs(monto_transferencia_actual)

        # --- GASTOS VARIABLES (simulación inicial) ---
        for _ in range(random.randint(3, 5)):
            monto = round(random.uniform(supermercado_rango[0],
supermercado_rango[1]), 2)
```

```
            gastos_variables_este_mes_lista.append(("SUPERMERCADO", monto))

        for _ in range(random.randint(4, 8)):
            monto = round(random.uniform(restauracion_rango[0],
restauracion_rango[1]), 2)
            gastos_variables_este_mes_lista.append(("RESTAURACION", monto))

        for _ in range(random.randint(2, 5)):
            monto = round(random.uniform(gastos_varios_rango[0],
gastos_varios_rango[1]), 2)
            gastos_variables_este_mes_lista.append(("GASTOS VARIOS (OCIO,
TRANSPORTE, ETC)", monto))

        if (mes_idx == 7 or mes_idx == 8) and random.random() < 0.7:
            monto_vacaciones = round(vacaciones_gasto_base *
random.uniform(0.8, 1.5), 2)
            # Asegurarse que vacaciones sea negativo
            monto_vacaciones = -abs(monto_vacaciones) if monto_vacaciones > 0
else monto_vacaciones
            gastos_variables_este_mes_lista.append(("GASTO VACACIONES",
monto_vacaciones))

        # --- AJUSTE DE GASTOS VARIABLES ---
        total_gastos_variables_original = sum(abs(g[1]) for g in
gastos_variables_este_mes_lista)

        margen_disponible_para_variables = ingresos_este_mes -
gastos_fijos_este_mes - objetivo_saldo_final_mes_cuenta_corriente
        margen_disponible_para_variables = max(0,
margen_disponible_para_variables)

        factor_ajuste = 1.0
        if total_gastos_variables_original > 0:
            factor_ajuste = margen_disponible_para_variables /
total_gastos_variables_original
        factor_ajuste = min(1.0, factor_ajuste)

        for concepto_var, importe_var_original in
gastos_variables_este_mes_lista:
            importe_var_ajustado = round(- (abs(importe_var_original) *
factor_ajuste), 2)
            fecha_gasto_var = generar_fecha_aleatoria_mes(ano_actual_sim,
mes_idx)
            transacciones.append([fecha_gasto_var.strftime('%Y-%m-%d'),
concepto_var, importe_var_ajustado])
```

```
  # Crear DataFrame y guardar
  df_final = pd.DataFrame(transacciones, columns=['Fecha', 'Concepto',
'Importe'])
  df_final['Fecha'] = pd.to_datetime(df_final['Fecha'])
  df_final = df_final.sort_values(by='Fecha')
  df_final['Fecha'] = df_final['Fecha'].dt.strftime('%Y-%m-%d')
  df_final['Importe'] = df_final['Importe'].round(2)

  directorio_script = os.path.dirname(os.path.abspath(__file__))
  ruta_completa_salida = os.path.join(directorio_script,
nombre_archivo_salida)

  try:
    df_final.to_csv(ruta_completa_salida, index=False,
float_format='%.2f')
    print(f"Extracto bancario simulado generado y guardado en:
{ruta_completa_salida}")
  except Exception as e:
    print(f"Error al guardar el CSV: {e}")

# --- Ejecución del Script ---
if __name__ == "__main__":

generar_extracto_simulado(nombre_archivo_salida="extracto_bancario.csv",
                anos_a_generar=10,
                objetivo_saldo_final_mes_cuenta_corriente=25.0)
```

generarcsvsp500.py

```
import yfinance as yf

import datetime

# Definir el ticker y el rango de fechas
ticker_symbol = "^GSPC"
end_date = datetime.date.today()
start_date = end_date - datetime.timedelta(days=10*365.25) #
Aproximadamente 10 años

# Descargar los datos históricos
# yfinance ya ajusta los precios por defecto (auto_adjust=True)
# así que la columna 'Close' ya será el cierre ajustado.
data = yf.download(ticker_symbol, start=start_date, end=end_date,
interval="1mo")

# Imprimir las columnas disponibles para depurar (opcional)
# print("Columnas descargadas:", data.columns)

# Seleccionar solo las columnas necesarias y renombrar
if not data.empty:
  # Como auto_adjust es True por defecto, usamos 'Close' que ya está
ajustado
  data_cleaned = data[['Close']].copy() # Usamos .copy() para evitar
SettingWithCopyWarning
  data_cleaned.rename(columns={'Close': 'PrecioCierre'}, inplace=True)
  data_cleaned.index.name = 'Fecha' # El índice ya es la fecha

  # Guardar a CSV
  nombre_archivo_csv = "sp500_historico.csv"
  data_cleaned.to_csv(nombre_archivo_csv)
  print(f"Datos guardados en {nombre_archivo_csv}")

  # Mostrar las primeras filas del DataFrame guardado (opcional)
  # print("\nPrimeras filas del CSV generado:")
  # print(data_cleaned.head())
else:
  print("No se pudieron descargar los datos.")
```

generarcsvsp500.py

```
import yfinance as yf

import datetime

# Definir el ticker y el rango de fechas
ticker_symbol_btc = "BTC-USD" # Ticker para Bitcoin
end_date_btc = datetime.date.today()
# Para Bitcoin, 10 años podría ser mucho, ya que no tenía tanta
liquidez/datos al principio.
# yfinance tomará los datos disponibles desde el inicio si 10 años es
demasiado.
# O puedes poner una fecha de inicio específica si lo prefieres, ej:
datetime.date(2014, 9, 17)
start_date_btc = end_date_btc - datetime.timedelta(days=10*365.25) #
Aproximadamente 10 años

# Descargar los datos históricos para Bitcoin
# yfinance ya ajusta los precios por defecto (auto_adjust=True)
data_btc = yf.download(ticker_symbol_btc, start=start_date_btc,
end=end_date_btc, interval="1mo")

# Imprimir las columnas disponibles para depurar (opcional)
# print(f"Columnas descargadas para {ticker_symbol_btc}:",
data_btc.columns)

# Seleccionar solo las columnas necesarias y renombrar
if not data_btc.empty:
  # Usamos 'Close' que ya está ajustado
  data_btc_cleaned = data_btc[['Close']].copy() # Usamos .copy() para
evitar SettingWithCopyWarning
  data_btc_cleaned.rename(columns={'Close': 'PrecioCierre'},
inplace=True)
  data_btc_cleaned.index.name = 'Fecha' # El índice ya es la fecha

  # Guardar a CSV
  nombre_archivo_csv_btc = "bitcoin_historico.csv"
  data_btc_cleaned.to_csv(nombre_archivo_csv_btc)
  print(f"Datos de Bitcoin guardados en {nombre_archivo_csv_btc}")

  # Mostrar las primeras filas del DataFrame guardado (opcional)
  # print(f"\nPrimeras filas del CSV de {ticker_symbol_btc} generado:")
  # print(data_btc_cleaned.head())
else:
  print(f"No se pudieron descargar los datos para {ticker_symbol_btc}.")
```

SCRIPTS NOTEBOOK PYSPARK

Celda 1

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import col, lag, avg, stddev, abs as _abs
from pyspark.sql.window import Window
from pyspark.sql.types import DoubleType
import numpy as np

spark =
SparkSession.builder.appName("SimuladorInversionMonteCarlo").getOrCreate(
)

# --- Parámetros ---
meta_financiera, plazo_anios, pct_sp500, pct_bitcoin = 60000.0, 5, 0.60,
0.10
plazo_meses = plazo_anios * 12
pct_cash = 1.0 - pct_sp500 - pct_bitcoin
num_simulaciones = 10000
bucket_name = "mi-simulador-inversion-pablo-vidal" # ¡TU BUCKET!

# --- Rutas S3 (Concatenadas) ---
s3_base = f"s3a://{bucket_name}"
path_sp500 = f"{s3_base}/datos-mercado/sp500_historico.csv"
path_bitcoin = f"{s3_base}/datos-mercado/bitcoin_historico.csv"
path_ahorro_usuario =
f"{s3_base}/procesado-glue/ahorro_promedio_usuario/"
path_out_dist =
f"{s3_base}/procesado-emr/resultados_simulacion_distribucion/"
path_out_median =
f"{s3_base}/procesado-emr/resultados_simulacion_proyeccion_mediana/"
path_out_summary = f"{s3_base}/procesado-emr/sumario_simulacion/"

print(f"Spark: OK. Meta: {meta_financiera}, Plazo: {plazo_meses}m,
S&P500:{pct_sp500:.0%}, BTC:{pct_bitcoin:.0%}, Cash:{pct_cash:.0%}")

def get_market_params(path, name):
  df = spark.read.csv(path, header=True, inferSchema=True)
  df = df.withColumn("PrecioCierre",
col("PrecioCierre").cast(DoubleType())) \
      .withColumn("Fecha", col("Fecha").cast("date")) # Asume formato
parseable
```

```
  df = df.withColumn("RetornoMensual", (col("PrecioCierre") -
lag("PrecioCierre", 1).over(Window.orderBy("Fecha"))) /
lag("PrecioCierre", 1).over(Window.orderBy("Fecha")))
  stats =
df.na.drop(subset=["RetornoMensual"]).select(avg("RetornoMensual").alias(
"mu"), stddev("RetornoMensual").alias("sigma")).first()
  if not stats or stats["mu"] is None or stats["sigma"] is None: raise
ValueError(f"Params NA for {name}")
  print(f"{name}: μ={stats['mu']:.4%}, σ={stats['sigma']:.4%}")
  return stats["mu"], stats["sigma"]

try:
  mu_sp500, sigma_sp500 = get_market_params(path_sp500, "S&P500")
  mu_bitcoin, sigma_bitcoin = get_market_params(path_bitcoin, "Bitcoin")
  ahorro_mensual_usuario = spark.read.csv(path_ahorro_usuario,
header=True, inferSchema=True).first()["AhorroMensualPromedio"]
  print(f"Ahorro Usuario: {ahorro_mensual_usuario:.2f}")
except Exception as e:
  print(f"Error en preparación de datos: {e}. Revisar S3 y job de Glue
previo.")
  raise
```

Celda 2

```
def run_simulation_trajectory(sim_id):

  # Variables locales para esta simulación (evita problemas de
serialización con np.random si se usa fuera)
  local_mu_sp500, local_sigma_sp500 = mu_sp500, sigma_sp500
  local_mu_bitcoin, local_sigma_bitcoin = mu_bitcoin, sigma_bitcoin
  local_ahorro_mensual = ahorro_mensual_usuario
  local_plazo_meses = plazo_meses
  local_pct_sp500, local_pct_bitcoin, local_pct_cash = pct_sp500,
pct_bitcoin, pct_cash

  cap_sp, cap_btc, cap_cash = 0.0, 0.0, 0.0
  trajectory = []
  for month in range(1, local_plazo_meses + 1):
    ret_sp = np.random.normal(local_mu_sp500, local_sigma_sp500)
    ret_btc = np.random.normal(local_mu_bitcoin, local_sigma_bitcoin)

    cap_sp = max(0, (cap_sp + local_ahorro_mensual * local_pct_sp500) *
(1 + ret_sp))
    cap_btc = max(0, (cap_btc + local_ahorro_mensual * local_pct_bitcoin)
* (1 + ret_btc))
    cap_cash += local_ahorro_mensual * local_pct_cash
    trajectory.append((sim_id, month, cap_cash, cap_sp, cap_btc, cap_cash
+ cap_sp + cap_btc))
  return cap_cash + cap_sp + cap_btc, trajectory

# Ejecutar simulaciones
simulation_ids_rdd =
spark.sparkContext.parallelize(range(num_simulaciones))
# resultados_rdd contiene tuplas: (capital_final_total,
lista_de_tuplas_de_trayectoria_mensual)
resultados_rdd =
simulation_ids_rdd.map(run_simulation_trajectory).cache()

# Obtener solo los capitales finales para análisis de distribución
df_final_capitals = resultados_rdd.map(lambda x:
(x[0],)).toDF(["CapitalFinalSimulacion"])

print(f"Simulación de {num_simulaciones} trayectorias completada.")
df_final_capitals.show(5)
```

Celda 3

```
from pyspark.sql.types import StructType, StructField, IntegerType,
DoubleType # Redefinir por si se ejecuta la celda aislada


# 1. Probabilidad de Éxito
prob_exito = df_final_capitals.filter(col("CapitalFinalSimulacion") >=
meta_financiera).count() / num_simulaciones
print(f"Probabilidad de alcanzar meta ({meta_financiera}):
{prob_exito:.2%}")

# 2. Trayectoria Mediana
median_capital =
df_final_capitals.approxQuantile("CapitalFinalSimulacion", [0.5], 0.001)
[0]
# Encontrar la simulación (ID y trayectoria) más cercana a la mediana
# Cada elemento en resultados_rdd es (capital_final, [(sim_id, mes, ...),
...])
# El sim_id está en la primera tupla de la trayectoria: x[1][0][0]
median_trajectory_data = resultados_rdd.min(key=lambda x: abs(x[0] -
median_capital))[1] # [1] es la lista de tuplas de la trayectoria

schema_trajectory = StructType([
  StructField("SimID", IntegerType()), StructField("Mes", IntegerType()),
  StructField("Capital_Cash", DoubleType()), StructField("Capital_SP500",
DoubleType()),
  StructField("Capital_Bitcoin", DoubleType()),
StructField("Capital_Total_Mes", DoubleType())
])
df_median_trajectory = spark.createDataFrame(median_trajectory_data,
schema_trajectory)
print(f"Mediana Capital: {median_capital:.2f}. Trayectoria mediana
(primeros 5 meses):")
df_median_trajectory.show(5)

# 3. DataFrame de Sumario
df_summary = spark.createDataFrame(
  [(meta_financiera, plazo_meses, pct_sp500, pct_bitcoin, pct_cash,
prob_exito)],
  ["MetaFinanciera", "PlazoMeses", "Pct_SP500", "Pct_Bitcoin",
"Pct_Cash", "ProbabilidadExito"]
)
print("Sumario de la Simulación:")
df_summary.show()

# 4. Guardar en S3
```

```
df_final_capitals.coalesce(1).write.mode("overwrite").option("header",
"true").csv(path_out_dist)
df_median_trajectory.coalesce(1).write.mode("overwrite").option("header",
"true").csv(path_out_median)
df_summary.coalesce(1).write.mode("overwrite").option("header",
"true").csv(path_out_summary)
print(f"Resultados guardados en S3 en las carpetas de: {path_out_dist},
{path_out_median}, {path_out_summary}")
```

CÓDIGOS SQL TRANSFORMACIÓN DE DATOS

Código transormación Transformar_extracto_detallado ETL visual Job Extracto_bancario

```
WITH TransaccionesConFecha AS (

  SELECT
    to_date(fecha, 'yyyy-MM-dd') AS FechaTransaccion, -- Asegúrate que
'fecha' es minúscula si así está en tu tabla
    LOWER(concepto) as concepto_lower, -- Asegúrate que 'concepto' es
minúscula
    CAST(importe AS DOUBLE) AS ImporteNumerico -- Asegúrate que 'importe'
es minúscula
  FROM myDataSource -- REEMPLAZA myDataSource con el nombre de tu nodo de
origen
),
CategorizacionDetallada AS (
  SELECT
    YEAR(FechaTransaccion) AS Anio,
    MONTH(FechaTransaccion) AS Mes,
    ImporteNumerico,
    concepto_lower,
    -- Clasificación primaria para Ingresos y Gastos (base para cálculo
de ahorro potencial)
    CASE
      WHEN ImporteNumerico > 0 THEN 'Ingreso_Operativo'
      WHEN concepto_lower LIKE '%transferencia%ahorro%' THEN
'Transferencia_Ahorro_Saliente' -- Marcarla especial
      WHEN ImporteNumerico < 0 THEN 'Gasto_Operativo'
      ELSE 'Otro'
    END as TipoPrincipalMovimiento
  FROM TransaccionesConFecha
),
AgregadosPorCategoriaMes AS (
  SELECT
    Anio,
    Mes,
    SUM(CASE WHEN concepto_lower LIKE '%nomina%' AND ImporteNumerico > 0
THEN ImporteNumerico ELSE 0 END) AS Ing_Nomina,
    SUM(CASE WHEN concepto_lower LIKE '%paga%extra%' AND ImporteNumerico
> 0 THEN ImporteNumerico ELSE 0 END) AS Ing_PagaExtra,
    SUM(CASE WHEN concepto_lower LIKE '%ingreso%extraordinario%' AND
ImporteNumerico > 0 THEN ImporteNumerico ELSE 0 END) AS
Ing_Extraordinario,
```

```
    SUM(CASE WHEN concepto_lower LIKE '%hipoteca%' AND ImporteNumerico <
0 THEN ABS(ImporteNumerico) ELSE 0 END) AS Gasto_Hipoteca,
    SUM(CASE WHEN concepto_lower LIKE '%suministros%' AND ImporteNumerico
< 0 THEN ABS(ImporteNumerico) ELSE 0 END) AS Gasto_Suministros,
    SUM(CASE WHEN concepto_lower LIKE '%supermercado%' AND
ImporteNumerico < 0 THEN ABS(ImporteNumerico) ELSE 0 END) AS
Gasto_Supermercado,
    SUM(CASE WHEN concepto_lower LIKE '%restauracion%' AND
ImporteNumerico < 0 THEN ABS(ImporteNumerico) ELSE 0 END) AS
Gasto_Restauracion,
    -- Gastos Varios: Asegurarse que no se solape con otras categorías de
gasto ya definidas
    SUM(CASE
        WHEN ImporteNumerico < 0
        AND NOT (concepto_lower LIKE '%hipoteca%')
        AND NOT (concepto_lower LIKE '%suministros%')
        AND NOT (concepto_lower LIKE '%supermercado%')
        AND NOT (concepto_lower LIKE '%restauracion%')
        AND NOT (concepto_lower LIKE '%vacaciones%')
        AND NOT (concepto_lower LIKE '%transferencia%ahorro%')
        -- Puedes añadir más exclusiones si tienes más categorías
específicas
        THEN ABS(ImporteNumerico)
        ELSE 0
      END) AS Gasto_OtrosVarios, -- Renombrado para claridad
    SUM(CASE WHEN concepto_lower LIKE '%vacaciones%' AND ImporteNumerico
< 0 THEN ABS(ImporteNumerico) ELSE 0 END) AS Gasto_Vacaciones,
    SUM(CASE WHEN TipoPrincipalMovimiento =
'Transferencia_Ahorro_Saliente' THEN ABS(ImporteNumerico) ELSE 0 END) AS
Monto_TransferenciaAhorro -- Usar ABS
  FROM CategorizacionDetallada
  GROUP BY Anio, Mes
)
SELECT
  Anio,
  Mes,
  COALESCE(Ing_Nomina, 0) AS Ing_Nomina,
  COALESCE(Ing_PagaExtra, 0) AS Ing_PagaExtra,
  COALESCE(Ing_Extraordinario, 0) AS Ing_Extraordinario,
  (COALESCE(Ing_Nomina, 0) + COALESCE(Ing_PagaExtra, 0) +
COALESCE(Ing_Extraordinario, 0)) AS Total_Ingresos_Mes,

  COALESCE(Gasto_Hipoteca, 0) AS Gasto_Hipoteca,
  COALESCE(Gasto_Suministros, 0) AS Gasto_Suministros,
  COALESCE(Gasto_Supermercado, 0) AS Gasto_Supermercado,
  COALESCE(Gasto_Restauracion, 0) AS Gasto_Restauracion,
```

```
  COALESCE(Gasto_OtrosVarios, 0) AS Gasto_OtrosVarios,
  COALESCE(Gasto_Vacaciones, 0) AS Gasto_Vacaciones,
  (COALESCE(Gasto_Hipoteca, 0) + COALESCE(Gasto_Suministros, 0) +
COALESCE(Gasto_Supermercado, 0) + COALESCE(Gasto_Restauracion, 0) +
COALESCE(Gasto_OtrosVarios, 0) + COALESCE(Gasto_Vacaciones, 0)) AS
Total_Gastos_Operativos_Mes,

  COALESCE(Monto_TransferenciaAhorro, 0) AS Monto_TransferenciaAhorro,

  ((COALESCE(Ing_Nomina, 0) + COALESCE(Ing_PagaExtra, 0) +
COALESCE(Ing_Extraordinario, 0)) - (COALESCE(Gasto_Hipoteca, 0) +
COALESCE(Gasto_Suministros, 0) + COALESCE(Gasto_Supermercado, 0) +
COALESCE(Gasto_Restauracion, 0) + COALESCE(Gasto_OtrosVarios, 0) +
COALESCE(Gasto_Vacaciones, 0))) AS Ahorro_Potencial_Bruto_Mes,

  ((COALESCE(Ing_Nomina, 0) + COALESCE(Ing_PagaExtra, 0) +
COALESCE(Ing_Extraordinario, 0)) - (COALESCE(Gasto_Hipoteca, 0) +
COALESCE(Gasto_Suministros, 0) + COALESCE(Gasto_Supermercado, 0) +
COALESCE(Gasto_Restauracion, 0) + COALESCE(Gasto_OtrosVarios, 0) +
COALESCE(Gasto_Vacaciones, 0)) - COALESCE(Monto_TransferenciaAhorro, 0))
AS Saldo_Neto_Cuenta_Corriente_Mes
FROM AgregadosPorCategoriaMes
ORDER BY Anio, Mes;
```

Código transormación Calcular_Promedios_Gasto ETL visual Job Extracto_bancario

```
-- Este SQL opera sobre la salida del nodo
'Transformar_Extracto_Detallado'

-- Reemplaza 'detalle_mensual_proveniente_del_nodo_anterior'
-- por el nombre real de la tabla/alias que Glue Studio le da a la salida
del nodo anterior.

SELECT
  'Hipoteca' AS Categoria_Gasto,
  AVG(Gasto_Hipoteca) AS Promedio_Mensual_Gasto
FROM myDataSource
WHERE Gasto_Hipoteca > 0 -- Opcional: solo promediar meses con este gasto

UNION ALL

SELECT
  'Suministros' AS Categoria_Gasto,
  AVG(Gasto_Suministros) AS Promedio_Mensual_Gasto
FROM myDataSource
WHERE Gasto_Suministros > 0

UNION ALL

SELECT
  'Supermercado' AS Categoria_Gasto,
  AVG(Gasto_Supermercado) AS Promedio_Mensual_Gasto
FROM myDataSource
WHERE Gasto_Supermercado > 0

UNION ALL

SELECT
  'Restauracion' AS Categoria_Gasto,
  AVG(Gasto_Restauracion) AS Promedio_Mensual_Gasto
FROM myDataSource
WHERE Gasto_Restauracion > 0

UNION ALL

SELECT
  'OtrosVarios' AS Categoria_Gasto, -- Corresponde a Gasto_OtrosVarios
  AVG(Gasto_OtrosVarios) AS Promedio_Mensual_Gasto
FROM myDataSource
WHERE Gasto_OtrosVarios > 0
```

```sql
UNION ALL

SELECT
  'Vacaciones' AS Categoria_Gasto,
  AVG(Gasto_Vacaciones) AS Promedio_Mensual_Gasto
FROM myDataSource
WHERE Gasto_Vacaciones > 0

UNION ALL

SELECT
  'TransferenciaAhorro' AS Categoria_Gasto, -- Corresponde a
Monto_TransferenciaAhorro
  AVG(Monto_TransferenciaAhorro) AS Promedio_Mensual_Gasto
FROM myDataSource
WHERE Monto_TransferenciaAhorro > 0;
```

Código transormación Calcular_Aporte_Base_Simulacion ETL visual Job Extracto_bancario

```
-- Este SQL opera sobre la salida del nodo
'Transformar_Extracto_Detallado'

-- Reemplaza 'detalle_mensual_proveniente_del_nodo_anterior'
-- por el nombre real de la tabla/alias que Glue Studio le da a la salida
del primer nodo SQL.

SELECT
  AVG(Monto_TransferenciaAhorro) AS AporteMensualBaseParaInversion
FROM myDataSource
WHERE Monto_TransferenciaAhorro > 0; -- Importante: Solo promediar si
hubo transferencia
                   -- para no diluir el promedio con meses sin
transferencia (si los hubiera).
```

Código transormación SQL_Calc_Retornos_SP500 ETL visual Job job_procesar_datos_mercado

```
--Para SP500

WITH PreciosOrdenados AS (
  SELECT
    -- Asumimos que col0 es la fecha y col1 es el precio de cierre
    to_date(col0, 'yyyy-MM-dd') AS FechaPrecio, -- O el nombre correcto
de tu columna de fecha
    CAST(col1 AS DOUBLE) AS Precio             -- O el nombre correcto de
tu columna de precio
  FROM myDataSource -- REEMPLAZA con el nombre real de tu tabla/alias de
entrada
  ORDER BY FechaPrecio ASC
),
PreciosConAnterior AS (
  SELECT
    FechaPrecio,
    Precio,
    LAG(Precio, 1, NULL) OVER (ORDER BY FechaPrecio ASC) AS
PrecioAnterior
  FROM PreciosOrdenados
),
RetornosMensuales AS (
  SELECT
    FechaPrecio, -- Mantenemos la fecha aquí
    Precio,
    PrecioAnterior,
    CASE
      WHEN PrecioAnterior IS NOT NULL AND PrecioAnterior != 0 THEN
(Precio - PrecioAnterior) / PrecioAnterior
      ELSE NULL
    END AS RetornoMensual
  FROM PreciosConAnterior
  WHERE PrecioAnterior IS NOT NULL -- Solo filas donde hay un precio
anterior para calcular retorno
)
-- CORRECCIÓN AQUÍ: Incluir FechaPrecio en el SELECT final de este CTE
SELECT
  'SP500' as Activo,
  FechaPrecio, -- <-- AÑADIDO
  RetornoMensual
FROM RetornosMensuales
WHERE RetornoMensual IS NOT NULL;
```

Código transormación SQL_Calc_Retornos_500 ETL visual Job job_procesar_datos_mercado

```
WITH PreciosOrdenados AS (

  SELECT
    -- Asumimos que col0 es la fecha y col1 es el precio de cierre para los datos de Bitcoin también
    -- Si los nombres de columna son diferentes para Bitcoin (ej. 'btc_fecha', 'btc_precio'), ajústalos.
    to_date(col0, 'yyyy-MM-dd') AS FechaPrecio, -- O el nombre correcto de tu columna de fecha para Bitcoin
    CAST(col1 AS DOUBLE) AS Precio             -- O el nombre correcto de tu columna de precio para Bitcoin
  FROM myDataSource -- REEMPLAZA con el nombre real de tu tabla/alias de entrada para Bitcoin
                    -- Si es el mismo 'myDataSource' pero de un nodo de origen diferente, asegúrate de que
                    -- Glue Studio lo maneje correctamente. Si el nombre del alias de entrada es el mismo
                    -- para dos fuentes diferentes que alimentan dos nodos SQL distintos, puede ser confuso.
                    -- Es mejor si cada nodo de origen tiene un alias de salida único
                    -- que luego usas en el FROM del nodo SQL correspondiente.
  ORDER BY FechaPrecio ASC
),
PreciosConAnterior AS (
  SELECT
    FechaPrecio,
    Precio,
    LAG(Precio, 1, NULL) OVER (ORDER BY FechaPrecio ASC) AS PrecioAnterior
  FROM PreciosOrdenados
),
RetornosMensuales AS (
  SELECT
    FechaPrecio, -- Mantenemos la fecha aquí
    Precio,
    PrecioAnterior,
    CASE
      WHEN PrecioAnterior IS NOT NULL AND PrecioAnterior != 0 THEN (Precio - PrecioAnterior) / PrecioAnterior
      ELSE NULL
```

```
    END AS RetornoMensual
  FROM PreciosConAnterior
  WHERE PrecioAnterior IS NOT NULL -- Solo filas donde hay un precio
anterior para calcular retorno
)
-- CORRECCIÓN AQUÍ: Incluir FechaPrecio y cambiar el nombre del Activo
SELECT
  'Bitcoin' as Activo,  -- <-- CAMBIO AQUÍ
  FechaPrecio,
  RetornoMensual
FROM RetornosMensuales
WHERE RetornoMensual IS NOT NULL;
```

Código transormación SQL_Union_Retornos ETL visual Job
job_procesar_datos_mercado

```
-- Este SQL une los resultados de los dos nodos anteriores.

-- Reemplaza 'retornos_sp500_output' y 'retornos_bitcoin_output'
-- con los nombres reales de las tablas/alias de entrada que Glue Studio
proporciona.

SELECT
  Activo,
  FechaPrecio,
  RetornoMensual
FROM myDataSourceSP500 -- Salida del nodo SQL para S&P 500

UNION ALL

SELECT
  Activo,
  FechaPrecio,
  RetornoMensual
FROM myDataSourceBitcoin; -- Salida del nodo SQL para Bitcoin
```

Código transormación SQL_Calc_Estadisticas_Mercado ETL visual Job job_procesar_datos_mercado

```
-- Este SQL calcula el promedio (mu) y la desviación estándar (sigma)
-- para cada activo, a partir de la tabla que contiene todos los retornos
mensuales unidos.

-- IMPORTANTE: Reemplaza 'tabla_unificada_de_retornos' en la cláusula
FROM
-- con el nombre real del alias/tabla de entrada que Glue Studio
proporciona
-- para la salida del nodo 'SQL_Union_Retornos'.

SELECT
  Activo,
  AVG(RetornoMensual) AS RetornoPromedioMensual_mu,
  STDDEV_SAMP(RetornoMensual) AS VolatilidadMensual_sigma
  -- Explicación de las funciones:
  -- AVG(RetornoMensual): Calcula el promedio de la columna
'RetornoMensual'.
  --                      Este será nuestro parámetro 'mu' (retorno
esperado).
  -- STDDEV_SAMP(RetornoMensual): Calcula la desviación estándar muestral
de la columna 'RetornoMensual'.
  --                              Esta será nuestra medida de 'sigma'
(volatilidad).
  --                              STDDEV_SAMP es comúnmente usado para
series financieras.
  --                              Si se consideraran los datos como una
población completa, se usaría STDDEV_POP.
FROM
  myDataSource -- <<< ¡¡¡REEMPLAZA ESTE NOMBRE SI NO ES CORRECTO!!!
               -- Este es el alias de la tabla que resulta del nodo
'SQL_Union_Retornos'.
               -- Verifica en Glue Studio cuál es este nombre (ej.
input_0, sql_union_retornos_alias, etc.)
GROUP BY
  Activo
  -- GROUP BY Activo: Asegura que las funciones de agregación (AVG,
STDDEV_SAMP)
  --                 se calculen por separado para cada valor único en
la columna 'Activo'
```

```
  --                  (es decir, una fila de resultados para 'SP500' y
otra para 'Bitcoin').
;
```