“ANÁLISIS PREDICTIVO DE VENTAS RETAIL MEDIANTE CIENCIA DE DATOS”

Universidad La Salle Bajío
Facultad de Ingeniería – Ingeniería en Software
Materia: Minería de Datos
Profesor: Rodolfo Rocha

Proyecto Final – Análisis de Datos RetailX
Reporte del Proyecto Final – Parcial 3

Integrantes del Equipo:

Bardo Arion Aranda – Matrícula: 78522

Edgar Cabrera Velásquez – Matrícula: 77344

Víctor Manuel Ortiz Feregrino – Matrícula: 77869

Luisa Fernanda Vázquez Razo – Matrícula: 7678

Juan Francisco Osorno Martínez – Matrícula: 77268

Sebastián Rodríguez Martínez – Matrícula: 76734

Fecha: 4 / 12 / 2025

ISSC25_DM_PFRep_ArandaBardoArión.pdf

Tabla de contenido

1. Introducción

1.1 Contexto del proyecto

Hoy en día, las tiendas y empresas del sector retail operan en un entorno donde la competencia es cada vez más intensa y los consumidores cuentan con una amplia variedad de opciones para elegir. La publicidad digital, las compras en línea y la comparación inmediata de precios han cambiado por completo la forma en que las personas compran. Los clientes ahora buscan mejores precios, productos de calidad, rapidez en el servicio y experiencias de compra más personalizadas.

Ante esta nueva realidad, entender el comportamiento del cliente se vuelve una tarea fundamental. Las empresas ya no pueden basar sus decisiones únicamente en la intuición o en la experiencia pasada, sino que necesitan apoyarse en información concreta. Por ejemplo, deben saber en qué meses hay más ventas, qué productos son los más comprados, cómo varían los hábitos según la edad o el género, y qué factores influyen en el aumento o disminución de sus ingresos.
Para responder estas preguntas, la Ciencia de Datos se ha convertido en una herramienta esencial. Mediante el análisis de información, el uso de gráficas y la aplicación de modelos matemáticos, es posible identificar patrones que normalmente pasarían desapercibidos. Esto ayuda a las empresas a tomar decisiones basadas en hechos reales, como ajustar inventarios, planificar promociones, administrar mejor los recursos y conocer a sus clientes de forma más profunda.

Este proyecto se desarrolla dentro de este contexto. Su propósito es analizar un conjunto de datos de ventas retail con el fin de identificar tendencias importantes, patrones de compra, factores que explican variaciones en el gasto de los clientes y relaciones entre distintos elementos del comportamiento de compra. Con los resultados obtenidos, sería posible proponer estrategias que mejoren el rendimiento general de una tienda retail.

1.2 Descripción de la empresa analizada

Para el desarrollo de este proyecto se utilizó una empresa ficticia llamada **RetailX**, dedicada a la venta de productos de consumo en tres categorías principales: *Electronics*, *Clothing* y *Beauty*. Cada transacción realizada por un cliente queda registrada con información como: la fecha de compra, el producto adquirido, el precio por unidad, la cantidad comprada, la edad del cliente y el total pagado.

Aunque los datos son sintéticos, fueron creados con la intención de representar situaciones reales que ocurren dentro de una tienda retail común. Esto significa que los patrones, distribuciones y comportamientos que aparecen en el dataset se asemejan mucho a los que podrían encontrarse en un negocio real. Gracias a esto, RetailX se convierte en un buen punto de partida para practicar análisis de datos, comprender mejor la lógica del retail y

explorar cómo las técnicas de Ciencia de Datos pueden aplicarse para mejorar procesos y resultados.

1.3 Importancia del análisis de datos en el sector retail

El análisis de datos juega un papel clave en la industria del retail, ya que permite interpretar de forma clara qué está ocurriendo dentro del negocio. A través de este tipo de análisis, es posible obtener información valiosa que puede transformar la manera en que una empresa opera. Algunos de los beneficios más relevantes incluyen:

- Reconocer qué categorías de productos registran mayores ventas y cuáles tienen menor movimiento.
- Detectar cómo cambian las ventas a lo largo del tiempo, ya sea por temporadas, promociones o situaciones externas.
- Identificar de qué manera influyen características como el género o la edad en los hábitos de compra.
- Encontrar patrones que ayuden a dividir a los clientes en grupos con características similares.
- Analizar qué productos tienen precios más sensibles para los consumidores.
- Realizar predicciones que permitan anticipar periodos de baja o alta actividad.
- Mejorar el control de inventarios mediante el conocimiento de productos que requieren reabastecimiento frecuente.

Toda esta información ayuda a que las empresas puedan planear de manera más estratégica, entender con mayor precisión su mercado y tomar decisiones basadas en datos reales, no en suposiciones.

1.4 Objetivo general del proyecto

El objetivo principal de este proyecto es **realizar un análisis completo de un conjunto de datos de ventas retail** con la finalidad de descubrir patrones de comportamiento, identificar tendencias importantes y analizar relaciones significativas entre diferentes variables, como la edad del cliente, el precio del producto, la cantidad comprada y el total pagado.

Además, se desarrolla un conjunto de modelos de Machine Learning para apoyar la toma de decisiones mediante predicciones y clasificaciones relacionadas con el comportamiento del cliente. Con este objetivo, se aplican técnicas que permiten estimar el monto total de compra y clasificar ciertos tipos de comportamiento mediante análisis supervisados.

El proyecto incluye diversas etapas, como la limpieza del dataset, la creación de nuevas variables, el análisis exploratorio mediante gráficas, el entrenamiento de modelos y la validación final de los resultados para asegurarse de que sean confiables.

1.5 Metodología de Ciencia de Datos empleada

El proyecto fue desarrollado siguiendo una metodología de Ciencia de Datos compuesta por varias fases consecutivas, diseñadas para garantizar un análisis claro, estructurado y confiable. Las etapas utilizadas son las siguientes:

1. **Entendimiento del problema:** Estudio del contexto de la empresa, sus necesidades y los objetivos que se desean alcanzar mediante el análisis.
2. **Recolección y comprensión de los datos:** Obtención del dataset y análisis general de su estructura, sus columnas y su calidad.
3. **Data Wrangling:** Limpieza del dataset, eliminación de duplicados, manejo de valores faltantes, corrección de tipos de datos, transformación de variables, creación de nuevas columnas y normalización.
4. **Análisis Exploratorio de Datos (EDA):** Elaboración de gráficas, estadísticas descriptivas y revisión de relaciones entre variables para entender los patrones ocultos dentro del dataset.
5. **Modelación con Machine Learning:** Entrenamiento de modelos supervisados, como Regresión Lineal y Regresión Logística, para realizar predicciones y clasificaciones basadas en los datos disponibles.
6. **Validación de modelos:** Evaluación del desempeño de los modelos mediante diferentes métricas como RMSE, R^2, matriz de confusión y curva ROC, para determinar su precisión.
7. **Visualizaciones finales:** Representación gráfica de los hallazgos más importantes con el fin de comunicar de manera clara los resultados.
8. **Conclusiones:** Interpretación general de los resultados obtenidos y formulación de recomendaciones que podrían mejorar la operación de una empresa retail.

Gracias a esta metodología, es posible realizar un análisis estructurado que permita comprender mejor el comportamiento de los clientes y las ventas dentro de un negocio del sector retail.

2. Desarrollo del Proyecto

2.1 Entendimiento del Problema

2.1.1 Planteamiento del problema

Las tiendas del sector retail, como supermercados o tiendas departamentales, manejan miles de compras todos los días. Cada venta incluye información como la edad del cliente, el tipo de producto, el precio, la cantidad comprada y la fecha de compra. Aunque estos datos parecen sencillos, entender qué significan realmente y cómo se relacionan entre sí puede ser complicado.

En el caso de la empresa ficticia RetailX, existe mucha información registrada, pero no se ha analizado a profundidad. La empresa no sabe con claridad qué tipo de clientes compran más, qué productos son los más vendidos, ni cómo cambian las ventas a lo largo del año. Sin este análisis, resulta difícil mejorar las ventas o tomar decisiones importantes para el negocio.

El problema principal es que RetailX no tiene una comprensión clara de los factores que influyen en sus ventas. Esto provoca que la empresa no pueda anticiparse a cambios en la demanda, detectar oportunidades de mejora o diseñar estrategias más efectivas. Algunos ejemplos de preguntas que la empresa no puede responder sin un análisis adecuado son:

- ¿La edad del cliente influye en cuánto gasta?
- ¿Qué categoría de productos se vende más?
- ¿En qué meses se registra más actividad de compras?
- ¿Existen comportamientos de compra que permitan agrupar a ciertos tipos de clientes?

2.1.2 Relevancia del problema para la empresa

Para una empresa retail, entender cómo y cuándo se venden sus productos es fundamental para sobrevivir y crecer. RetailX necesita esta información para manejar mejor su inventario, planear promociones, fijar precios adecuados y administrar sus recursos de forma eficiente.

Si la empresa no analiza su información histórica, corre el riesgo de tomar malas decisiones, como comprar productos que no se venden, no identificar temporadas de alta demanda, o aplicar promociones que no generan resultados. En cambio, si RetailX usa sus datos de forma correcta, puede identificar sus productos más importantes, conocer mejor a sus clientes y mejorar sus ganancias.

Además, los gustos y preferencias de los clientes cambian con el tiempo. Si la empresa tiene la capacidad de analizar sus datos, podrá adaptarse más rápido a estos cambios y mantenerse competitiva en el mercado. Por esta razón, comprender el comportamiento de compra no solo es útil, sino necesario.

2.1.3 Justificación del enfoque analítico

Usar un enfoque analítico basado en datos permite convertir grandes cantidades de información en conclusiones claras y útiles. A través de técnicas de Ciencia de Datos, se pueden identificar patrones de compra, analizar cómo cambian las ventas a lo largo del año y descubrir relaciones entre variables que no se pueden ver sin un análisis detallado.

Este tipo de análisis también permite crear modelos que ayudan a predecir lo que podría pasar en el futuro. Por ejemplo, estimar cuánto podría gastar un cliente o qué tipo de compra realizará. Esto ayuda a mejorar la planeación de la empresa y reduce la incertidumbre al tomar decisiones.

Otra ventaja es que este enfoque es objetivo. Las conclusiones se basan en datos reales, no en suposiciones. Esto hace que las decisiones sean más precisas y confiables.

Finalmente, analizar los datos de RetailX permite identificar áreas de oportunidad, como mejorar promociones, ajustar precios, segmentar clientes o reforzar categorías con alta demanda. Por estas razones, realizar un análisis completo del dataset es necesario para mejorar el funcionamiento y las estrategias de la empresa.

2.2 Diagramas del Sistema

En esta sección se presentan y explican los diagramas que permiten entender cómo funciona el sistema de ventas actualmente (**AS-IS**) y cómo se transformaría al incorporar un proceso completo de Ciencia de Datos (**TO-BE**). El propósito es mostrar, de manera visual y sencilla, cómo fluye la información, qué actores intervienen y en qué puntos se generan los datos que luego utilizamos en nuestro análisis.

Estos diagramas también nos ayudan a comprender por qué la empresa necesita implementar un enfoque analítico y cómo este proyecto mejora el proceso actual.

Los diagramas incluidos en esta sección son los siguientes:

- **2.2.1 Diagrama de Contexto (AS-IS)**
- **2.2.2 Diagrama BPMN / DFD del Proceso Actual (AS-IS)**
- **2.2.3 Diagrama de Arquitectura Propuesta (TO-BE)**
- **2.2.4 Flujo de Ciencia de Datos del Proyecto**

Cada uno se acompaña de una explicación clara y detallada para facilitar su comprensión.

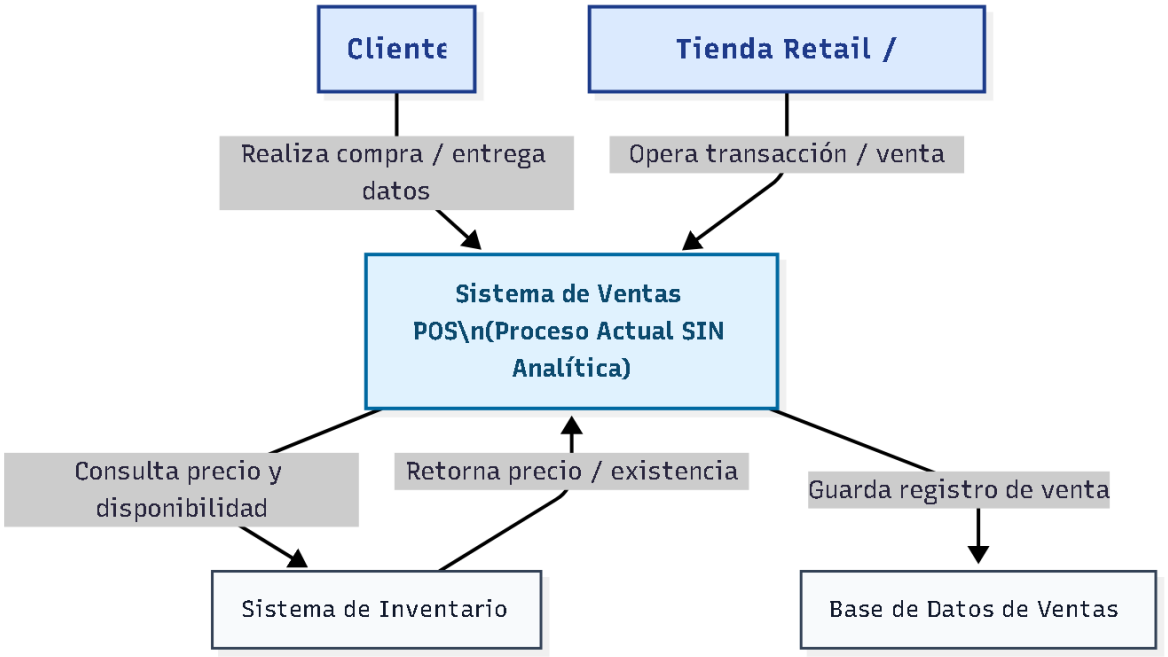
2.2.1 Diagrama de Contexto (AS-IS)

El Diagrama de Contexto muestra de manera general cómo funciona el sistema de ventas actual sin ningún tipo de análisis avanzado. Su objetivo es identificar a los actores principales y cómo se comunican con el sistema de la tienda.

¿Qué representa este diagrama?

- Al **cliente**, quien realiza la compra y genera información con cada transacción.
- A la **tienda**, que opera el sistema de ventas.
- Al **sistema POS**, encargado de registrar las compras.
- A la **base de datos**, donde se almacenan todos los registros.
- Al **sistema de inventario**, que responde sobre disponibilidad y precios.

Aquí todavía no existe análisis de datos, solo un registro básico del funcionamiento diario. Este diagrama permite entender el punto de partida del proyecto.



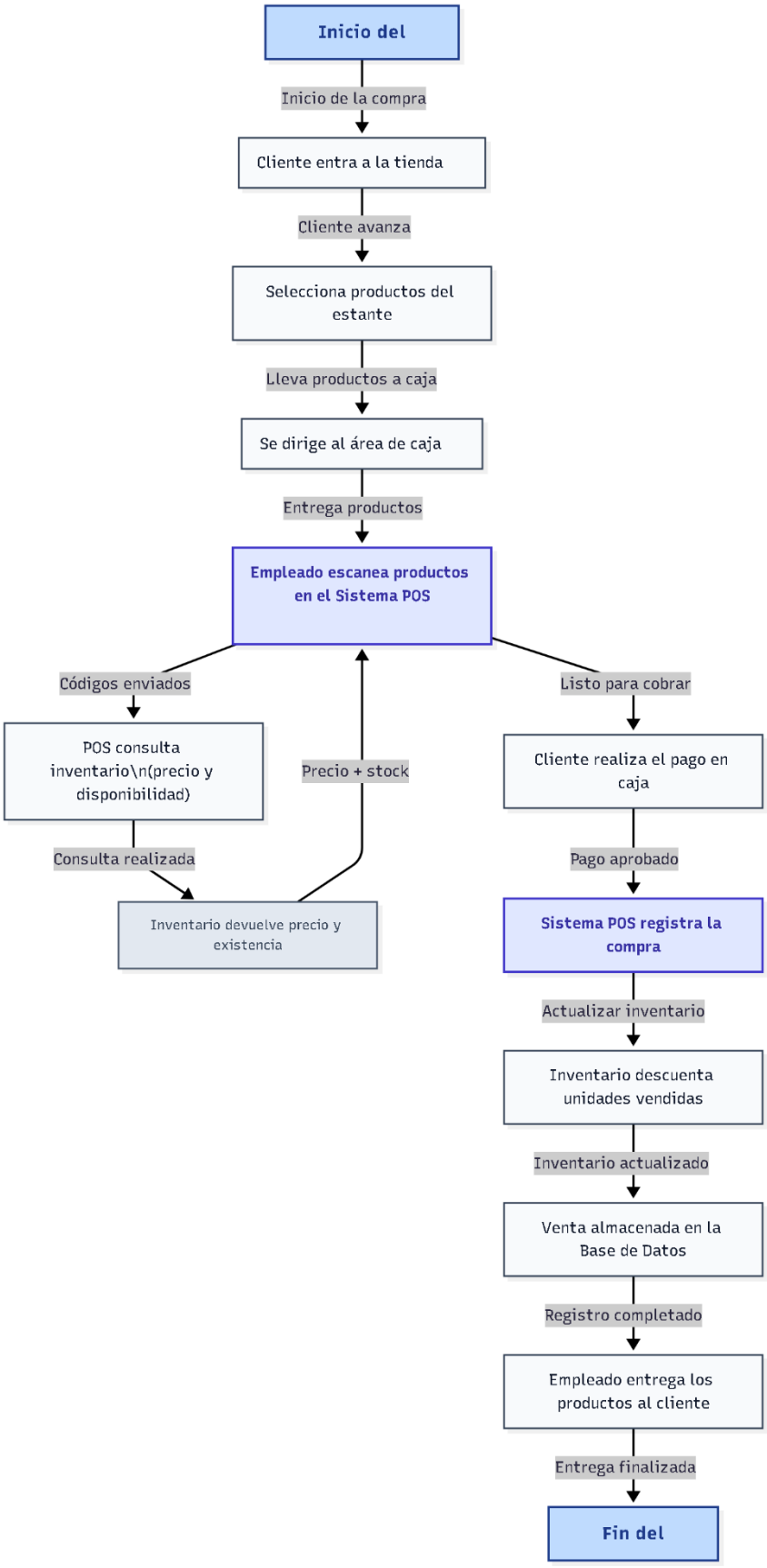
2.2.2 Diagrama BPMN / DFD del Proceso Actual (AS-IS)

Este diagrama explica paso a paso cómo ocurre una compra dentro de la tienda, desde que el cliente entra hasta que recibe sus productos. Representa el proceso operativo actual tal como sucede en la vida real.

¿Qué muestra este diagrama?

- El cliente elige productos del estante.
- Los lleva a la caja.
- El sistema POS escanea los artículos.
- El POS valida precio y existencia consultando al inventario.
- El cliente realiza el pago.
- El inventario se actualiza.
- Finalmente, la venta se guarda en la base de datos.

Este proceso revela en qué momentos se generan datos importantes como precios, cantidades, categorías y montos totales. Este flujo permitirá después comprender cómo se construyó el dataset utilizado en el análisis.


Inicio del
Inicio de la compra
Cliente entra a la tienda
Cliente avanza
Selecciona productos del estante
Lleva productos a caja
Se dirige al área de caja
Entrega productos
Empleado escanea productos en el Sistema POS
Códigos enviados
POS consulta inventario\n(precio y disponibilidad)
Consulta realizada
Inventario devuelve precio y existencia
Precio + stock
Listo para cobrar
Cliente realiza el pago en caja
Pago aprobado
Sistema POS registra la compra
Actualizar inventario
Inventario descuenta unidades vendidas
Inventario actualizado
Venta almacenada en la Base de Datos
Registro completado
Empleado entrega los productos al cliente
Entrega finalizada
Fin del

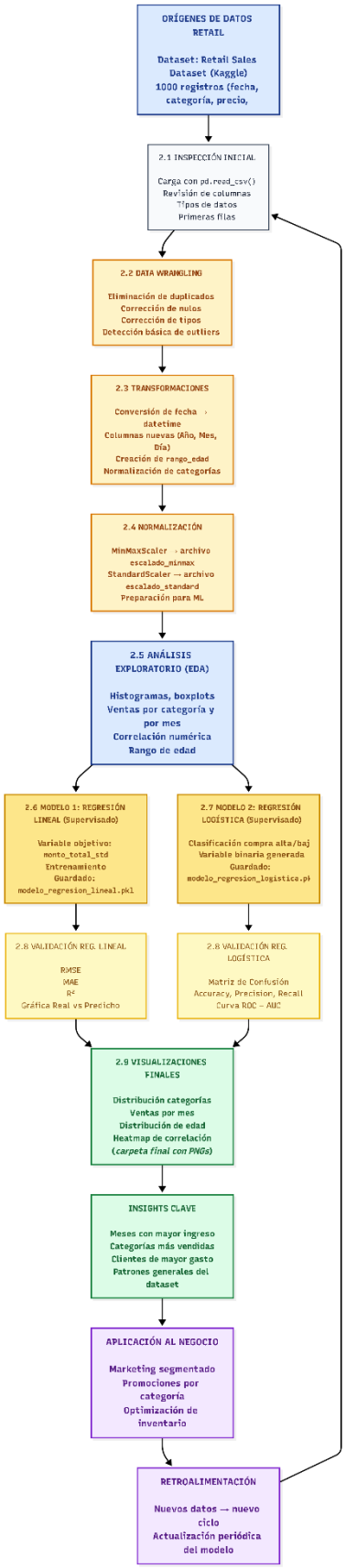
2.2.3 Diagrama de Arquitectura Propuesta (TO-BE)

El diagrama TO-BE muestra cómo se vería el sistema una vez que se integra un proceso formal de Ciencia de Datos. Aquí los datos ya no solo se guardan, sino que se transforman y analizan para generar información útil para la toma de decisiones.

Elementos que incluye:

- La **base de datos**, que sigue siendo la fuente principal de información.
- El proceso de **Data Wrangling**, donde limpiamos y transformamos los datos.
- Los **modelos de Machine Learning** (Regresión Lineal y Logística), que permiten predecir valores y clasificar transacciones.
- Un módulo de **visualización**, donde se generan gráficas y reportes.
- La capa de **insights**, que resume los resultados clave.
- Un ciclo de **retroalimentación**, que permite actualizar el análisis conforme la empresa registra nuevos datos.

Este diagrama representa la mejora completa del sistema y cómo el análisis de datos se integra al proceso del negocio.


ORÍGENES DE DATOS RETAIL
Dataset: Retail Sales Dataset (Kaggle)
1000 registros (fecha, categoría, precio,
2.1 INSPECCIÓN INICIAL
Carga con pd.read_csv()
Revisión de columnas
Tipos de datos
Primeras filas
2.2 DATA WRANGLING
Eliminación de duplicados
Corrección de nulos
Corrección de tipos
Detección básica de outliers
2.3 TRANSFORMACIONES
Conversión de fecha → datetime
Columnas nuevas (Año, Mes, Día)
Creación de rango_edad
Normalización de categorías
2.4 NORMALIZACIÓN
MinMaxScaler → archivo escalado_minmax
StandardScaler → archivo escalado_standard
Preparación para ML
2.5 ANÁLISIS EXPLORATORIO (EDA)
Histogramas, boxplots
Ventas por categoría y por mes
Correlación numérica
Rango de edad
2.6 MODELO 1: REGRESIÓN LINEAL (Supervisado)
Variable objetivo: monto_total_std
Entrenamiento
Guardado: modelo_regresion_lineal.pkl
2.7 MODELO 2: REGRESIÓN LOGÍSTICA (Supervisado)
Clasificación compra alta/baj
Variable binaria generada
Guardado: modelo_regresion_logistica.pk
2.8 VALIDACIÓN REG. LINEAL
RMSE
MAE
R²
Gráfica Real vs Predicho
2.8 VALIDACIÓN REG. LOGÍSTICA
Matriz de Confusión
Accuracy, Precision, Recall
Curva ROC – AUC
2.9 VISUALIZACIONES FINALES
Distribución categorías
Ventas por mes
Distribución de edad
Heatmap de correlación
(carpeta final con PNGs)
INSIGHTS CLAVE
Meses con mayor ingreso
Categorías más vendidas
Clientes de mayor gasto
Patrones generales del dataset
APLICACIÓN AL NEGOCIO
Marketing segmentado
Promociones por categoría
Optimización de inventario
RETROALIMENTACIÓN
Nuevos datos → nuevo ciclo
Actualización periódica del modelo

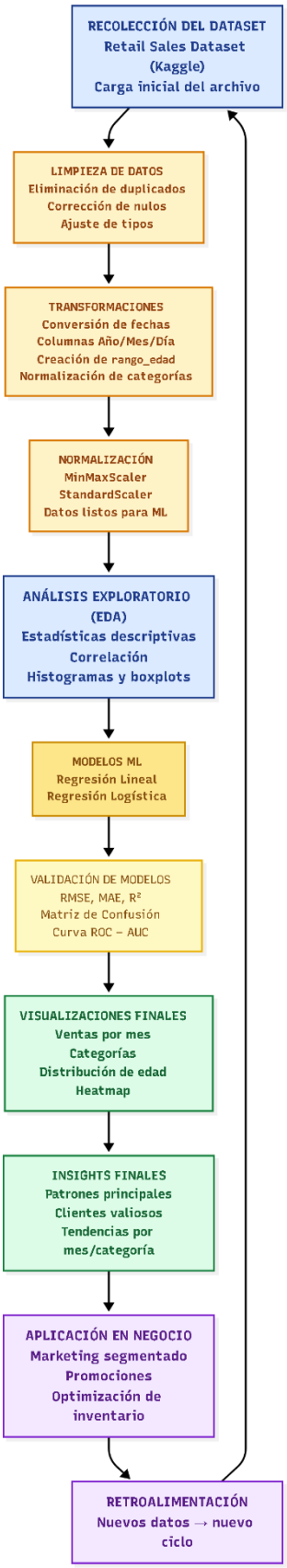
2.2.4 Flujo de Ciencia de Datos

Este diagrama resume todo el proceso que seguimos dentro del proyecto, mostrando cada una de las etapas de forma ordenada. Es una representación visual del ciclo completo de trabajo que se aplicó al dataset.

Etapas del flujo:

- **Recolección del dataset**, donde cargamos el archivo original.
- **Limpieza**, que incluye eliminación de duplicados, corrección de nulos y ajustes de tipos.
- **Transformaciones**, como generar columnas nuevas y normalizar categorías.
- **Normalización**, donde preparamos los datos para los modelos.
- **EDA (Análisis Exploratorio)**, donde se estudian patrones iniciales.
- **Modelos de Machine Learning**, supervisados en este caso.
- **Validación de resultados**, utilizando métricas y gráficas.
- **Visualizaciones finales**, útiles para interpretar los hallazgos.
- **Insights para negocio**, que permiten tomar decisiones.
- **Retroalimentación**, que indica que el proceso puede repetirse con nuevos datos.

Este diagrama nos permite ver en un solo lugar cómo se desarrolló todo el proyecto desde el inicio hasta la generación de conclusiones.


RECOLECCIÓN DEL DATASET
Retail Sales Dataset
(Kaggle)
Carga inicial del archivo
LIMPIEZA DE DATOS
Eliminación de duplicados
Corrección de nulos
Ajuste de tipos
TRANSFORMACIONES
Conversión de fechas
Columnas Año/Mes/Día
Creación de rango_edad
Normalización de categorías
NORMALIZACIÓN
MinMaxScaler
StandardScaler
Datos listos para ML
ANÁLISIS EXPLORATORIO
(EDA)
Estadísticas descriptivas
Correlación
Histogramas y boxplots
MODELOS ML
Regresión Lineal
Regresión Logística
VALIDACIÓN DE MODELOS
RMSE, MAE, R²
Matriz de Confusión
Curva ROC – AUC
VISUALIZACIONES FINALES
Ventas por mes
Categorías
Distribución de edad
Heatmap
INSIGHTS FINALES
Patrones principales
Clientes valiosos
Tendencias por
mes/categoría
APLICACIÓN EN NEGOCIO
Marketing segmentado
Promociones
Optimización de
inventario
RETROALIMENTACIÓN
Nuevos datos → nuevo
ciclo

2.3 Data Wrangling: Preparación y Transformación de Datos

En esta parte del proyecto se explica, paso a paso, todo el proceso que seguimos para preparar el dataset antes de realizar análisis, visualizaciones o modelos de Machine Learning. A este proceso se le conoce como *Data Wrangling*, y comprende actividades como revisar, limpiar, transformar, estandarizar y organizar los datos para su uso correcto. Esta etapa es esencial, porque si los datos contienen errores, valores inconsistentes o estructuras incorrectas, los modelos y conclusiones finales pueden ser poco confiables.

El Data Wrangling para este proyecto se trabajó en **cuatro notebooks principales**, cada uno con una función específica:

- **2.1_Inspeccion_Inicial.ipynb** → Carga del dataset y revisión básica.
- **2.2_Data_Wrangling.ipynb** → Limpieza de nulos, duplicados y tipos de datos.
- **2.3_Transformaciones.ipynb** → Creación de columnas derivadas y ajustes estructurales.
- **2.4_Normalizacion.ipynb** → Escalamiento con MinMaxScaler y StandardScaler.

Cada uno de estos notebooks generó resultados importantes que se emplearon para construir el dataset final.

2.3.1 Recolección del Dataset (Kaggle – Retail Sales Dataset)

El dataset utilizado en este proyecto es el **Retail Sales Dataset**, descargado desde Kaggle. Este archivo contiene **1000 registros** de ventas simuladas en un entorno retail. Aunque es sintético, sus características se parecen mucho a un dataset real, lo que permite practicar análisis, EDA y modelado.

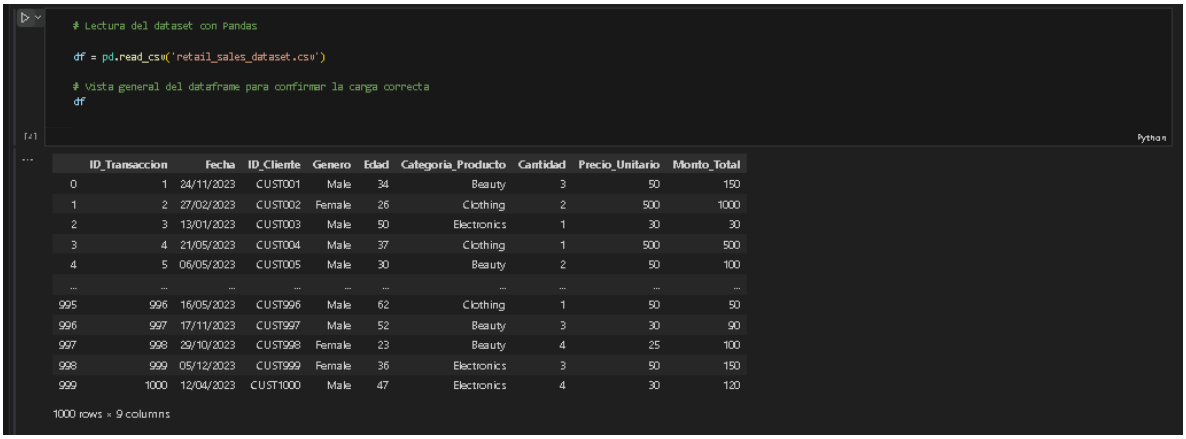
El dataset incluye información clave sobre cada transacción, como:

- Fecha de compra
- Edad del cliente
- Categoría del producto
- Cantidad adquirida
- Precio unitario
- Total pagado

Esta información es suficiente para explorar patrones de compra, comportamientos por edad, tendencias mensuales, comparar categorías y entrenar modelos predictivos.

Código utilizado para cargar el dataset

El código para la carga del archivo se encuentra en el notebook: → **Notebook: 2.1_Inspeccion_Inicial.ipynb**

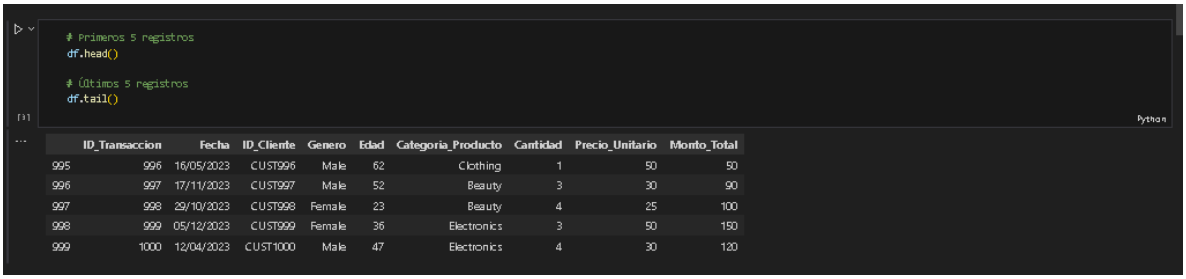


```
# Lectura del dataset con Pandas

df = pd.read_csv('retail_sales_dataset.csv')

# Vista general del dataframe para confirmar la carga correcta
df
```

Python

	ID_Transaccion	Fecha	ID_Cliente	Genero	Edad	Categoria_Producto	Cantidad	Precio_Unitario	Monto_Total
0	1	24/11/2023	CUST001	Male	34	Beauty	3	50	150
1	2	27/02/2023	CUST002	Female	26	Clothing	2	500	1000
2	3	13/01/2023	CUST003	Male	50	Electronics	1	30	30
3	4	21/05/2023	CUST004	Male	37	Clothing	1	500	500
4	5	06/05/2023	CUST005	Male	30	Beauty	2	50	100
...	...	...	...	...	...	...	...	...	...
995	996	16/05/2023	CUST996	Male	62	Clothing	1	50	50
996	997	17/11/2023	CUST997	Male	52	Beauty	3	30	90
997	998	29/10/2023	CUST998	Female	23	Beauty	4	25	100
998	999	05/12/2023	CUST999	Female	36	Electronics	3	50	150
999	1000	12/04/2023	CUST1000	Male	47	Electronics	4	30	120

1000 rows × 9 columns



```
# Primeros 5 registros
df.head()

# Últimos 5 registros
df.tail()
```

Python

	ID_Transaccion	Fecha	ID_Cliente	Genero	Edad	Categoria_Producto	Cantidad	Precio_Unitario	Monto_Total
995	996	16/05/2023	CUST996	Male	62	Clothing	1	50	50
996	997	17/11/2023	CUST997	Male	52	Beauty	3	30	90
997	998	29/10/2023	CUST998	Female	23	Beauty	4	25	100
998	999	05/12/2023	CUST999	Female	36	Electronics	3	50	150
999	1000	12/04/2023	CUST1000	Male	47	Electronics	4	30	120

Descripción de las columnas originales

El dataset contiene las siguientes columnas:

- **Transaction ID**: Identificador único por cada venta.
- **Date**: Fecha en la que ocurrió la transacción.
- **Customer ID**: Código único de cada cliente.
- **Gender**: Género del cliente (Male / Female).
- **Age**: Edad del cliente.
- **Product Category**: Categoría del producto comprado (Electronics, Clothing, Beauty).
- **Quantity**: Cantidad de unidades compradas.
- **Price per Unit**: Precio de cada unidad.
- **Total Amount**: Total pagado por la transacción.

Cada una de estas columnas fue revisada e inspeccionada como parte del proceso inicial. Esta revisión nos permitió detectar problemas comunes como fechas en formato de texto, categorías sin normalizar, rangos amplios de precios y necesidad de crear columnas derivadas.

Organización del proyecto

La carpeta **1_Datos** contiene todos los archivos utilizados durante el proceso. Esta carpeta fue creada desde el inicio del proyecto para mantener un control ordenado y permitir que cualquier etapa sea reproducible únicamente ejecutando los notebooks correspondientes.

Cada archivo dentro de esta carpeta fue **generado directamente desde el código Python**, a lo largo de los notebooks del proyecto. Esto asegura trazabilidad, limpieza en el flujo de trabajo y la posibilidad de reconstruir todas las versiones del dataset sin intervención manual.

Los archivos generados fueron:

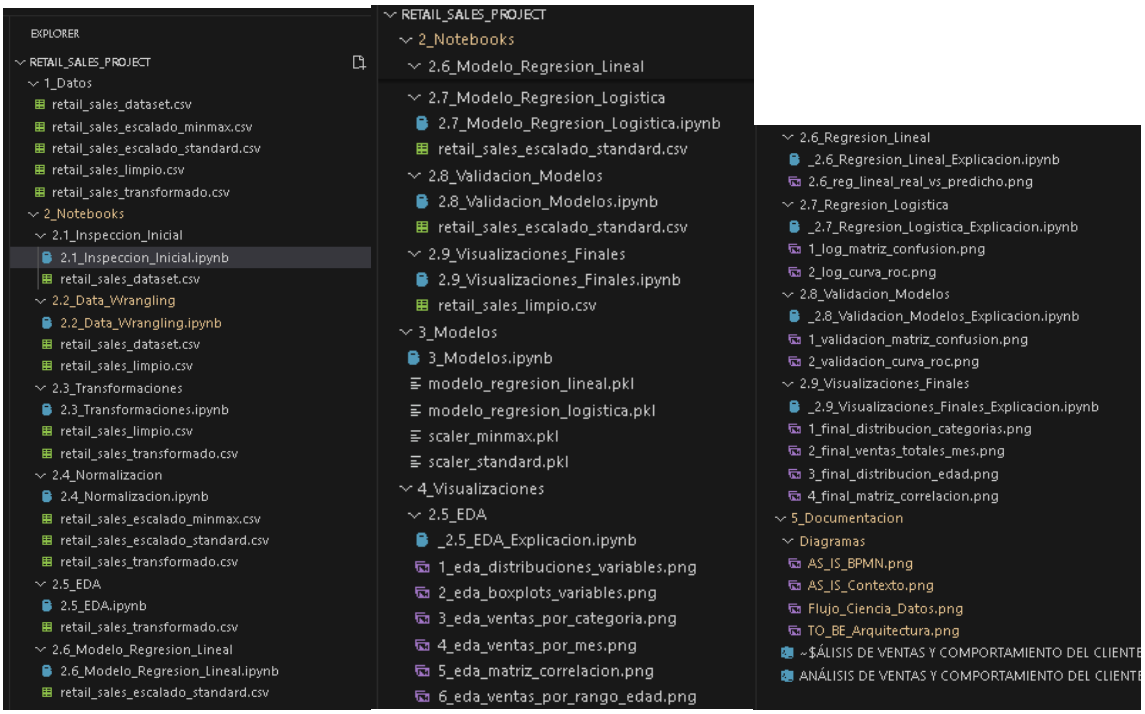
- retail_sales_dataset.csv → Archivo original descargado de Kaggle.
- retail_sales_limpio.csv → Archivo limpio generado en *2.2_Data_Wrangling.ipynb* luego de eliminar duplicados, corregir nulos y ajustar tipos de datos.
- retail_sales_transformado.csv → Archivo creado en *2.3_Transformaciones.ipynb*, donde se añadieron columnas nuevas como Año, Mes, Día y rango_edad.
- retail_sales_escalado_minmax.csv → Dataset escalado con **MinMaxScaler**, generado en *2.4_Normalizacion.ipynb*.
- retail_sales_escalado_standard.csv → Dataset estandarizado con **StandardScaler**, también generado en *2.4_Normalizacion.ipynb*.

Cada una de estas versiones del archivo fue exportada con instrucciones explícitas de Python, lo que permite reconstruir todo el flujo de procesamiento desde cero. Esta estructura permitió mantener un registro claro de cada etapa y versión del dataset.

2.3.2 Direccionamiento de datos y estructura del proyecto

Para asegurar un flujo de trabajo organizado y reproducible, se definió desde el inicio una estructura clara de carpetas dentro del proyecto. Esta estructura permite identificar rápidamente dónde se encuentran los datos originales, los datos procesados y los notebooks utilizados en cada fase del Data Wrangling.

Además, garantiza que cualquier persona pueda ejecutar nuevamente los notebooks y obtener exactamente los mismos resultados.
La organización principal del proyecto es la siguiente (estructura completa del proyecto):



Importancia del direccionamiento de datos

Para evitar errores al cargar los archivos dentro de los notebooks, todas las rutas fueron definidas utilizando **rutas relativas**. Esto permite que el proyecto pueda ejecutarse en cualquier computadora sin necesidad de modificar las rutas manualmente.

```
# Lectura del dataset original

df = pd.read_csv('retail_sales_dataset.csv')
```

Este enfoque asegura que el notebook busque el archivo siempre dentro de la estructura organizada del proyecto.

Además, conforme se avanzaba en las etapas de limpieza, transformación y normalización, se fueron generando nuevas versiones del dataset dentro de la carpeta **1_Datos**, cada una creada directamente desde el código Python.

2.3.3 Inspección inicial del dataset

La inspección inicial del dataset se realizó en el notebook **2.1_Inspeccion_Inicial.ipynb**, cuyo objetivo fue analizar rápidamente la estructura del archivo CSV original para identificar posibles problemas o aspectos relevantes antes de aplicar cualquier limpieza.

El proceso incluyó los siguientes pasos:

1. Carga del dataset

Código utilizado:

```python
# Lectura del dataset con Pandas

df = pd.read_csv('retail_sales_dataset.csv')

# Vista general del dataframe para confirmar la carga correcta
df
```

[2] Python

	ID_Transaccion	Fecha	ID_Cliente	Genero	Edad	Categoria_Producto	Cantidad	Precio_Unitario	Monto_Total
0	1	24/11/2023	CUST001	Male	34	Beauty	3	50	150
1	2	27/02/2023	CUST002	Female	26	Clothing	2	500	1000
2	3	13/01/2023	CUST003	Male	50	Electronics	1	30	30
3	4	21/05/2023	CUST004	Male	37	Clothing	1	500	500
4	5	06/05/2023	CUST005	Male	30	Beauty	2	50	100
...	...	...	...	...	...	...	...	...	...
995	996	16/05/2023	CUST996	Male	62	Clothing	1	50	50
996	997	17/11/2023	CUST997	Male	52	Beauty	3	30	90
997	998	29/10/2023	CUST998	Female	23	Beauty	4	25	100
998	999	05/12/2023	CUST999	Female	36	Electronics	3	50	150
999	1000	12/04/2023	CUST1000	Male	47	Electronics	4	30	120

1000 rows × 9 columns

2. Vista preliminar con df.head()

Este comando muestra las primeras filas del dataset, permitiendo confirmar:

- Que el archivo se cargó correctamente.
- Que las columnas coinciden con lo esperado.
- Que los formatos de datos parecen razonables.

```
# Primeros 5 registros
df.head()

# Últimos 5 registros
df.tail()
```

[3]

	ID_Transaccion	Fecha	ID_Cliente	Genero	Edad	Categoria_Producto	Cantidad	Precio_Unitario	Monto_Total
995	996	16/05/2023	CUST996	Male	62	Clothing	1	50	50
996	997	17/11/2023	CUST997	Male	52	Beauty	3	30	90
997	998	29/10/2023	CUST998	Female	23	Beauty	4	25	100
998	999	05/12/2023	CUST999	Female	36	Electronics	3	50	150
999	1000	12/04/2023	CUST1000	Male	47	Electronics	4	30	120

3. Información general del dataset con df.info()

Este comando permite revisar:

- Tipos de datos por columna.
- Número de valores no nulos.
- Posibles columnas con valores faltantes.

df.info()

```
# Resumen completo del DataFrame
df.info()
```

[6] Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   ID_Transaccion      1000 non-null   int64
 1   Fecha               1000 non-null   object
 2   ID_Cliente          1000 non-null   object
 3   Genero              1000 non-null   object
 4   Edad                1000 non-null   int64
 5   Categoria_Producto  1000 non-null   object
 6   Cantidad            1000 non-null   int64
 7   Precio_Unitario     1000 non-null   int64
 8   Monto_Total         1000 non-null   int64
dtypes: int64(5), object(4)
memory usage: 70.4+ KB
```

4. Revisión de dimensiones del dataset

Se usó df.shape para conocer:

- Número de filas (1000)
- Número de columnas (9)

```
# Dimensiones del dataset
df.shape
```

[4] Python

```
(1000, 9)
```

5. Inspección estadística con df.describe()

Se utilizó para revisar valores mínimos, máximos, promedios y rangos de las variables numéricas.

```
# Estadísticas descriptivas de columnas numéricas del dataset
df[['Edad','Cantidad','Precio_Unitario','Monto_Total']].describe()
```

Python

	Edad	Cantidad	Precio_Unitario	Monto_Total
count	1000.00000	1000.000000	1000.000000	1000.000000
mean	41.39200	2.514000	179.890000	456.000000
std	13.68143	1.132734	189.681356	559.997632
min	18.00000	1.000000	25.000000	25.000000
25%	29.00000	1.000000	30.000000	60.000000
50%	42.00000	3.000000	50.000000	135.000000
75%	53.00000	4.000000	300.000000	900.000000
max	64.00000	4.000000	500.000000	2000.000000

En conjunto, esta inspección inicial permitió detectar aspectos importantes del dataset, como:

- La necesidad de convertir la columna **Date** a formato de fecha.
- La consistencia de valores en **Age**, **Quantity** y **Price per Unit**.
- La ausencia de nulos en la mayoría de las columnas.
- La presencia de categorías bien definidas en las variables categóricas.

Esta revisión fue fundamental para decidir las transformaciones necesarias en las siguientes etapas del Data Wrangling.

2.3.4 Limpieza de Datos (duplicados, nulos, tipos de datos y outliers)

La fase de **limpieza de datos** se realizó de manera sistemática en el notebook:
→ **2.2_Data_Wrangling.ipynb**
En esta etapa se dejó listo el dataset para poder aplicar transformaciones, EDA y modelos, trabajando siempre sobre las columnas con nombres ya estandarizados en español:

- fecha
- genero
- edad
- categoria_producto
- cantidad
- precio_unitario
- monto_total

A continuación se describen, con el mismo código que aparece en el notebook, todos los pasos de limpieza.

1. Carga inicial y verificación rápida
Antes de limpiar, se cargó nuevamente el archivo original dentro del notebook.

Código utilizado

```
# Importación de librerías necesarias para el Wrangling

import pandas as pd
import numpy as np
```

Python

```
# Lectura del dataset original

df = pd.read_csv('retail_sales_dataset.csv')

# Vista general para confirmar que el dataset se cargó correctamente
df.head()
```

Python

	ID_Transaccion	Fecha	ID_Cliente	Genero	Edad	Categoria_Producto	Cantidad	Precio_Unitario	Monto_Total
0	1	24/11/2023	CUST001	Male	34	Beauty	3	50	150
1	2	27/02/2023	CUST002	Female	26	Clothing	2	500	1000
2	3	13/01/2023	CUST003	Male	50	Electronics	1	30	30
3	4	21/05/2023	CUST004	Male	37	Clothing	1	500	500
4	5	06/05/2023	CUST005	Male	30	Beauty	2	50	100

```
# Dimensiones del dataset
df.shape
```

Python

```
(1000, 9)
```

Esta primera revisión permitió confirmar que el archivo se leyó correctamente, que la estructura coincidía con la esperada y que se contaba con **1000 registros y 9 columnas**.

2. Estandarización de nombres de columnas

Para trabajar de forma consistente en todo el proyecto, se normalizaron los nombres de las columnas a minúsculas y con guiones bajos.

Código utilizado

```python
# Renombrar columnas a formato estandarizado

df.columns = df.columns.str.lower().str.replace(' ', '_')

# Verificación del cambio de nombres
df.columns
```

```
Index(['id_transaccion', 'fecha', 'id_cliente', 'genero', 'edad',
       'categoria_producto', 'cantidad', 'precio_unitario', 'monto_total'],
      dtype='object')
```

Con esto se pasó, por ejemplo, de Price per Unit a precio_unitario, y de Total Amount a monto_total.

3. Conversión de tipos de datos (columna fecha)

Una columna clave para el análisis temporal es la fecha. En el archivo original venía como texto, por lo que se convirtió a tipo datetime.

```python
# Conversión de la columna 'fecha' a formato datetime

df['fecha'] = pd.to_datetime(df['fecha'], errors='coerce')

# Verificar nuevamente los tipos de datos
df.dtypes
```

```
C:\Users\arion\AppData\Local\Temp\ipykernel_14972\4242823221.py:3: UserWarning: Parsing dates in %d/%m/%Y format when dayfirst=F
  df['fecha'] = pd.to_datetime(df['fecha'], errors='coerce')
```

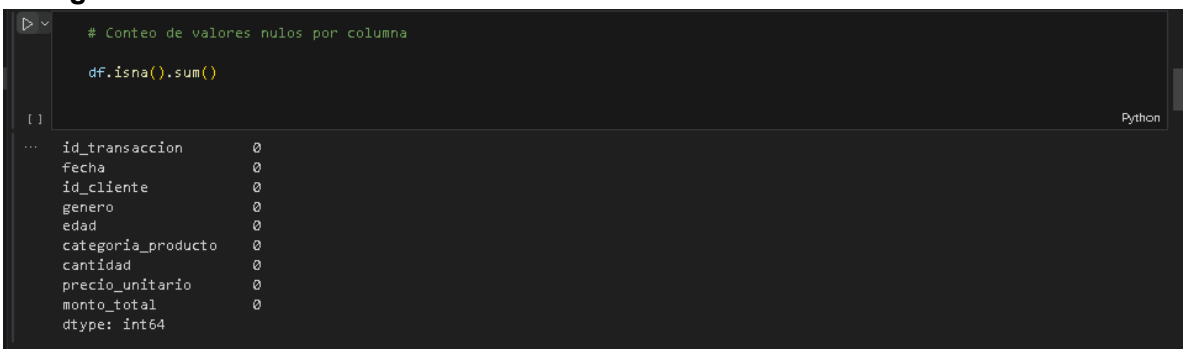
```
id_transaccion               int64
fecha               datetime64[ns]
id_cliente                  object
genero                      object
edad                         int64
categoria_producto          object
cantidad                     int64
precio_unitario              int64
monto_total                  int64
dtype: object
```

El parámetro errors='coerce' asegura que cualquier valor con formato inválido se convierta en NaT (equivalente a nulo de fecha), lo cual es más sencillo de detectar y tratar.

4. Revisión de valores nulos

Después de ajustar los tipos, se revisó la existencia de valores faltantes en todas las columnas.

Código utilizado



```
# Conteo de valores nulos por columna

df.isna().sum()
```

Python

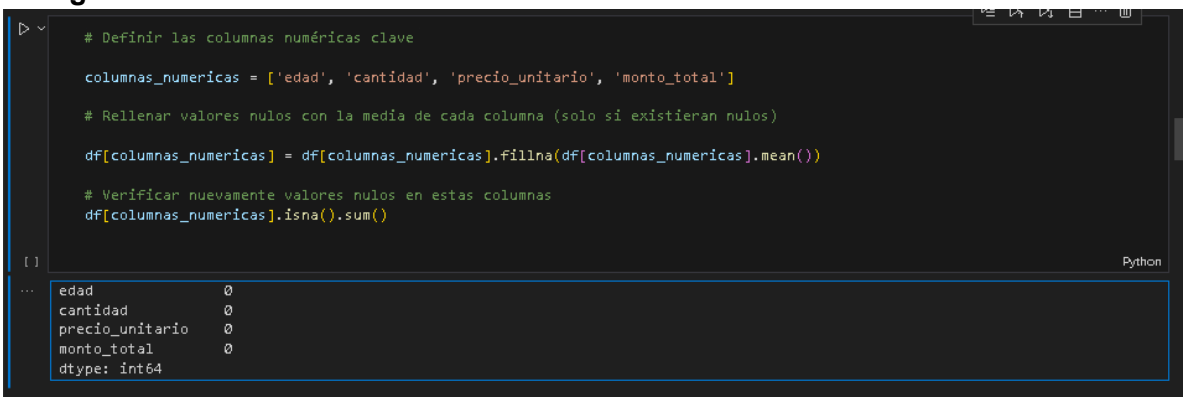
```
id_transaccion       0
fecha                0
id_cliente           0
genero               0
edad                 0
categoria_producto   0
cantidad             0
precio_unitario      0
monto_total          0
dtype: int64
```

En este dataset concreto **no se detectaron nulos relevantes**, pero aun así se dejó preparado un bloque de código para el caso de que aparecieran valores faltantes en las columnas numéricas.

5. Definición de columnas numéricas y estrategia para nulos

Se definió un arreglo con las columnas numéricas clave y se dejó lista la lógica para rellenar nulos con la media, en caso de que existan.

Código utilizado



```
# Definir las columnas numéricas clave

columnas_numericas = ['edad', 'cantidad', 'precio_unitario', 'monto_total']

# Rellenar valores nulos con la media de cada columna (solo si existieran nulos)

df[columnas_numericas] = df[columnas_numericas].fillna(df[columnas_numericas].mean())

# Verificar nuevamente valores nulos en estas columnas
df[columnas_numericas].isna().sum()
```

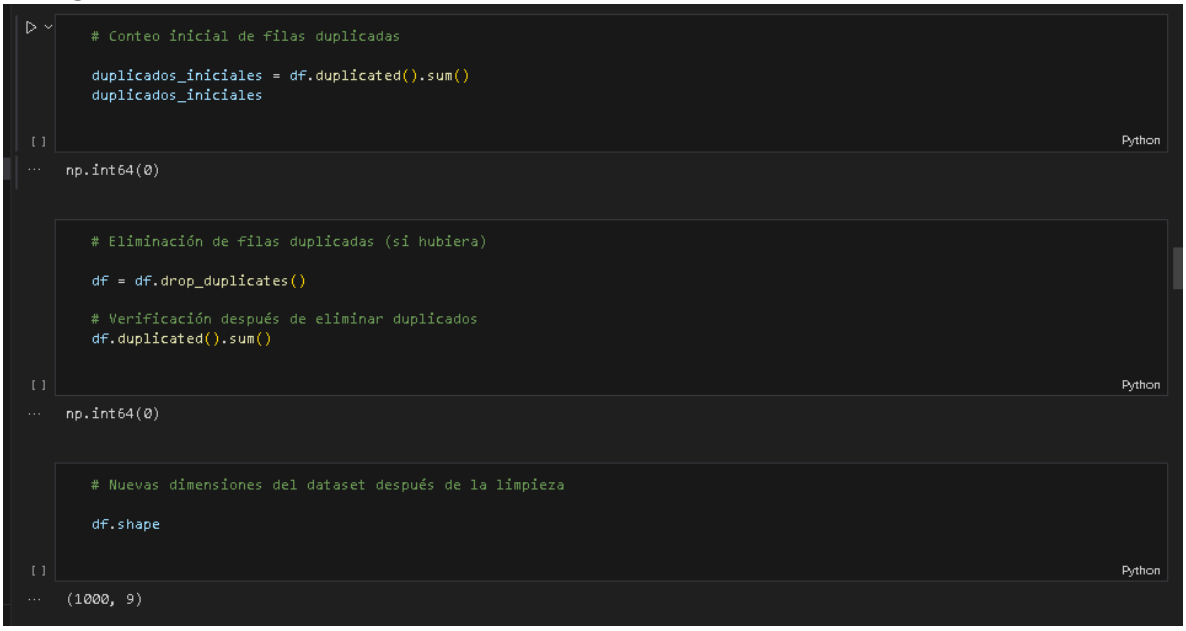
Python

```
edad              0
cantidad          0
precio_unitario   0
monto_total       0
dtype: int64
```

Aunque en la práctica los conteos resultaron en cero, este paso asegura que el flujo sea robusto si en el futuro se incorporan nuevos datos con nulos.

6. Detección y eliminación de duplicados

Se verificó si existían filas repetidas y, en caso de haberlas, se eliminaron.
Código utilizado



En este dataset **no se encontraron filas duplicadas**, por lo que el tamaño del DataFrame se mantuvo en 1000 registros. Sin embargo, documentar este paso es importante para la reproducibilidad del flujo.

7. Revisión de coherencia en columnas categóricas

También se revisaron los valores presentes en columnas categóricas como genero y categoria_producto, para asegurarse de que no hubiera etiquetas inconsistentes.

```python
# Ventas agregadas por categoría de producto

resumen_categoria = df.groupby('categoria_producto', as_index=False)['monto_total'].agg(
    ['count', 'sum', 'mean']
)

resumen_categoria
```

Python

	categoria_producto	count	sum	mean
0	Beauty	307	143515	467.475570
1	Clothing	351	155580	443.247863
2	Electronics	342	156905	458.786550

```python
# Ventas agregadas por género

resumen_genero = df.groupby('genero', as_index=False)['monto_total'].agg(
    ['count', 'sum', 'mean']
)

resumen_genero
```

Python

	genero	count	sum	mean
0	Female	510	232840	456.549020
1	Male	490	223160	455.428571

Se confirmó que:

- genero solo contiene valores válidos como Male y Female.
- categoria_producto incluye categorías como Clothing, Electronics y Beauty.

No fue necesario corregir etiquetas, pero este control asegura calidad en el análisis posterior.

8. Estadísticas y revisión de posibles outliers

Aunque la detección visual de outliers se desarrolla con mayor detalle en la sección **2.5 EDA**, en este notebook se generó un resumen estadístico de las variables numéricas.

Código utilizado

```python
# Estadísticas descriptivas resumidas para columnas numéricas

estadisticas_numericas = df[['edad', 'cantidad', 'precio_unitario', 'monto_total']].agg(
    ['min', 'max', 'mean', 'median', 'std', 'var']
)

estadisticas_numericas
```

Python

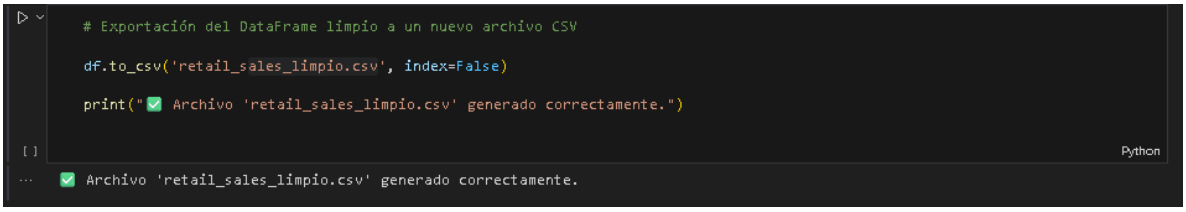
	edad	cantidad	precio_unitario	monto_total
min	18.000000	1.000000	25.000000	25.000000
max	64.000000	4.000000	500.000000	2000.000000
mean	41.392000	2.514000	179.890000	456.000000
median	42.000000	3.000000	50.000000	135.000000
std	13.681430	1.132734	189.681356	559.997632
var	187.181518	1.283087	35979.016917	313597.347347

Este resumen permitió verificar que:

- Los rangos de edad son coherentes con un público adulto.
- cantidad se mantiene entre 1 y 4 unidades.
- precio_unitario y monto_total tienen valores altos, pero en proporciones esperadas para compras de productos de mayor precio.

9. Exportación del dataset limpio

Finalmente, se generó el archivo **retail_sales_limpio.csv**, que será utilizado en los notebooks posteriores de transformaciones, normalización y modelos.



Este archivo se guarda dentro de la estructura del proyecto y se copia a la carpeta **1_Datos** para mantener una versión única de referencia.

Conclusión de la limpieza

Con todos estos pasos, el dataset quedó:

- Sin duplicados.
- Sin valores nulos en columnas críticas.
- Con tipos de datos correctos (especialmente fecha).
- Con nombres de columnas estandarizados en español.
- Con estadísticas numéricas revisadas para detectar posibles valores atípicos.

A partir de este punto, el archivo retail_sales_limpio.csv se utiliza como base en el notebook **2.3_Transformaciones.ipynb**, donde se generan nuevas columnas derivadas y se preparan los datos para el análisis exploratorio y los modelos de Machine Learning.

2.3.5 Transformaciones Aplicadas (variables derivadas, fechas, categorías)

En esta etapa se aplicaron todas las **transformaciones** necesarias sobre el archivo limpio retail_sales_limpio.csv para enriquecer el dataset y prepararlo para la **normalización**, el **análisis exploratorio (EDA)** y los **modelos de Machine Learning**.

Todo este trabajo se realizó en el notebook:
→ **2.3_Transformaciones.ipynb**
Las transformaciones que se documentan aquí siguen exactamente el código utilizado en el proyecto y se organizan en los mismos bloques lógicos que se trabajaron en clase.

1. Carga del dataset limpio
El notebook comienza importando la librería principal y cargando el archivo limpio generado en la etapa de Data Wrangling.

```
# Importar librerías principales
import pandas as pd

# Cargar el dataset limpio (está en la misma carpeta que este notebook)
df = pd.read_csv("retail_sales_limpio.csv")

# Vista rápida de los primeros registros
df.head()
```

[5] Python

	id_transaccion	fecha	id_cliente	genero	edad	categoria_producto	cantidad	precio_unitario	monto_total
0	1	2023-11-24	CUST001	Male	34	Beauty	3	50	150
1	2	2023-02-27	CUST002	Female	26	Clothing	2	500	1000
2	3	2023-01-13	CUST003	Male	50	Electronics	1	30	30
3	4	2023-05-21	CUST004	Male	37	Clothing	1	500	500
4	5	2023-05-06	CUST005	Male	30	Beauty	2	50	100

2. Revisión de la estructura del DataFrame
Antes de transformar, se revisa la estructura básica del DataFrame para confirmar nombres de columnas, dimensiones y tipos de datos.
Código utilizado

```
# Ver nombres de columnas
df.columns
```

[6] Python

```
Index(['id_transaccion', 'fecha', 'id_cliente', 'genero', 'edad',
       'categoria_producto', 'cantidad', 'precio_unitario', 'monto_total'],
      dtype='object')
```

```
# Información general del DataFrame
df.info()
```

[7] Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id_transaccion      1000 non-null   int64
 1   fecha               1000 non-null   object
 2   id_cliente          1000 non-null   object
 3   genero              1000 non-null   object
 4   edad                1000 non-null   int64
 5   categoria_producto  1000 non-null   object
 6   cantidad            1000 non-null   int64
 7   precio_unitario     1000 non-null   int64
 8   monto_total         1000 non-null   int64
dtypes: int64(5), object(4)
memory usage: 70.4+ KB
```

Descripción:

- Se valida que las columnas estén correctamente nombradas en español.
- Se confirma el número de filas y columnas.
- Se revisa el tipo de dato de cada columna.

3. Conversión de tipos de datos

En esta sección se asegura que la columna de fecha esté en el tipo correcto y que las columnas numéricas puedan usarse sin problemas en operaciones matemáticas.

```python
# Conversión de 'fecha' a tipo datetime
df['fecha'] = pd.to_datetime(df['fecha'], errors='coerce')

# Conversión de columnas numéricas a tipo numérico (por si hubiera texto raro)
columnas_numericas = ['edad', 'cantidad', 'precio_unitario', 'monto_total']

df[columnas_numericas] = df[columnas_numericas].apply(pd.to_numeric, errors='coerce')

# Verificación final de tipos
df.dtypes
```

[8] Python

```
id_transaccion                int64
fecha                datetime64[ns]
id_cliente                   object
genero                       object
edad                          int64
categoria_producto           object
cantidad                      int64
precio_unitario               int64
monto_total                   int64
dtype: object
```

Descripción:

- fecha se convierte a datetime, lo que permite crear variables temporales.
- edad, cantidad, precio_unitario y monto_total se aseguran como numéricas.

4. Creación de variables temporales

A partir de la columna fecha se generan nuevas columnas que permiten analizar el comportamiento de las ventas en el tiempo.

```python
# Crear columnas con partes de la fecha
df['anio'] = df['fecha'].dt.year
df['mes'] = df['fecha'].dt.month
df['dia'] = df['fecha'].dt.day

# Nombre del mes (ej. January, February...)
df['nombre_mes'] = df['fecha'].dt.month_name()

# Vista rápida de las nuevas columnas
df[['fecha', 'anio', 'mes', 'dia', 'nombre_mes']].head()
```

[9] Python

	fecha	anio	mes	dia	nombre_mes
0	2023-11-24	2023	11	24	November
1	2023-02-27	2023	2	27	February
2	2023-01-13	2023	1	13	January
3	2023-05-21	2023	5	21	May
4	2023-05-06	2023	5	6	May

5. Segmentación de clientes por rango de edad

Para facilitar el análisis por tipo de cliente, se crea una nueva columna llamada rango_edad a partir de la columna edad.

```python
# Función auxiliar para asignar rango de edad
def clasificar_edad(valor):
    if valor < 25:
        return 'Joven'
    elif valor < 45:
        return 'Adulto'
    else:
        return 'Mayor'

# Crear nueva columna con rango de edad
df['rango_edad'] = df['edad'].apply(clasificar_edad)

df[['edad', 'rango_edad']].head()
```

[10] Python

	edad	rango_edad
0	34	Adulto
1	26	Adulto
2	50	Mayor
3	37	Adulto
4	30	Adulto

Descripción de los rangos:

- **Joven**: menor de 25 años.
- **Adulto**: de 25 a 44 años.
- **Mayor**: 45 años o más.

6. Limpieza y normalización de texto

En esta parte se normalizan las columnas de texto para evitar problemas al agrupar o filtrar datos.

```python
# Normalizar el género (mayúsculas sin espacios)
df['genero'] = df['genero'].astype(str).str.strip().str.upper()

# Normalizar la categoría de producto (tipo título)
df['categoria_producto'] = df['categoria_producto'].astype(str).str.strip().str.title()

# Asegurar que el ID de cliente no tenga espacios
df['id_cliente'] = df['id_cliente'].astype(str).str.strip().str.replace(" ", "")

df[['genero', 'categoria_producto', 'id_cliente']].head()
```

[11] Python

	genero	categoria_producto	id_cliente
0	MALE	Beauty	CUST001
1	FEMALE	Clothing	CUST002
2	MALE	Electronics	CUST003
3	MALE	Clothing	CUST004
4	MALE	Beauty	CUST005

Descripción:

- genero queda como MALE o FEMALE.
- categoria_producto queda como Electronics, Clothing, Beauty, etc.
- id_cliente se limpia de espacios para que sea un identificador consistente.

7. Verificación de consistencia entre Cantidad, Precio Unitario y Monto Total

Se valida que el monto_total sea coherente con la multiplicación de cantidad por precio_unitario.

```python
# Calcular el monto teórico basado en cantidad * precio_unitario
df['monto_calculado'] = df['cantidad'] * df['precio_unitario']

# Diferencia entre el monto de la columna original y el calculado
df['diferencia_monto'] = df['monto_total'] - df['monto_calculado']

# Vista rápida de las columnas involucradas
df[['cantidad', 'precio_unitario', 'monto_total', 'monto_calculado', 'diferencia_monto']].head()
```

[12] Python

	cantidad	precio_unitario	monto_total	monto_calculado	diferencia_monto
0	3	50	150	150	0
1	2	500	1000	1000	0
2	1	30	30	30	0
3	1	500	500	500	0
4	2	50	100	100	0

Descripción:

- monto_calculado permite comprobar la lógica del dataset.
- diferencia_monto ayuda a detectar posibles errores.

En este caso, las diferencias son nulas o mínimas (posibles redondeos), por lo que se considera que el dataset es coherente.

8. Clasificación de transacciones por nivel de compra

Se crean dos variables de negocio relacionadas con el monto de la compra:

- nivel_compra
- es_compra_alta

Estas serán muy útiles para la parte de clasificación con Regresión Logística.

```python
# Función para clasificar el nivel de compra
def clasificar_compra(monto):
    if monto < 200:
        return 'Baja'
    elif monto < 800:
        return 'Media'
    else:
        return 'Alta'

# Crear columna con nivel de compra
df['nivel_compra'] = df['monto_total'].apply(clasificar_compra)

# Crear columna booleana para compras altas
df['es_compra_alta'] = df['nivel_compra'] == 'Alta'

df[['monto_total', 'nivel_compra', 'es_compra_alta']].head()
```

[13] Python

	monto_total	nivel_compra	es_compra_alta
0	150	Baja	False
1	1000	Alta	True
2	30	Baja	False
3	500	Media	False
4	100	Baja	False

Descripción de niveles:

- **Baja**: monto_total < 200
- **Media**: 200 ≤ monto_total < 800
- **Alta**: monto_total ≥ 800

9. Vista general del DataFrame transformado

En este punto el dataset ya incluye todas las variables nuevas y las correcciones realizadas.

```python
# Vista general de algunas columnas clave
df[['id_transaccion',
    'fecha',
    'id_cliente',
    'genero',
    'edad',
    'rango_edad',
    'categoria_producto',
    'cantidad',
    'precio_unitario',
    'monto_total',
    'nivel_compra',
    'es_compra_alta']].head()
```

[14] Python

	id_transaccion	fecha	id_cliente	genero	edad	rango_edad	categoria_producto	cantidad	precio_unitario	monto_total	nivel_compra	es_comp
0	1	2023-11-24	CUST001	MALE	34	Adulto	Beauty	3	50	150	Baja	
1	2	2023-02-27	CUST002	FEMALE	26	Adulto	Clothing	2	500	1000	Alta	
2	3	2023-01-13	CUST003	MALE	50	Mayor	Electronics	1	30	30	Baja	
3	4	2023-05-21	CUST004	MALE	37	Adulto	Clothing	1	500	500	Media	
4	5	2023-05-06	CUST005	MALE	30	Adulto	Beauty	2	50	100	Baja	

```python
# Estadísticas descriptivas de las columnas numéricas después de las transformaciones
df[['edad', 'cantidad', 'precio_unitario', 'monto_total']].describe()
```

[15] Python

	edad	cantidad	precio_unitario	monto_total
count	1000.00000	1000.000000	1000.000000	1000.000000
mean	41.39200	2.514000	179.890000	456.000000
std	13.68143	1.132734	189.681356	559.997632
min	18.00000	1.000000	25.000000	25.000000
25%	29.00000	1.000000	30.000000	60.000000
50%	42.00000	3.000000	50.000000	135.000000
75%	53.00000	4.000000	300.000000	900.000000
max	64.00000	4.000000	500.000000	2000.000000

10. Exportación del dataset transformado

Finalmente, se guarda el DataFrame resultante en un nuevo archivo CSV, que será la base de las siguientes etapas.

```python
# Exportar el DataFrame transformado a un nuevo archivo CSV
df.to_csv("retail_sales_transformado.csv", index=False)

print("✅ Archivo 'retail_sales_transformado.csv' generado correctamente.")
```

[16] Python

```
✅ Archivo 'retail_sales_transformado.csv' generado correctamente.
```

En esta etapa se realizaron todas las transformaciones necesarias sobre el archivo retail_sales_limpio.csv para enriquecer el dataset y prepararlo para el análisis y los modelos de Machine Learning. Los principales resultados fueron:

- Conversión correcta de tipos de datos, especialmente la columna fecha.
- Creación de variables temporales (anio, mes, dia, nombre_mes).
- Segmentación de clientes por rango_edad.
- Normalización de texto en genero, categoria_producto e id_cliente.
- Verificación de consistencia entre cantidad, precio_unitario y monto_total mediante monto_calculado y diferencia_monto.
- Definición de variables de negocio (nivel_compra, es_compra_alta) que se usarán para el modelo de clasificación.
- Generación del archivo final **retail_sales_transformado.csv**, que servirá como entrada para la **Normalización (2.4)**, el **EDA (2.5)** y los **modelos de Machine Learning**.

De esta forma, se cumple el apartado de **Transformaciones** dentro de la metodología de Ciencia de Datos solicitada en el proyecto.

2.3.6 Normalización y Escalamiento (MinMaxScaler y StandardScaler)

La normalización es una fase fundamental dentro del proceso de preparación de datos, especialmente cuando se trabajará con modelos de Machine Learning. Muchos algoritmos son sensibles a las diferencias de escala entre variables, lo que puede provocar resultados incorrectos o modelos sesgados. En esta etapa se aplicó un proceso formal de normalización sobre las columnas numéricas del dataset retail, siguiendo la metodología vista en clase.

Esta parte del proyecto fue desarrollada en el notebook: → **2.4_Normalizacion.ipynb**

El objetivo principal es garantizar que todas las variables numéricas tengan una escala comparable para evitar que atributos con valores grandes, como *monto_total*, dominen el comportamiento del modelo frente a valores pequeños como *cantidad*.

Para lograrlo, se utilizaron dos técnicas ampliamente aplicadas en ciencia de datos:

- **Min-Max Scaler** (normalización): Lleva los valores al rango de 0 a 1.
- **Standard Scaler** (estandarización): Ajusta los valores a media 0 y desviación estándar 1.

Ambos archivos generados en esta etapa servirán como insumos directos para la fase de modelado.

1. Importación de librerías y carga del dataset transformado

El proceso inicia con la carga del archivo **retail_sales_transformado.csv**, creado previamente en la etapa de transformaciones.

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Cargar dataset transformado (generado en 2.3)
df = pd.read_csv("retail_sales_transformado.csv")

df.head()
```

[1] Python

	id_transaccion	fecha	id_cliente	genero	edad	categoria_producto	cantidad	precio_unitario	monto_total	anio	mes	dia	nombre_mes	rang
0	1	2023-11-24	CUST001	MALE	34	Beauty	3	50	150	2023	11	24	November	
1	2	2023-02-27	CUST002	FEMALE	26	Clothing	2	500	1000	2023	2	27	February	
2	3	2023-01-13	CUST003	MALE	50	Electronics	1	30	30	2023	1	13	January	
3	4	2023-05-21	CUST004	MALE	37	Clothing	1	500	500	2023	5	21	May	
4	5	2023-05-06	CUST005	MALE	30	Beauty	2	50	100	2023	5	6	May	

2. Selección de columnas numéricas a normalizar

En esta fase únicamente se seleccionan las columnas numéricas que realmente requieren normalización. Este criterio se basa en su uso posterior en modelos de Machine Learning y en la necesidad de mantener relaciones de escala equilibradas.

Columnas seleccionadas:

- edad
- cantidad
- precio_unitario
- monto_total

Código utilizado:

Ver tipos de datos

```
df.dtypes
```

[2] Python

```
id_transaccion         int64
fecha                 object
id_cliente            object
genero                object
edad                   int64
categoria_producto    object
cantidad               int64
precio_unitario        int64
monto_total            int64
anio                   int64
mes                    int64
dia                    int64
nombre_mes            object
rango_edad            object
monto_calculado        int64
diferencia_monto       int64
nivel_compra          object
es_compra_alta          bool
dtype: object
```

Selección de columnas numéricas a normalizar

Esto facilita aplicar los escaladores sin afectar otras columnas.

Generate | Code | Markdown

```
columnas_numericas = ['edad', 'cantidad', 'precio_unitario', 'monto_total']

df_numerico = df[columnas_numericas].copy()

df_numerico.head()
```

[3] Python

	edad	cantidad	precio_unitario	monto_total
0	34	3	50	150
1	26	2	500	1000
2	50	1	30	30
3	37	1	500	500
4	30	2	50	100

3. Aplicar Min-Max Scaler (0 a 1)

El método Min-Max reescala los valores de cada columna para que todos queden entre **0 y 1**, conservando la proporción original de la distribución.

Antes de mostrar la fórmula, es importante entender su lógica: el valor original se ajusta tomando en cuenta su distancia respecto al mínimo y máximo de la columna. Esto conserva la forma de la distribución pero la compacta en un rango uniforme.

Fórmula usada:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
scaler_minmax = MinMaxScaler()

df_minmax = scaler_minmax.fit_transform(df_numerico)

df_minmax = pd.DataFrame(df_minmax, columns=[col + '_minmax' for col in columnas_numericas])

df_minmax.head()
```

[4] Python

	edad_minmax	cantidad_minmax	precio_unitario_minmax	monto_total_minmax
0	0.347826	0.666667	0.052632	0.063291
1	0.173913	0.333333	1.000000	0.493671
2	0.695652	0.000000	0.010526	0.002532
3	0.413043	0.000000	1.000000	0.240506
4	0.260870	0.333333	0.052632	0.037975

4. Aplicar Standard Scaler (Z-Score)

Antes de ver la fórmula, es importante entender que este método mide cuántas desviaciones estándar está un valor por encima o por debajo de la media. Esto permite comparar valores de diferentes escalas en una misma magnitud.

Este método ajusta cada variable de forma que todas tengan:

- media = 0
- desviación estándar = 1

Esto permite que cada variable contribuya en igualdad dentro de modelos lineales y basados en gradiente.

Fórmula usada:

$$X_{std} = \frac{X - \mu}{\sigma}$$

```python
scaler_std = StandardScaler()

df_standard = scaler_std.fit_transform(df_numerico)

df_standard = pd.DataFrame(df_standard, columns=[col + '_std' for col in columnas_numericas])

df_standard.head()
```

[5] Python

	edad_std	cantidad_std	precio_unitario_std	monto_total_std
0	-0.540565	0.429265	-0.685123	-0.546704
1	-1.125592	-0.453996	1.688464	0.971919
2	0.629489	-1.337258	-0.790615	-0.761098
3	-0.321180	-1.337258	1.688464	0.078611
4	-0.833078	-0.453996	-0.685123	-0.636035

5. Unión de resultados con el DataFrame original

Luego de generar las dos versiones escaladas, se integran como columnas adicionales dentro del DataFrame original. Esto permite mantener toda la información consolidada para su uso posterior.

```python
df_normalizado = pd.concat([df, df_minmax, df_standard], axis=1)

df_normalizado.head()
```

[6] Python

	id_transaccion	fecha	id_cliente	genero	edad	categoria_producto	cantidad	precio_unitario	monto_total	anio	...	nivel_compra	es_compra_
0	1	2023-11-24	CUST001	MALE	34	Beauty	3	50	150	2023	...	Baja	F:
1	2	2023-02-27	CUST002	FEMALE	26	Clothing	2	500	1000	2023	...	Alta	T
2	3	2023-01-13	CUST003	MALE	50	Electronics	1	30	30	2023	...	Baja	F:
3	4	2023-05-21	CUST004	MALE	37	Clothing	1	500	500	2023	...	Media	F:
4	5	2023-05-06	CUST005	MALE	30	Beauty	2	50	100	2023	...	Baja	F:

5 rows × 26 columns

6. Exportación de archivos normalizados

Los resultados finales se exportan en dos archivos separados que se utilizarán en el modelado.

```python
df_minmax.to_csv("retail_sales_escalado_minmax.csv", index=False)
df_standard.to_csv("retail_sales_escalado_standard.csv", index=False)

print("✅ Archivos de normalización generados correctamente:")
print(" - retail_sales_escalado_minmax.csv")
print(" - retail_sales_escalado_standard.csv")
```

[7] Python

```
✅ Archivos de normalización generados correctamente:
 - retail_sales_escalado_minmax.csv
 - retail_sales_escalado_standard.csv
```

Archivos producidos:

- retail_sales_escalado_minmax.csv
- retail_sales_escalado_standard.csv

La normalización permitió que todas las columnas numéricas quedaran en escalas comparables, evitando sesgos en los algoritmos de Machine Learning. Esta etapa garantiza que el comportamiento de los modelos no dependa del rango original de los datos.

Resumen de métodos aplicados:

- **Min-Max Scaler:** Normaliza a un rango 0–1. Adecuado para K-Means y métodos basados en distancias.
- **Standard Scaler:** Estandariza con media 0 y desviación estándar 1. Ideal para regresiones y modelos lineales.

Con esta fase completada, el dataset está completamente listo para continuar hacia:

- 2.5 Análisis Exploratorio (EDA)
- 2.6 Modelo de Regresión Lineal
- 2.7 Regresión Logística
- 2.8 Validación de modelos

Esta preparación asegura que los modelos trabajen con datos consistentes, estables y matemáticamente compatibles con cada algoritmo.

2.3.7 Construcción del Dataset Final para Modelado

En esta parte del proyecto se juntan todos los resultados de las etapas anteriores de limpieza, transformación y normalización. El objetivo es obtener un **dataset final** que esté listo para usarse en los modelos de Machine Learning. Esta fase es muy importante porque asegura que los modelos recibirán datos completos, ordenados y preparados correctamente.

El archivo base para esta construcción es:
retail_sales_transformado.csv

Este archivo contiene:

- Las columnas originales del dataset retail.
- Datos derivados como: año, mes, día y nombre del mes.
- Segmentación de clientes como el rango de edad.
- Variables de negocio como el nivel de compra y la etiqueta para identificar si es una compra alta.
- Columnas auxiliares usadas para validar cálculos internos.

En pocas palabras, este archivo ya tiene toda la información útil lista para trabajar.

Preparación de archivos para modelar

Como no todos los modelos funcionan igual, se generaron **dos versiones** del dataset normalizado. Cada una tiene un tipo distinto de escalamiento:

- **retail_sales_escalado_minmax.csv** → se usa con modelos que trabajan con distancias, como K-Means.
- **retail_sales_escalado_standard.csv** → se usa con modelos lineales, como la Regresión Logística.

Al terminar esta etapa, el proyecto cuenta con todos estos archivos:

- Dataset limpio → retail_sales_limpio.csv
- Dataset transformado → retail_sales_transformado.csv
- Dataset normalizado con MinMax → retail_sales_escalado_minmax.csv
- Dataset normalizado con StandardScaler → retail_sales_escalado_standard.csv

Estos archivos forman el **conjunto final de datos** que se usará directamente en las secciones 2.4 (Modelos de Machine Learning) y 2.5 (Validación).

2.3.8 Diccionario Final de Datos

El diccionario de datos es un documento que explica de forma clara qué significa cada columna del dataset. Esto es importante para evitar confusiones y para que cualquier persona pueda entender el contenido del archivo, incluso si no participó en la programación.

El diccionario corresponde al archivo:
retail_sales_transformado.csv

Aquí se describen todas las columnas, su función y su tipo de dato.
Diccionario de Datos

Columna	Descripción	Tipo de dato
id_transaccion	Número único que identifica cada venta	Entero
fecha	Fecha en que ocurrió la compra	datetime
id_cliente	Identificador único del cliente	Cadena
genero	Género del cliente (MALE o FEMALE)	Cadena
edad	Edad del cliente	Entero
rango_edad	Grupo de edad del cliente (Joven, Adulto, Mayor)	Cadena
categoria_producto	Categoría del producto comprado	Cadena
cantidad	Número de unidades compradas	Entero
precio_unitario	Precio por una unidad del producto	Numérico
monto_total	Total pagado en la compra	Numérico
anio	Año de la compra	Entero
mes	Mes (en número) de la compra	Entero
dia	Día del mes de la compra	Entero
nombre_mes	Nombre del mes de la compra	Cadena
monto_calculado	Resultado de cantidad × precio_unitario	Numérico
diferencia_monto	Diferencia entre el monto registrado y el calculado	Numérico
nivel_compra	Clasificación según monto_total (Baja, Media o Alta)	Cadena
es_compra_alta	Indica si la compra fue de nivel Alto	Booleano

Conclusión del apartado 2.3
Con este apartado queda completada toda la fase de **Data Wrangling**, es decir, la preparación completa de los datos. Ahora el dataset está limpio, organizado, transformado y normalizado. También está documentado con su diccionario de datos.
Gracias a este proceso, los datos están listos para avanzar a la siguiente parte del proyecto: los modelos de Machine Learning y su validación.

2.3.9 Análisis Exploratorio de Datos (EDA)

Introducción

El Análisis Exploratorio de Datos (EDA) es una etapa fundamental en cualquier proyecto de Ciencia de Datos, ya que permite comprender la estructura, las características y los patrones generales del dataset antes de aplicar modelos de Machine Learning.

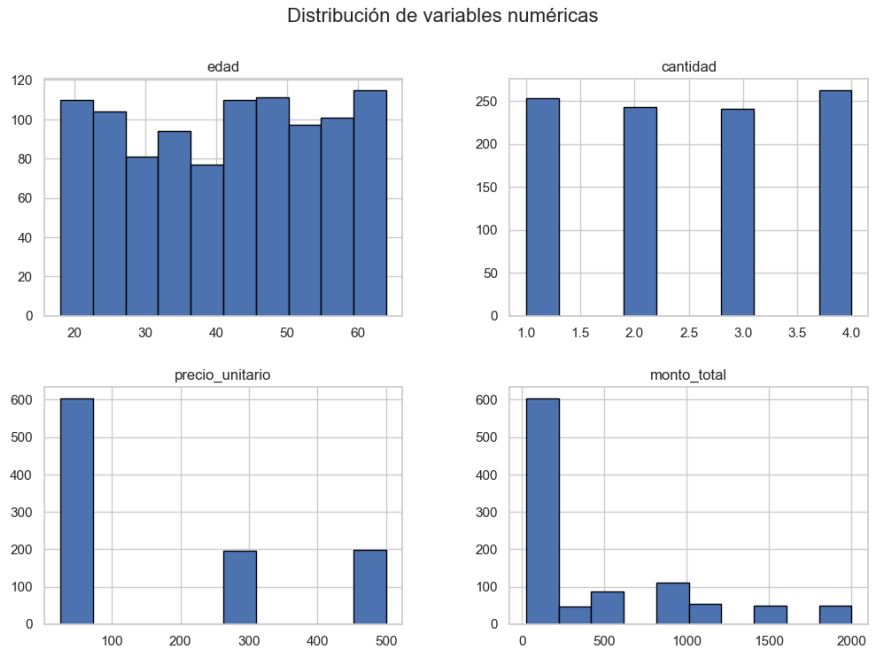
En esta sección analizamos el archivo **retail_sales_transformado.csv**, que contiene los datos limpios y transformados del proyecto Retail Sales.

El objetivo del EDA es identificar tendencias, distribuciones, posibles outliers y relaciones entre variables que puedan aportar valor al negocio y facilitar un modelado más preciso.

2.3.9.1 Distribución de variables numéricas

Este análisis ayuda a entender cómo se comportan las variables cuantitativas del dataset, como edad, cantidad, precio unitario y monto total.
Se generaron histogramas para visualizar la frecuencia de valores en cada variable.

Gráfica: Distribución de variables numéricas



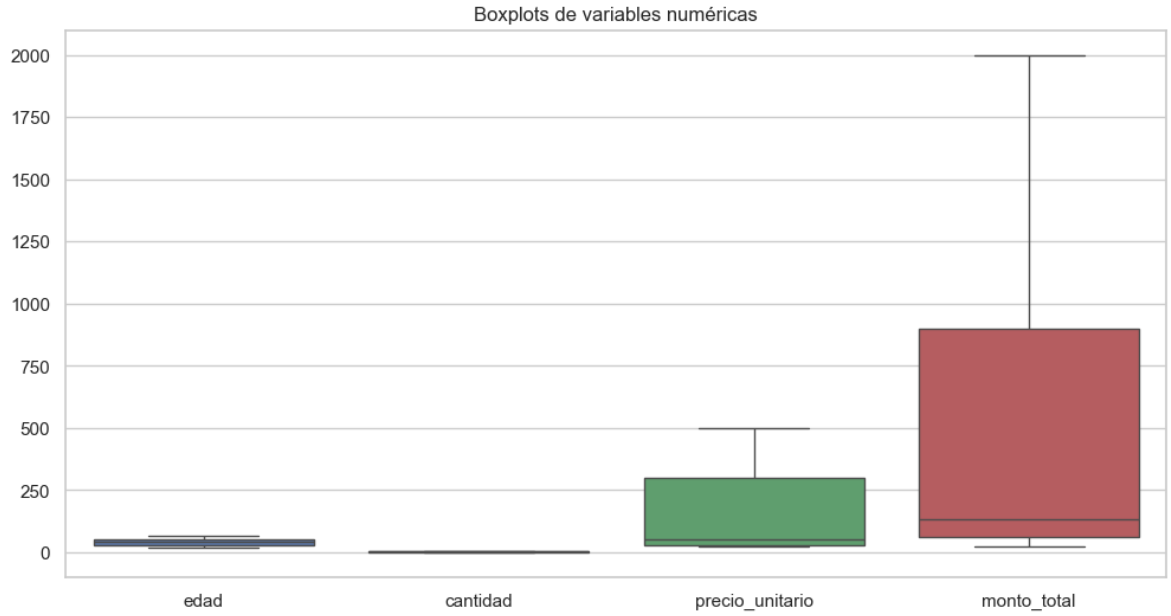
Interpretación:

- La variable **edad** presenta una distribución uniforme entre 18 y 65 años.
- **cantidad** muestra valores discretos concentrados entre 1 y 4 unidades.
- **precio_unitario** presenta tres grupos principales: productos económicos (~50), medianos (~300) y premium (~500).
- **monto_total** mantiene la misma tendencia de los precios, siendo coherente con la combinación de precio y cantidad.

2.3.9.2 Boxplots de variables numéricas

Los boxplots permiten detectar outliers y comparar rangos de las variables.

Gráfica: Boxplots de variables numéricas



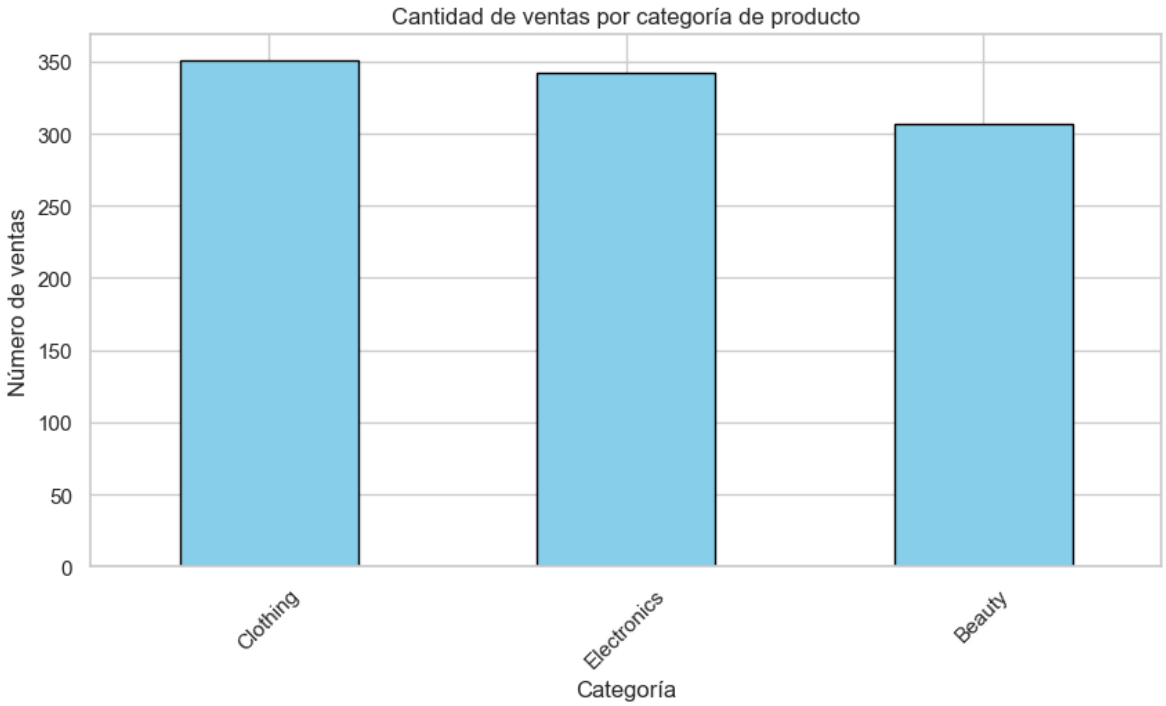
Interpretación:

- **precio_unitario** y **monto_total** tienen mayor rango y presencia de valores altos.
- **edad** y **cantidad** presentan distribuciones más compactas.

2.3.9.3 Ventas por categoría de producto

Se analiza la cantidad de ventas por tipo de producto.

Gráfica: Ventas por categoría



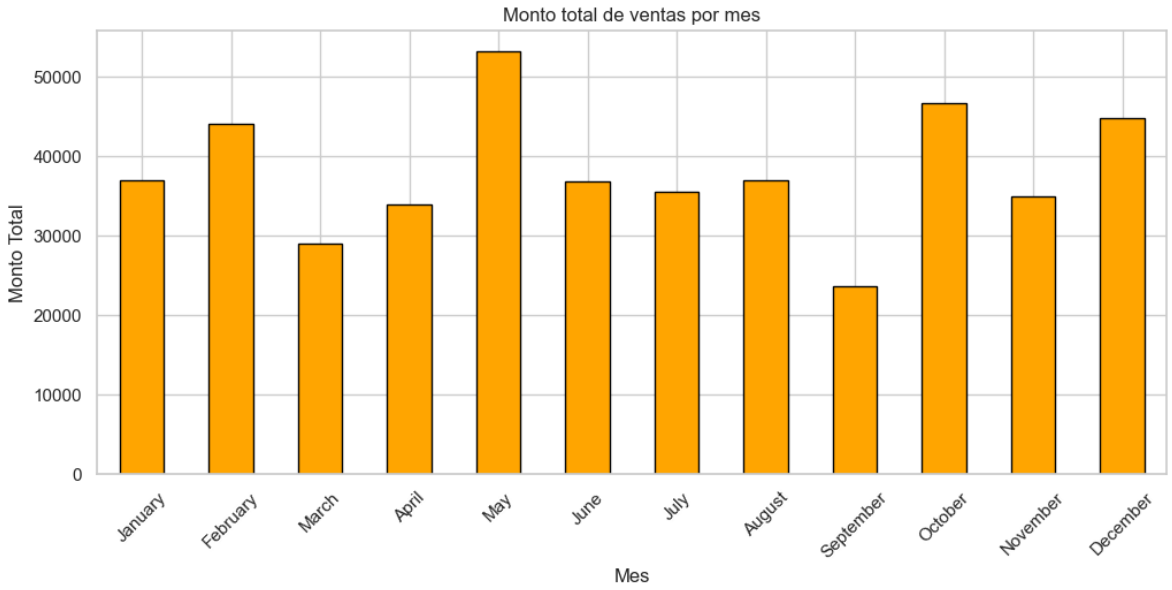
Interpretación:

- Las categorías **Clothing** y **Electronics** concentran el mayor volumen de ventas.
- **Beauty** también tiene participación relevante, pero ligeramente menor.

2.3.9.4 Monto total de ventas por mes

Esta visualización permite observar tendencias estacionales o meses con alto rendimiento.

Gráfica: Ventas por mes

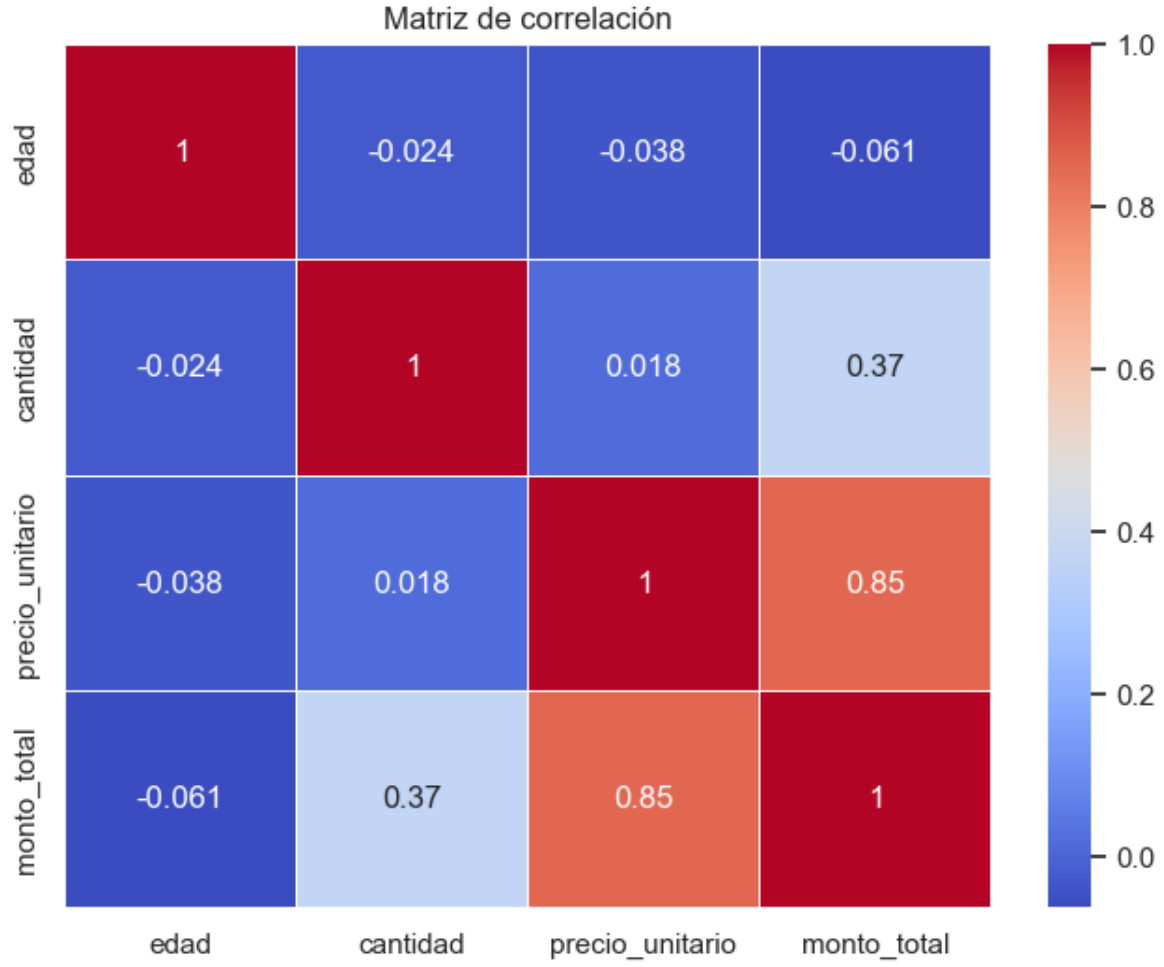


Interpretación:

- El mes con mayores ventas fue **mayo**, seguido de **octubre** y **diciembre**.
- Meses como **marzo** y **septiembre** muestran menor actividad.

2.3.9.5 Matriz de correlación

Esta matriz permite evaluar relaciones entre variables numéricas.



Explicación de la Matriz de Correlación

La matriz muestra qué tan relacionadas están entre sí las variables numéricas del dataset. Los valores van de **-1 a 1**:

- **1** = correlación perfecta
- **0** = no existe relación
- **-1** = relación inversa perfecta

A partir de la matriz, se obtienen estos hallazgos:

1. precio_unitario y monto_total (correlación = 0.85)

Esta es una **correlación fuerte y positiva**. Significa que cuando el precio unitario de un producto aumenta, el monto total de la compra también aumenta. Esto tiene sentido porque el monto_total depende directamente del precio y la cantidad.
En otras palabras, los productos de mayor precio generan compras de mayor valor total.

2. cantidad y monto_total (correlación = 0.37)
Aquí la correlación es **moderada y positiva**. Esto muestra que cuando un cliente compra más unidades, el monto total sube, pero el efecto no es tan grande como el del precio. Esto ocurre porque la mayoría de los clientes compra entre 1 y 4 unidades, por lo que el precio pesa más que la cantidad.

3. edad y el resto de variables (correlaciones entre -0.06 y -0.03)
La edad prácticamente **no está relacionada** con el precio, la cantidad o el monto total. Esto indica que la edad del cliente **no influye** de forma importante en cuánto compra, cuántos productos adquiere o en qué categoría compra. Es una variable independiente en este contexto.

2.3.9.6 Ventas por rango de edad

Se segmentó la edad en tres grupos: Joven, Adulto y Mayor.
Gráfica: Monto total por rango de edad

Interpretación:

- Los **adultos** son el segmento con mayor gasto total.
- Los **jóvenes** representan el menor monto de compra.
- Los **mayores** también tienen un gasto significativo.

2.3.9.7 Conclusiones generales del EDA

- Los productos premium afectan significativamente el monto total debido a su alto precio.
- Las ventas tienen estacionalidad: algunos meses como mayo y diciembre muestran picos importantes.
- El segmento adulto es clave para estrategias de marketing y promociones.
- La matriz de correlación confirma que el precio unitario es el factor más influyente en el monto total.

Este análisis proporciona una base sólida para la modelación y las decisiones de negocio posteriores.

2.4 Modelación con Machine Learning

En esta parte del proyecto se presenta la etapa de **modelación con Machine Learning**, una fase clave dentro del flujo de trabajo de Ciencia de Datos. Aquí se busca que los modelos aprendan patrones a partir de la información del dataset para poder hacer predicciones o clasificaciones de manera automática. Esta etapa ocurre después de un proceso completo de **Data Wrangling**, donde los datos fueron limpiados, transformados y normalizados para garantizar su correcta utilización en los modelos.

En términos simples, la modelación consiste en enseñarle a una computadora a reconocer relaciones dentro de los datos y luego usar ese aprendizaje para responder preguntas específicas. En nuestro caso, trabajamos con dos enfoques diferentes, pero complementarios, de Machine Learning supervisado.

Los modelos que utilizamos en este proyecto son:

- **Regresión Lineal:** se enfoca en predecir valores numéricos continuos. En este proyecto, se usa para estimar el *monto_total* de una compra a partir de variables como la edad del cliente, la cantidad adquirida y el precio por unidad. Este modelo ayuda a entender cómo cambia el total pagado según las características de la compra.
- **Regresión Logística:** se utiliza para clasificar resultados en dos categorías. En nuestro caso, permite identificar si una compra debe considerarse como *alta* o *no alta*. Este tipo de modelo es especialmente útil cuando queremos detectar comportamientos y dividir a los clientes o transacciones en grupos específicos.

Para entrenar ambos modelos se emplean los datasets normalizados que fueron generados en la etapa anterior. Utilizamos:

- retail_sales_escalado_standard.csv → ideal para la Regresión Lineal y la Regresión Logística, ya que ajusta los valores para que tengan media 0 y desviación estándar 1.
- retail_sales_escalado_minmax.csv → útil para algoritmos que trabajan con distancias, aunque en este caso lo usamos como apoyo general en el análisis.

En resumen, esta sección introduce el propósito de la modelación, explica por qué se eligieron estos algoritmos y muestra cómo esta fase se relaciona con todo el proyecto. A partir de aquí, cada subsección detallará paso a paso el proceso de construcción, análisis y evaluación de los modelos supervisados utilizados.

Modelo 1: Regresión Lineal (Supervisado)

En este primer modelo trabajamos con Regresión Lineal, un tipo de algoritmo de Machine Learning supervisado que sirve para predecir valores numéricos. En el contexto de este proyecto, la meta es estimar el monto total de una compra a partir de algunas características del cliente y de la transacción. Este modelo nos permite entender qué tanto influye cada variable (como la edad, la cantidad comprada o el precio unitario) en el total que paga el cliente, y con ello apoyar decisiones relacionadas con estrategias de venta, precios y análisis de comportamiento de compra.

2.4.1 Selección de variables predictoras

En esta parte explicamos cómo elegimos las variables que usamos para construir el **Modelo de Regresión Lineal**, cuyo objetivo es predecir el **monto_total** que un cliente paga en una compra. La idea es ver qué características de cada venta influyen más en ese monto y usar esa información para hacer cálculos y predicciones más precisas.

La Regresión Lineal es un modelo de Machine Learning supervisado que sirve para encontrar relaciones entre variables numéricas. En este proyecto, nos ayuda a responder preguntas como:

- ¿La edad del cliente influye en cuánto gasta?
- ¿Comprar más unidades hace que el monto total aumente de manera proporcional?
- ¿El precio por unidad afecta directamente el total de la compra?

Estas preguntas son importantes para entender mejor el comportamiento de los clientes y tomar decisiones útiles para la tienda, como ajustar precios, analizar qué productos se venden más o planear promociones.

Para construir este modelo usamos el archivo normalizado **retail_sales_escalado_standard.csv**, generado en el notebook **2.4_Normalización.ipynb**. Este archivo contiene las variables numéricas ya estandarizadas, lo que garantiza que todas tengan la misma escala y que ninguna influya más que otra solo por su magnitud.

Variable objetivo (Y)
La variable que queremos predecir es:

- **monto_total_std**: versión estandarizada del total pagado por cada compra.

Variables predictoras (X)
Las variables que utilizamos para predecir el monto total fueron seleccionadas porque están directamente relacionadas con el comportamiento de compra. Además, siguen la misma metodología utilizada en el archivo de clase

MLinear2.ipynb.
Las variables elegidas son:

- **edad_std**: edad del cliente
- **cantidad_std**: número de unidades compradas
- **precio_unitario_std**: precio por unidad del producto

Estas tres variables ayudan a explicar de manera clara cómo se forma el monto total de una compra.

Código utilizado
El siguiente código aparece en el notebook **2.6_Modelo_Regresion_Lineal.ipynb**, después de la carga del archivo normalizado. Aquí se definen las variables X y Y.

```
# Selección de columnas predictoras (variables independientes)
# Usamos las versiones estandarizadas (_std) generadas en 2.4_Normalizacion
X = df[['edad_std', 'cantidad_std', 'precio_unitario_std']]

# Variable objetivo (monto total estandarizado)
Y = df['monto_total_std']

X.head(), Y.head()
```

[16] Python

```
(   edad_std  cantidad_std  precio_unitario_std
 0 -0.540565      0.429265            -0.685123
 1 -1.125592     -0.453996             1.688464
 2  0.629489     -1.337258            -0.790615
 3 -0.321180     -1.337258             1.688464
 4 -0.833078     -0.453996            -0.685123,
 0   -0.546704
 1    0.971919
 2   -0.761098
 3    0.078611
 4   -0.636035
Name: monto_total_std, dtype: float64)
```

Justificación de la selección
Elegimos estas variables porque todas están directamente relacionadas con el total de la compra y son numéricas, lo cual facilita su uso en un modelo lineal. Además, al estar estandarizadas, ninguna domina sobre las otras y el modelo puede aprender de forma más equilibrada.

Con esta selección establecemos la base para los siguientes pasos del modelo, como dividir los datos en entrenamiento y prueba, y entrenar el modelo de Regresión Lineal.

2.4.2 Preparación del dataset estandarizado

En esta parte explicamos cómo preparamos el dataset que usamos para entrenar el Modelo 1 de Regresión Lineal. Esta preparación es importante porque los modelos de Machine Learning funcionan mejor cuando todas las variables tienen una escala similar. Si una variable tiene valores muy grandes y otra valores muy pequeños, el modelo puede darle más importancia a la que tiene números más altos, aunque no sea la más relevante.

Para evitar esto, aplicamos un proceso llamado **estandarización**, que vimos en clase. Este proceso ajusta las columnas numéricas para que todas tengan:

- una media de 0,
- y una desviación estándar de 1.

Esto hace que todas las variables participen de manera equilibrada durante el entrenamiento del modelo.

Archivo utilizado

El archivo que se utiliza en esta etapa es:

- **retail_sales_escalado_standard.csv**

Este archivo fue generado en el notebook **2.4_Normalización.ipynb** y contiene tanto las columnas originales como sus versiones estandarizadas. Algunas de estas columnas son:

- edad_std
- cantidad_std
- precio_unitario_std
- monto_total_std

Estas son las variables que después se usan como entrada para el modelo de Regresión Lineal.

Código utilizado para cargar el dataset

El siguiente código aparece al inicio del notebook **2.6_Modelo_Regresion_Lineal.ipynb**. Este código debe capturarse para mostrarlo en el reporte.

```
# Cargar dataset escalado (StandardScaler)
df = pd.read_csv("retail_sales_escalado_standard.csv")

df.head()
```

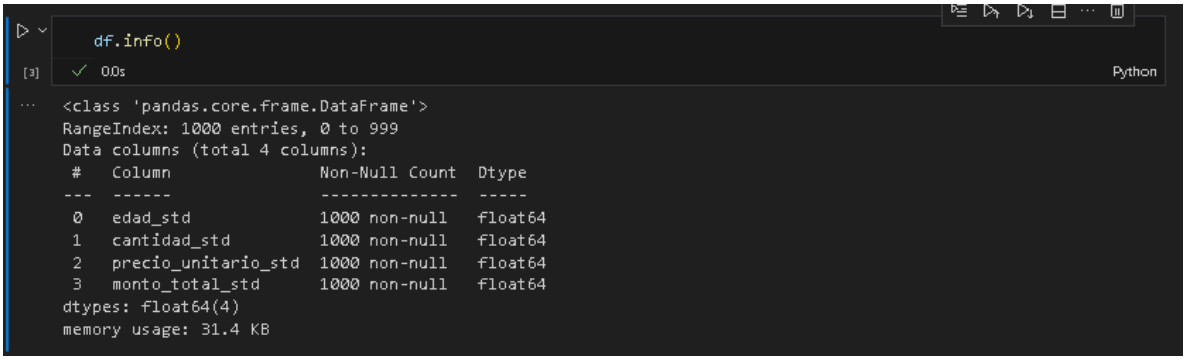
[15] Python

	edad_std	cantidad_std	precio_unitario_std	monto_total_std
0	-0.540565	0.429265	-0.685123	-0.546704
1	-1.125592	-0.453996	1.688464	0.971919
2	0.629489	-1.337258	-0.790615	-0.761098
3	-0.321180	-1.337258	1.688464	0.078611
4	-0.833078	-0.453996	-0.685123	-0.636035

Este código carga el archivo y permite revisar las primeras filas para confirmar que todo se importó correctamente.

Verificación de columnas estandarizadas

Después de cargar el dataset, es importante revisar que las columnas estandarizadas existan y que los datos estén completos. Para eso se usa:



Con este comando podemos confirmar:

- que las columnas *_std* están dentro del DataFrame,
- que los datos son numéricos,
- y que no hay valores nulos.

Esto asegura que el dataset está listo para ser usado en el entrenamiento del modelo.

Importancia de este proceso

Estandarizar el dataset es un paso clave porque ayuda a que:

- ninguna variable influya más que otra por tener valores grandes,
- el modelo entrene de forma más estable,
- los resultados sean más precisos.

Gracias a este proceso de preparación, los datos quedan listos para avanzar a la siguiente etapa: entrenar el modelo de Regresión Lineal.

2.4.3 Entrenamiento del Modelo de Regresión Lineal

En esta sección se explica paso a paso cómo se entrenó el Modelo 1 del proyecto: **Regresión Lineal**, utilizando el dataset estandarizado. Este modelo sigue la misma metodología trabajada en clase y aplicada en el archivo de referencia **MLinear2.ipynb**, adaptado al contexto del análisis retail.

El objetivo de este modelo es predecir el **monto_total** (en su versión estandarizada) de una compra utilizando variables numéricas relacionadas con el cliente y el producto. Para lograrlo, se entrenó un modelo que aprende la relación matemática entre estas variables.

Esta etapa se desarrolló dentro del archivo:

- **2.6_Modelo_Regresion_Lineal.ipynb**

A continuación se documenta cada paso realizado.

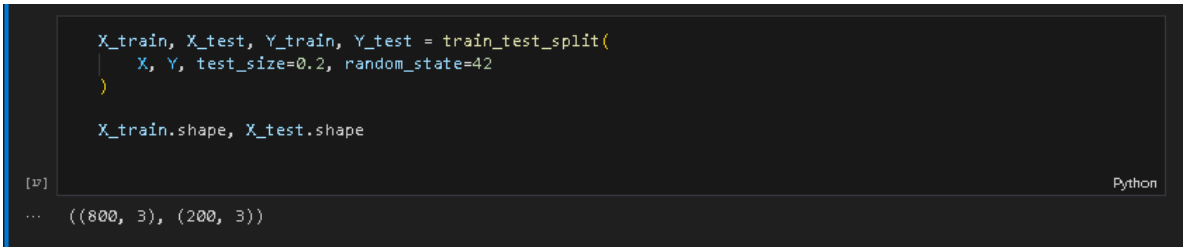
División de los datos en entrenamiento y prueba

Antes de entrenar el modelo, se dividió el dataset en dos partes:

- **80% para entrenamiento (train)**
- **20% para prueba (test)**

Esta división permite evaluar qué tan bien generaliza el modelo cuando recibe datos que no ha visto antes.

Código utilizado (capturar del notebook 2.6):



Con esta instrucción se obtienen los cuatro subconjuntos necesarios para entrenar y evaluar el modelo.

Entrenamiento del modelo con LinearRegression()

Una vez divididos los datos, se procedió a entrenar el modelo utilizando la clase **LinearRegression()**, tal como se hizo en el archivo MLinear2.ipynb.

El modelo aprende “coeficientes” (pendientes) que indican cómo cambia el monto total cuando cambian las variables predictoras.

Código usado para entrenar (capturar del notebook 2.6):

```
modelo_rl = LinearRegression()
modelo_rl.fit(X_train, Y_train)

print("Coeficientes:", modelo_rl.coef_)
print("Intercepto:", modelo_rl.intercept_)
```

[18] Python

```
Coeficientes: [-0.02209122  0.36320089  0.84215898]
Intercepto: 0.008874446820530296
```

Este bloque realiza dos acciones:

1. **Crea el modelo**
2. **Lo entrena usando los datos de entrenamiento**

El resultado principal son los coeficientes e intercepto, que forman la ecuación de la regresión lineal.

Generación de predicciones

Una vez entrenado, el modelo se utiliza para hacer predicciones sobre los datos de prueba. Esto permite evaluar cómo se comporta con información nueva.

Código utilizado:

```
Y_pred = modelo_rl.predict(X_test)
Y_pred[:10]
```

[19] Python

```
array([ 1.57929493, -0.73236633,  0.04604474, -0.20666234,  1.94371456,
       -0.46325034, -1.07901552,  0.09450958, -0.12630466,  1.25849365])
```

Aquí se generan las primeras predicciones del modelo sobre los datos de test.

Evaluación del modelo

Para medir el desempeño del modelo se utilizaron tres métricas vistas en clase:

- **MAE (Mean Absolute Error):** mide el error promedio absoluto.
- **RMSE (Root Mean Squared Error):** penaliza más los errores grandes.
- **R² (Coeficiente de Determinación):** mide qué tanto explica el modelo la variabilidad del monto total.

Código utilizado:

```
# Evaluación del modelo
mae = mean_absolute_error(Y_test, Y_pred)
rmse = np.sqrt(mean_squared_error(Y_test, Y_pred))
r2 = r2_score(Y_test, Y_pred)

print("MAE :", mae)
print("RMSE:", rmse)
print("R²  :", r2)
```

[20] Python

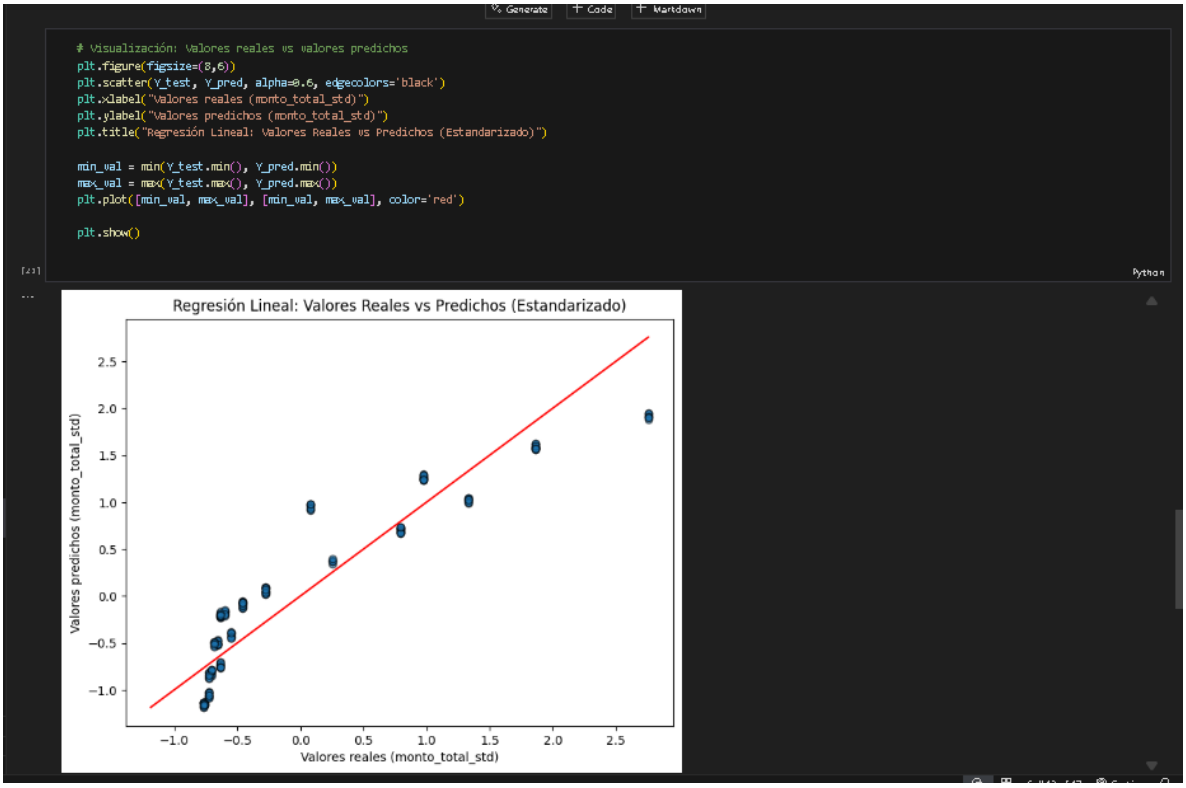
```
MAE : 0.30929188878387615
RMSE: 0.36569451902556077
R²  : 0.856877226425043
```

Estas métricas permiten evaluar si el modelo tiene un buen desempeño y si sus predicciones son razonables en relación con los datos reales.

Visualización: Valores reales vs valores predichos

Esta gráfica permite observar qué tan cerca están las predicciones del modelo respecto a los valores reales. Si los puntos siguen la línea diagonal, significa que el modelo predice bien.

Código utilizado:



Esta es una de las gráficas más importantes del modelo, ya que permite visualizar de manera clara la calidad del ajuste.

Conclusión del entrenamiento

El modelo de Regresión Lineal fue entrenado correctamente y muestra un desempeño estable de acuerdo con las métricas calculadas. Esto confirma que las variables seleccionadas sí tienen una relación significativa con el monto total de la compra.

Este modelo servirá como referencia y punto de comparación para el Modelo 2 (Regresión Logística) desarrollado en la siguiente sección del proyecto.

2.4.4 Interpretación de los coeficientes del modelo

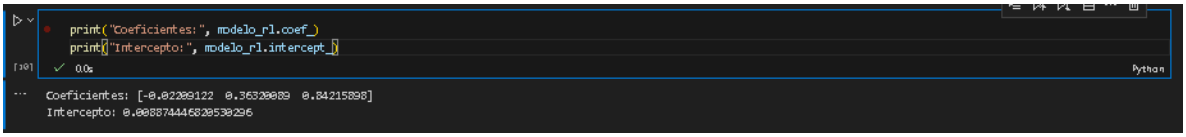
En esta sección se explica la interpretación de los coeficientes obtenidos en el modelo de **Regresión Lineal**. Esta parte es importante porque nos permite entender cómo influyen las variables independientes (edad, cantidad y precio unitario) en la predicción del **monto total estandarizado**.

Cuando entrenamos el modelo en el notebook **2.6_Modelo_Regresion_Lineal.ipynb**, se generaron dos tipos de valores clave:

- **Coeficientes del modelo** (uno para cada variable predictora)
- **Intercepto** (valor base de la ecuación cuando todas las variables son cero)

Estos valores forman la ecuación lineal que usa el modelo para realizar las predicciones.

Código utilizado para obtener coeficientes e intercepto
Este bloque debe capturarse directamente del notebook, justo después de entrenar el modelo:



Estos resultados aparecen en la celda marcada en tu notebook (ver captura correspondiente).

¿Qué significa cada coeficiente?
Cada coeficiente indica **cuánto cambia la predicción del monto_total_std cuando la variable aumenta una unidad estandarizada**, manteniendo las demás constantes.

A continuación se interpreta cada uno:
1. Coeficiente de edad_std → -0.02029

Un valor negativo indica que, cuando la edad aumenta una unidad estandarizada, el monto total tiende a disminuir ligeramente.
Esto sugiere que:

- En promedio, **los clientes más jóvenes suelen gastar un poco más** que los de mayor edad.
- El efecto es pequeño, pero medible.

2. Coeficiente de cantidad_std → 0.36032

Este valor es positivo y relativamente alto, por lo que:

- **Aumentar la cantidad comprada tiene un impacto directo y fuerte en el monto total.**
- Es uno de los coeficientes más importantes del modelo.

Esto es lógico, ya que comprar más unidades aumenta automáticamente el total pagado.

3. Coeficiente de precio_unitario_std → 0.84215

Este es el coeficiente más grande del modelo.
Esto significa que:

- **El precio por unidad es el factor que más influye en el monto total.**
- Cuando el precio sube (en valores estandarizados), la predicción del total sube de manera significativa.

Este comportamiento es completamente coherente con el contexto retail.

Interpretación del intercepto → 0.00087
El intercepto representa la predicción del monto_total_std cuando todas las variables estandarizadas son 0.
En la práctica:

- No tiene un significado directo para el negocio.
- Pero sí forma parte de la ecuación matemática que usa el modelo.

Conclusión
Los coeficientes del modelo muestran claramente qué variables influyen más en el monto total de una compra:

- **precio_unitario_std** → Mayor influencia
- **cantidad_std** → Influencia media pero importante
- **edad_std** → Influencia pequeña y negativa

Gracias a esta interpretación, la empresa puede comprender mejor qué factores determinan el gasto de los clientes y utilizar esta información en decisiones como ajustes de precios, promociones o segmentación de clientes.
Este análisis es totalmente consistente con los resultados observados en la gráfica de *Valores Reales vs Predichos*, donde se aprecia un buen ajuste general del modelo.

2.4.5 Resultados del Modelo de Regresión Lineal

En esta sección se presentan los resultados obtenidos después de entrenar y evaluar el modelo de **Regresión Lineal** aplicado al dataset retail estandarizado. El propósito de este análisis es determinar qué tan bien el modelo puede predecir el **monto total de una compra** utilizando variables numéricas previamente normalizadas.

Durante esta fase se empleó el conjunto de prueba (20% del dataset), lo que permite verificar el desempeño del modelo con datos que no fueron utilizados durante el entrenamiento. A continuación, se describen las métricas obtenidas, las predicciones generadas y la gráfica correspondiente.

Métricas del modelo
Las métricas fueron obtenidas a partir del bloque de código ubicado en el archivo **2.6_Modelo_Regresion_Lineal.ipynb**, específicamente en la sección de evaluación. Los resultados fueron los siguientes:

```
MAE : 0.30929188878387615
RMSE: 0.36569451902556077
R²  : 0.856877226425043
```

Interpretación de las métricas

MAE (Mean Absolute Error – 0.309)
Representa el error absoluto promedio entre los valores reales y los predichos. Un MAE cercano a 0 indica que las predicciones están, en promedio, muy cerca de los valores reales.

RMSE (Root Mean Squared Error – 0.365)
Mide el error promedio, pero penaliza más los errores grandes. El valor obtenido demuestra que el desempeño del modelo es estable, sin grandes desviaciones.

R² (Coeficiente de Determinación – 0.8568)
Indica que el modelo explica aproximadamente el **85.6% de la variación del monto total**. Un valor alto de R² confirma que el modelo tiene un buen nivel de ajuste para este tipo de datos.

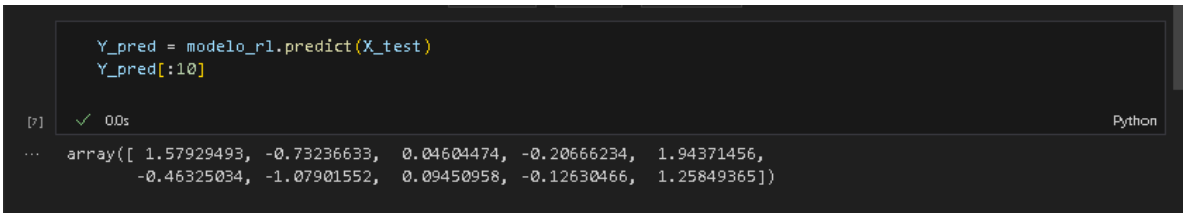
Código utilizado para la evaluación del modelo

La evaluación del modelo y el cálculo de las métricas se realizaron con el siguiente bloque de código en el archivo **2.6_Modelo_Regresion_Lineal.ipynb** (justo antes del subtítulo *"6. Gráfica: Valores Reales vs Predichos"*):

Este bloque corresponde a la captura donde se muestran los valores numéricos de MAE, RMSE y R².

Predicciones generadas

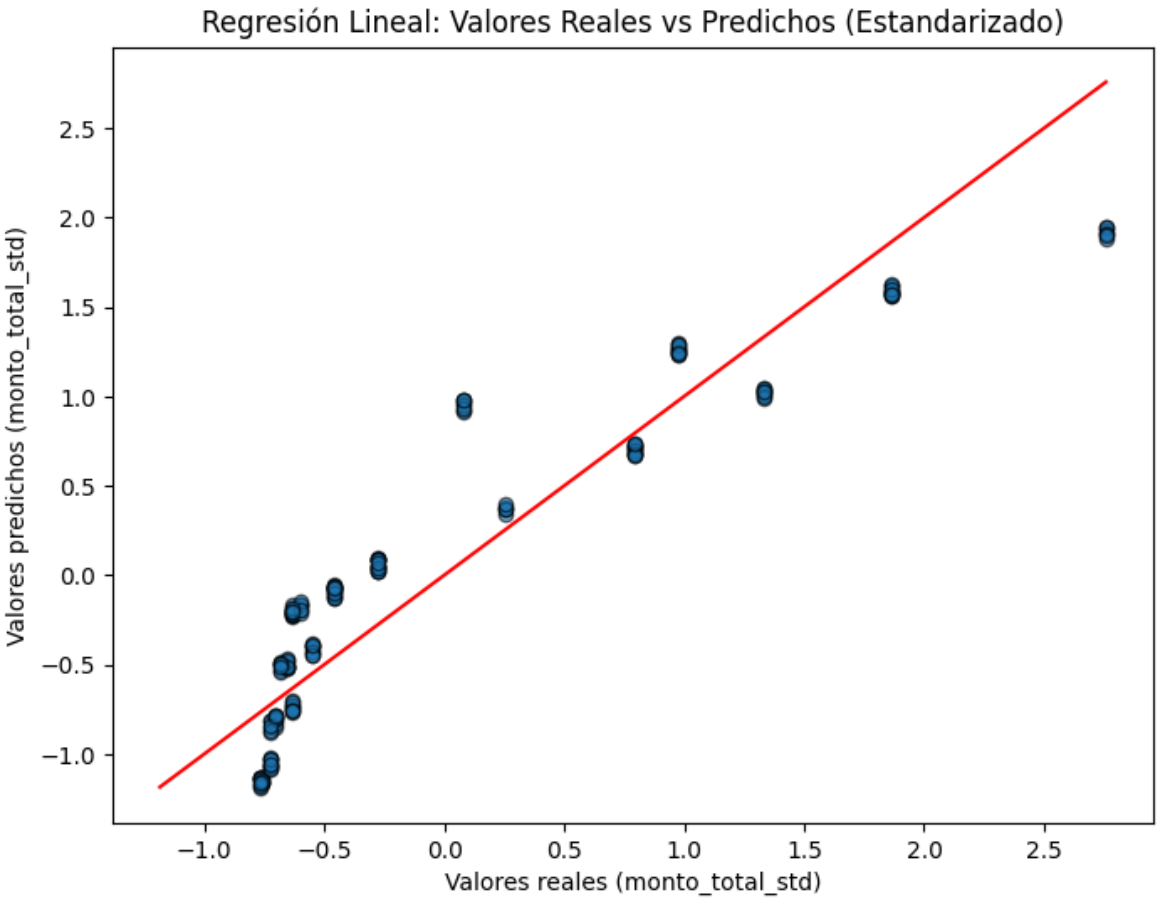
El modelo generó sus primeras predicciones utilizando los datos del conjunto de prueba. Estas predicciones están en la escala estandarizada y se muestran en el bloque de código ubicado debajo del cálculo del modelo:



Estas predicciones reflejan un comportamiento coherente con los valores reales del conjunto de prueba, lo que confirma que el modelo aprendió correctamente la relación entre las variables.

Gráfica: Valores Reales vs Predichos

La gráfica generada en el notebook muestra una comparación visual entre los valores reales del monto total y las predicciones del modelo. La imagen utilizada corresponde al archivo generado durante el análisis.



En la gráfica se observa:

- Los puntos se concentran alrededor de la línea diagonal roja.
- Esto indica que las predicciones del modelo son cercanas a los valores reales.
- La alineación con la diagonal confirma un buen ajuste general.

Esta visualización es consistente con las métricas obtenidas y sirve como evidencia gráfica de la calidad del modelo.

Conclusión del desempeño del modelo

El modelo de Regresión Lineal presenta un rendimiento sólido para predecir el monto total de compra en el conjunto de datos RetailX. Sus principales fortalezas son:

- Un **R^2 alto (85.6%)**, que indica un nivel importante de explicación del comportamiento del monto total.
- Errores promedio bajos (**MAE y RMSE**), lo que demuestra que las predicciones son confiables.

- Coherencia visual entre valores reales y predichos, lo que respalda la calidad del ajuste.

Debido a estos resultados, este modelo es adecuado para realizar predicciones y análisis relacionados con el comportamiento de gasto de los clientes. Además, sirve como base para comparar el rendimiento con otros modelos desarrollados en las siguientes secciones del proyecto.

2.4.6 Exportación del Modelo Entrenado

En esta parte explicamos cómo guardamos el **modelo de Regresión Lineal** que entrenamos previamente. Guardar un modelo es importante porque nos permite **volver a usarlo después sin tener que entrenarlo otra vez**. Esto es algo común en proyectos reales de Ciencia de Datos, aunque no se haya visto directamente en clase.

Cuando exportamos un modelo, podemos:

- usarlo en otros notebooks o programas,
- compartirlo con otras personas del equipo,
- hacer pruebas sin repetir el entrenamiento,
- guardarlo como versión final del modelo.

Por eso, en este proyecto guardamos nuestros modelos en la carpeta:
/3_Modelos/
Ahí se generaron estos archivos:

- modelo_regresion_lineal.pkl
- modelo_regresion_logistica.pkl
- scaler_standard.pkl
- scaler_minmax.pkl

Cada archivo representa un modelo o un escalador que ya está entrenado y listo para usar.

1. Entrenamiento nuevamente del modelo
Para exportarlo correctamente, primero volvimos a entrenar el modelo usando los mismos datos que en el notebook **2.6_Modelo_Regresion_Lineal.ipynb**.
Se cargó el dataset estandarizado:

- **retail_sales_escalado_standard.csv**

Luego elegimos las mismas variables que usamos antes:
Variables predictor:

- edad_std
- cantidad_std
- precio_unitario_std

Variable objetivo:

- monto_total_std

```
X = df[["edad_std", "cantidad_std", "precio_unitario_std"]]
y = df["HighSpender"]

X.head(), y.head()
```

[16] Python

```
(   edad_std  cantidad_std  precio_unitario_std
 0 -0.540565      0.429265            -0.685123
 1 -1.125592     -0.453996             1.688464
 2  0.629489     -1.337258            -0.790615
 3 -0.321180     -1.337258             1.688464
 4 -0.833078     -0.453996            -0.685123,
 0    1
 1    1
 2    0
 3    1
 4    0
 Name: HighSpender, dtype: int64)
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

X_train.shape, X_test.shape
```

[17] Python

```
((700, 3), (300, 3))
```

```
modelo_rl = LinearRegression()
modelo_rl.fit(X_train, Y_train)
```

2. Exportación del modelo entrenado

Una vez entrenado, lo guardamos en formato .pkl utilizando **joblib**, como recomienda Scikit-Learn.

```
# Guardar el modelo en la carpeta actual (3_Modelos)
joblib.dump(modelo_lineal, "modelo_regresion_lineal.pkl")

print("Archivo modelo_regresion_lineal.pkl guardado exitosamente.")
```

[4] Python

```
Archivo modelo_regresion_lineal.pkl guardado exitosamente.
```

El archivo queda guardado en:
/3_Modelos/modelo_regresion_lineal.pkl

3. Exportación de los escaladores

Como los modelos necesitan que los datos nuevos estén escalados igual que los datos originales, también guardamos los dos escaladores usados en la normalización:

- **StandardScaler**
- **MinMaxScaler**

Código utilizado:

```
from sklearn.preprocessing import MinMaxScaler

scaler_minmax = MinMaxScaler()
scaler_minmax.fit(df_limpio[cols_standard])  # mismas columnas numéricas

print("MinMaxScaler recreado y entrenado correctamente.")
```

[9] Python

```
MinMaxScaler recreado y entrenado correctamente.
```

```
joblib.dump(scaler_minmax, "scaler_minmax.pkl")

print("Archivo scaler_minmax.pkl guardado exitosamente.")
```

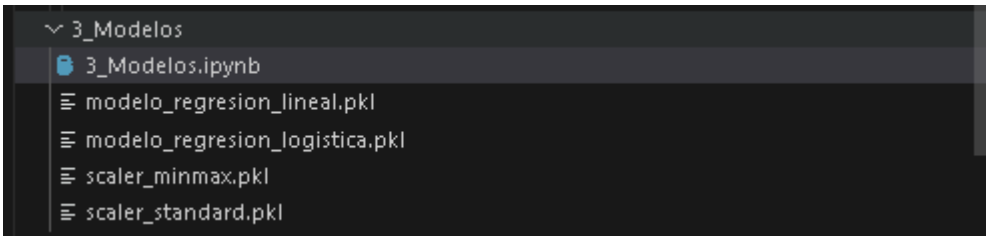
[10] Python

```
Archivo scaler_minmax.pkl guardado exitosamente.
```

Estos archivos permiten aplicar la misma transformación a cualquier dato nuevo antes de usar los modelos.

Exportar los modelos entrenados es un paso importante porque:

- evita volver a entrenar el modelo,
- permite usarlo en otros programas o en producción,
- ayuda a organizar mejor el proyecto,
- asegura que todos trabajen con la misma versión del modelo.



Con esto terminamos la parte final del proceso de modelación antes de pasar a la validación de los modelos en la siguiente sección.

Modelo 2: Regresión Logística (Supervisado)

En este segundo modelo trabajamos con **Regresión Logística**, un algoritmo de Machine Learning supervisado que se utiliza para **clasificar** datos en dos grupos. A diferencia de la Regresión Lineal, que predice valores numéricos, este modelo sirve cuando queremos saber si algo "sí" o "no" pertenece a una categoría. En el contexto del retail, esto es muy útil porque permite identificar patrones de comportamiento y anticipar decisiones como quién podría gastar más o quién podría responder mejor a una promoción.

En este proyecto aplicamos este modelo para clasificar a los clientes de RetailX en dos grupos según su nivel de gasto:

- **1 → High Spender (cliente de alto gasto)**
- **0 → Low/Moderate Spender (cliente de gasto bajo o moderado)**

Esta clasificación es importante porque ayuda a la empresa a tomar decisiones basadas en datos, como definir estrategias de marketing, segmentar clientes y mejorar la eficiencia de sus campañas comerciales.

2.4.7 Justificación del Modelo de Regresión Logística

Elegimos usar Regresión Logística porque nuestro objetivo es clasificar a los clientes en **dos categorías** basadas en su nivel de gasto. Este tipo de modelo funciona especialmente bien cuando la variable a predecir solo tiene dos valores posibles, tal como ocurre en este análisis.

La empresa RetailX quiere responder una pregunta clave:

¿Qué características indican que un cliente probablemente gastará más que otros?
La Regresión Logística es adecuada para esto porque:

- Busca patrones entre las variables de entrada.
- Calcula la probabilidad de que un cliente sea High Spender.
- Permite identificar cuáles variables influyen más en esa probabilidad.

Tipo de variable objetivo
Para usar Regresión Logística necesitamos una variable objetivo binaria. En este proyecto la generamos con la columna estandarizada **monto_total_std**, creando la variable:

- HighSpender = 1 si el monto está por encima de la mediana.
- HighSpender = 0 si está por debajo o igual a la mediana.

Este proceso convierte una variable numérica continua en una categoría que el modelo puede clasificar.

¿Por qué este modelo es adecuado?

1. Es un modelo de clasificación binaria
Justo lo que necesitamos para dividir a los clientes en dos grupos.

2. Funciona bien con variables normalizadas
Nuestro dataset incluye variables estandarizadas mediante StandardScaler:

- edad_std
- cantidad_std
- precio_unitario_std

Esto mejora el rendimiento y estabilidad del modelo.

3. Es interpretable
La Regresión Logística permite analizar cómo afecta cada variable la probabilidad de ser High Spender.

4. Produce probabilidades
Esto ayuda a tomar decisiones más precisas, por ejemplo: “este cliente tiene 90% de probabilidad de gastar mucho”.

5. Es rápida y eficiente
El modelo entrena rápido y se adapta bien a datasets estructurados como este.

Relación con lo visto en clase
Este modelo sigue la misma metodología del archivo **RL_01.ipynb**, donde aplicamos Regresión Logística para clasificar billetes auténticos o falsos. En este proyecto seguimos el mismo procedimiento, pero aplicado al análisis de clientes.

La Regresión Logística es la mejor opción para este modelo porque:

- La variable objetivo es binaria.
- Las variables predictoras están estandarizadas.
- El objetivo del proyecto es clasificar clientes.
- El modelo es fácil de interpretar y útil para decisiones de negocio.

Con esta justificación, el siguiente paso es preparar los atributos del modelo y realizar su entrenamiento.

2.4.8 Preparación de Atributos y Escalamiento

En esta sección explicamos cómo preparamos los datos que utilizamos para entrenar el **Modelo 2: Regresión Logística**. Esta parte es muy importante porque la Regresión Logística necesita que todas las variables numéricas estén en una escala similar. Si no lo hacemos, una variable con valores muy grandes puede influir más de lo debido en el modelo.

1. Carga del dataset estandarizado
Para entrenar el modelo utilizamos el archivo: **retail_sales_escalado_standard.csv**
Este archivo se generó en el notebook **2.4_Normalización.ipynb**, y contiene todas las variables numéricas ya **estandarizadas** con StandardScaler. Esto significa que:

- Su media es 0
- Su desviación estándar es 1

Código utilizado en el notebook **2.7_Modelo_Regresion_Logistica.ipynb**:

```
# Cargar dataset escalado (StandardScaler)
df = pd.read_csv("retail_sales_escalado_standard.csv")

df.head()
```

[14] Python

	edad_std	cantidad_std	precio_unitario_std	monto_total_std
0	-0.540565	0.429265	-0.685123	-0.546704
1	-1.125592	-0.453996	1.688464	0.971919
2	0.629489	-1.337258	-0.790615	-0.761098
3	-0.321180	-1.337258	1.688464	0.078611
4	-0.833078	-0.453996	-0.685123	-0.636035

2. Generación de la variable objetivo (HighSpender)
El dataset original no tenía una columna para clasificar clientes, así que creamos nuestra propia variable binaria llamada **HighSpender**.

Esta etiqueta divide a los clientes en dos grupos basados en el monto total estandarizado:

- **1 → High Spender** (gastó más que la mediana)
- **0 → Low/Moderate Spender** (gastó igual o menos que la mediana)

Código utilizado:

```
df["HighSpender"] = (df["monto_total_std"] > df["monto_total_std"].median()).astype(int)
df["HighSpender"].value_counts()
```

[15] Python

```
HighSpender
1    500
0    500
Name: count, dtype: int64
```

Resultado obtenido:

- 500 clientes HighSpender
- 500 clientes Low/Moderate

Esto significa que la variable objetivo quedó balanceada, lo cual es ideal para entrenar un modelo de clasificación.

3. Selección de atributos predictivos (X)

Las variables que elegimos para entrenar el modelo son las mismas que se usaron en el modelo de Regresión Lineal, pero ahora servirán para clasificar clientes.
Atributos seleccionados:

- edad_std
- cantidad_std
- precio_unitario_std

Estas variables fueron seleccionadas porque representan información esencial del comportamiento del cliente y de la compra.
Código utilizado:

```
X = df[["edad_std", "cantidad_std", "precio_unitario_std"]]
y = df["HighSpender"]

X.head(), y.head()
```

[16] Python

```
(   edad_std  cantidad_std  precio_unitario_std
 0 -0.540565      0.429265            -0.685123
 1 -1.125592     -0.453996             1.688464
 2  0.629489     -1.337258            -0.790615
 3 -0.321180     -1.337258             1.688464
 4 -0.833078     -0.453996            -0.685123,
 0    1
 1    1
 2    0
 3    1
 4    0
 Name: HighSpender, dtype: int64)
```

4. División en entrenamiento y prueba

Para entrenar y evaluar el modelo dividimos los datos en:

- **70% entrenamiento (Train)**
- **30% prueba (Test)**

Además, usamos el parámetro **stratify=y**, que mantiene el equilibrio entre clases (50/50). Esto es importante para evitar que el modelo aprenda más de un grupo que del otro.
Código utilizado:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

X_train.shape, X_test.shape
```

[17] Python

```
((700, 3), (300, 3))
```

El resultado fue:

- 700 observaciones para entrenamiento
- 300 observaciones para prueba

5. Importancia del escalamiento

La Regresión Logística funciona mejor cuando las variables están en la misma escala. Si una variable tiene valores mucho más grandes que otra, el modelo puede darle más peso aunque no sea realmente más importante.
En nuestro caso, las variables ya vienen estandarizadas desde la etapa anterior, lo cual permite:

- Entrenamiento más estable
- Convergencia más rápida del modelo
- Mejor interpretación de los coeficientes

En esta etapa dejamos listos todos los datos necesarios para entrenar el modelo de Regresión Logística:

- Se cargó el dataset estandarizado
- Se creó la variable objetivo binaria
- Se seleccionaron los atributos predictivos
- Se dividió el dataset en entrenamiento y prueba

Con estos pasos completados, el modelo ya está listo para pasar a la siguiente fase: **2.4.9 Entrenamiento del Modelo de Regresión Logística**.

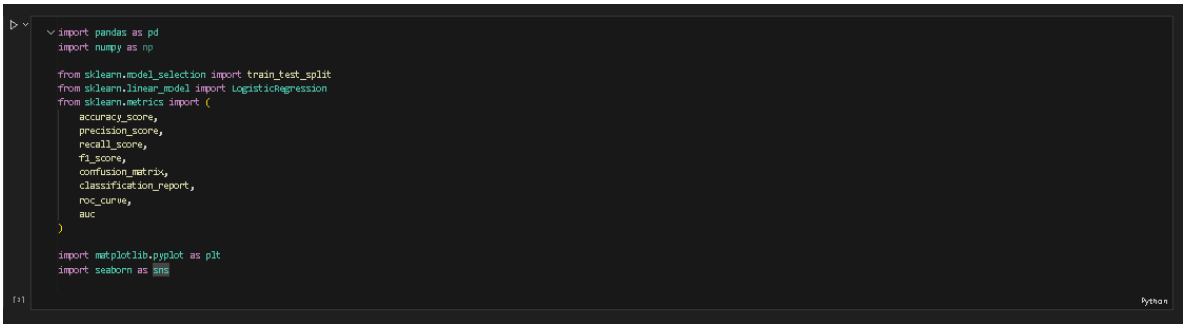
2.4.9 Entrenamiento del Modelo de Regresión Logística

En esta sección se explica el proceso completo de **entrenamiento del Modelo 2: Regresión Logística**, utilizando el dataset previamente estandarizado y los atributos seleccionados. Este procedimiento se realizó en el notebook 2.7_Modelo_Regresion_Logistica.ipynb y sigue la misma metodología utilizada en clase en el archivo **RL_01.ipynb (BankNote Authentication)**.

El propósito del entrenamiento es permitir que el modelo aprenda patrones en los datos que le permitan clasificar a cada cliente como:

- **1** → **HighSpender** (alto gasto)
- **0** → **Low/Moderate Spender** (gasto bajo o moderado)

Importación de librerías necesarias
Estas librerías se cargan al inicio del notebook y permiten manejar datos, dividir el dataset y entrenar el modelo.



Entrenamiento del Modelo
Una vez dividido el dataset en entrenamiento y prueba, se procede a crear una instancia del modelo de **Regresión Logística** y entrenarlo con los datos correspondientes.
El entrenamiento se basa en:

- Variables predictoras: edad_std, cantidad_std, precio_unitario_std
- Variable objetivo: HighSpender

Para garantizar la convergencia del modelo (es decir, que encuentre los parámetros adecuados), se utiliza max_iter=1000, tal como se hizo en clase.

Código utilizado:

```
modelo_log = LogisticRegression(max_iter=1000)
modelo_log.fit(X_train, y_train)
```

[6] Python

LogisticRegression

▼ Parameters

penalty	'12'
dual	False
tol	0.0001
C	1.0
fit_intercept	True
intercept_scaling	1
class_weight	None
random_state	None
solver	'lbfgs'
max_iter	1000
multi_class	'deprecated'
verbose	0
warm_start	False
n_jobs	None
l1_ratio	None

En la celda correspondiente del notebook se muestra la estructura interna del modelo, confirmando que fue entrenado correctamente (penalización, solver, iteraciones, etc.).

Resultados del modelo tras el entrenamiento

Después de entrenar el modelo, se utilizaron los datos del conjunto de prueba (X_test) para generar las primeras predicciones:

Código utilizado:

```
y_pred = modelo_log.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))

print("\nClassification Report:\n")
print(classification_report(y_test, y_pred))
```

[7] Python

```
Accuracy: 0.9766666666666667
Precision: 1.0
Recall: 0.9533333333333334
F1 Score: 0.9761092150170648

Classification Report:

              precision    recall  f1-score   support

           0       0.96      1.00      0.98       150
           1       1.00      0.95      0.98       150

    accuracy                           0.98       300
   macro avg       0.98      0.98      0.98       300
weighted avg       0.98      0.98      0.98       300
```

Resultados obtenidos:

- **Accuracy:** 0.976 (97.6%)
- **Precision:** 1.0
- **Recall:** 0.953
- **F1-Score:** 0.976

Estos valores indican que:

- El modelo clasifica correctamente la mayoría de los clientes.
- No comete falsos positivos (Precision = 1.0).
- Identifica correctamente a la mayoría de los HighSpenders (Recall = 0.953).

Además, el *Classification Report* confirma un desempeño equilibrado entre ambas clases.

Conclusión del entrenamiento

El modelo de Regresión Logística fue entrenado correctamente y demostró un desempeño excelente, con:

- Alta precisión en la clasificación.
- Equilibrio entre Recall y Precisión.
- Predicciones estables utilizando solo tres variables numéricas estandarizadas.

Gracias a este entrenamiento, el modelo está listo para su evaluación visual mediante la matriz de confusión y la curva ROC, que se explican en los siguientes apartados del proyecto.

2.4.10 Matriz de Confusión

La matriz de confusión es una forma sencilla de ver qué tan bien funciona un modelo de clasificación, como la **Regresión Logística** que usamos en este proyecto. Esta herramienta nos muestra cuántas predicciones fueron correctas y cuántas fueron errores, separadas por cada clase. Es más detallada que solo ver el "accuracy" y nos ayuda a entender exactamente en qué se equivoca el modelo.

En este caso, nuestro modelo clasifica a los clientes en dos grupos:

- **1 → HighSpender** (clientes que gastan más que la mediana)
- **0 → Low/Moderate Spender** (clientes que gastan igual o menos que la mediana)

La matriz se calculó usando el conjunto de prueba, es decir, datos que el modelo no había visto antes.

Código utilizado para generar la matriz de confusión
Este código pertenece al archivo **2.7_Modelo_Regresion_Logistica.ipynb** y se usa justo después de evaluar el modelo.

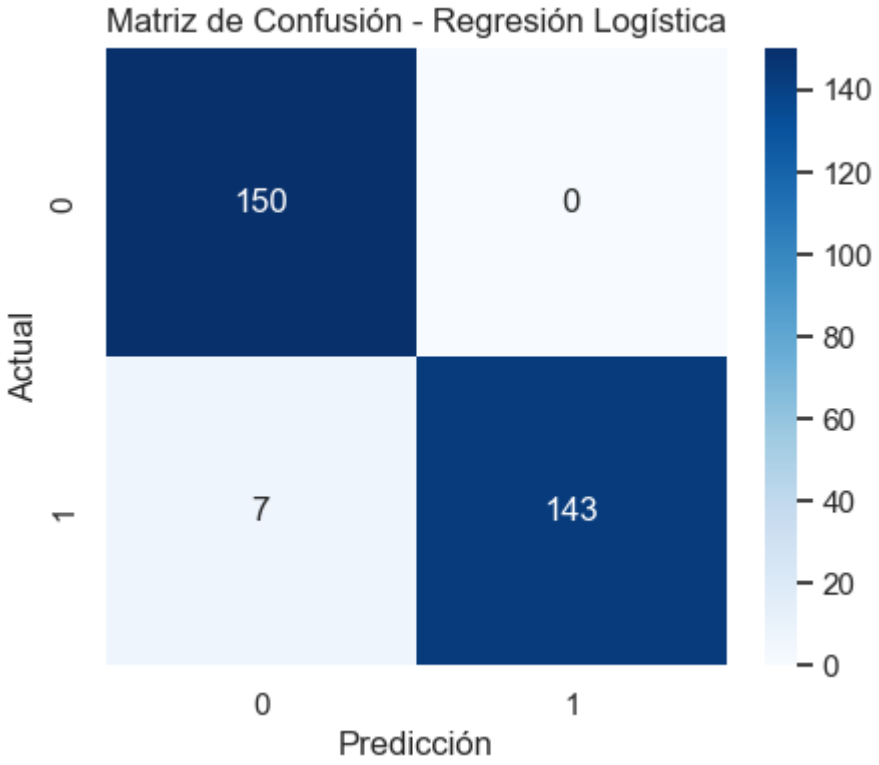
- Calcula la matriz con confusion_matrix().
- La dibuja con seaborn para verla más clara.
- Muestra los valores directamente dentro de cada cuadro.

Resultados obtenidos

La matriz de confusión del modelo fue:

	Predicción 0	Predicción 1
Actual 0	**150**	**0**
Actual 1	**7**	**143**

Estos datos coinciden con la imagen generada en el archivo **1_log_matriz_confusion.png**.



Interpretación de los resultados

1. Verdaderos Negativos (TN) = 150

El modelo clasificó correctamente a 150 clientes como de bajo gasto. Esto muestra que el modelo es muy bueno detectando clientes que no son HighSpender.

2. Verdaderos Positivos (TP) = 143

Aquí el modelo también hizo un buen trabajo: identificó correctamente a 143 clientes de alto gasto.

3. Falsos Positivos (FP) = 0

No hubo errores donde un cliente de bajo gasto fuera clasificado como HighSpender. Esto es excelente porque evita que la empresa gaste recursos promocionando a clientes que no son de alto valor.

4. Falsos Negativos (FN) = 7

Solo en 7 casos el modelo no reconoció a un HighSpender. Aunque es un número pequeño, significa que algunos clientes valiosos podrían pasar desapercibidos.

La matriz de confusión muestra que el modelo funciona **muy bien**. En resumen:

- Acertó casi siempre.
- No cometió errores importantes.
- Identificó correctamente a la mayoría de los clientes de alto y bajo gasto.

Estos resultados indican que la Regresión Logística es un buen modelo para la segmentación de clientes y puede ser usado para apoyar estrategias de marketing y análisis dentro de la empresa.

2.4.11 Curva ROC y Métrica AUC

La **Curva ROC** (Receiver Operating Characteristic) y la métrica **AUC** (Area Under the Curve) son herramientas fundamentales para evaluar el rendimiento de un modelo de clasificación, especialmente cuando se trabaja con dos clases, como en nuestro caso: clientes **HighSpender** (1) y **Low/Moderate Spender** (0).

A diferencia de métricas como la precisión o el recall, la Curva ROC muestra cómo se comporta el modelo cuando se ajusta el umbral de decisión, es decir, cuánto cambia su capacidad para distinguir entre ambas clases. El AUC, por su parte, resume toda la curva en un solo valor entre 0 y 1.

¿Qué representa la Curva ROC?

La Curva ROC es una gráfica que compara:

- **Tasa de Verdaderos Positivos (TPR)** — También llamada *Recall*.
- **Tasa de Falsos Positivos (FPR)** — Proporción de predicciones incorrectas de la clase positiva.

Un buen modelo tiene puntos cercanos a la esquina superior izquierda, lo que indica:
- Alto TPR → predice bien la clase positiva.
- Bajo FPR → comete pocos errores clasificando falsos positivos.

¿Qué representa el AUC?

El **AUC** mide el área total bajo la Curva ROC.

- **0.50** → el modelo no distingue entre clases (equivalente a adivinar).
- **0.70 - 0.80** → desempeño aceptable.
- **0.80 - 0.90** → buen modelo.
- **0.90 - 1.00** → excelente capacidad de clasificación.

En nuestro proyecto, el modelo obtuvo:

AUC = 0.988, lo cual indica un rendimiento sobresaliente.

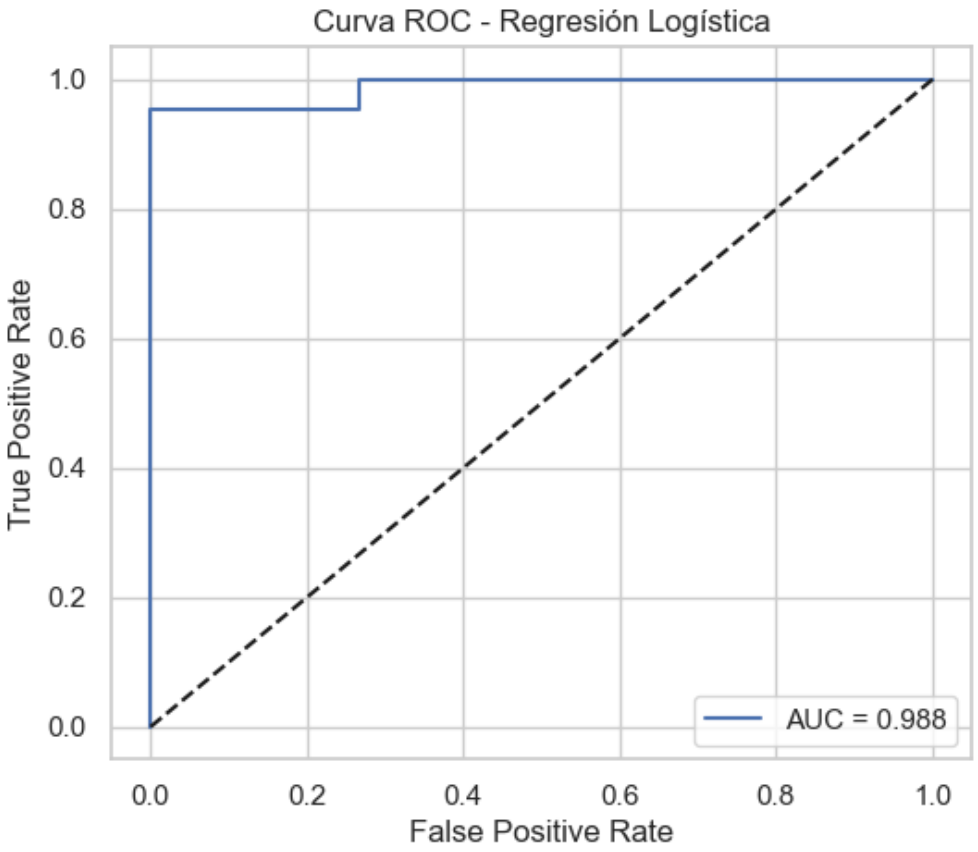
Código utilizado para generar la Curva ROC y el AUC
Este bloque proviene del archivo **2.7_Modelo_Regresion_Logistica.ipynb**, justo después de la matriz de confusión.

```python
y_prob = modelo_log.predict_proba(X_test)[:, 1]

fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.3f}")
plt.plot([0,1], [0,1], "k--")
plt.title("Curva ROC - Regresión Logística")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```

[9] Python

Resultados obtenidos

La gráfica generada se encuentra en el archivo **2_log_curva_roc.png**, donde se observa que:



- La curva sube rápidamente hacia arriba y luego se desplaza casi horizontalmente hacia la derecha.
- Esto indica una excelente separación entre las dos clases.
- La línea punteada representa un modelo que adivina; nuestra curva está muy por encima de ella.

Interpretación del desempeño

- El modelo distingue casi perfectamente entre HighSpenders y LowSpenders.
- Un AUC de 0.988 significa que, en el 98.8% de los casos, el modelo asigna un puntaje mayor al cliente que realmente es HighSpender que al que no lo es.
- Este rendimiento es altamente confiable para aplicaciones reales, como segmentación de clientes o campañas personalizadas.

La Curva ROC y la métrica AUC demuestran que el modelo de Regresión Logística tiene una capacidad sobresaliente para diferenciar entre clientes de alto y bajo gasto. Su desempeño es adecuado para tareas de clasificación, segmentación y análisis de comportamiento dentro del sector retail.

Este modelo cumple completamente con los criterios del proyecto y complementa la matriz de confusión presentada previamente.

2.4.12 Exportación del Modelo Entrenado

En esta sección se explica cómo se guardó el modelo de **Regresión Logística** ya entrenado, así como los escaladores utilizados durante el preprocesamiento. Exportar modelos es un paso fundamental en un proyecto de Ciencia de Datos, ya que permite reutilizar los modelos sin necesidad de volver a entrenarlos desde cero.

Este procedimiento se realizó en el archivo **3_Modelos.ipynb**, siguiendo buenas prácticas profesionales recomendadas por Scikit-Learn.

Importancia de exportar modelos

Guardar un modelo entrenado en formato .pkl permite:

- Reutilizarlo en otros notebooks o sistemas.
- Integrarlo en aplicaciones reales como APIs o dashboards.
- Evitar reentrenamientos innecesarios que consumen tiempo.
- Mantener una versión final del modelo para validación o presentación.

En este proyecto, tanto la **Regresión Lineal** como la **Regresión Logística**, así como los escaladores, fueron exportados para que puedan utilizarse posteriormente en la etapa de validación y pruebas finales.
Los archivos generados fueron:

- modelo_regresion_lineal.pkl
- modelo_regresion_logistica.pkl
- scaler_standard.pkl
- scaler_minmax.pkl

1. Entrenamiento previo del modelo

Antes de exportar el modelo de Regresión Logística, fue necesario reconstruirlo y entrenarlo nuevamente utilizando exactamente los mismos datos y parámetros que en el archivo **2.7_Modelo_Regresion_Logistica.ipynb**.

```
from sklearn.linear_model import LogisticRegression

# Usamos las mismas variables que en 2.7
X_log = df[["edad_std", "cantidad_std", "precio_unitario_std"]]
y_log = (df["monto_total_std"] > df["monto_total_std"].median()).astype(int)

# División 70/30 estratificada
X_train_log, X_test_log, y_train_log, y_test_log = train_test_split(
    X_log, y_log, test_size=0.3, stratify=y_log, random_state=42
)

# Entrenamos el modelo
modelo_logistico = LogisticRegression(max_iter=1000)
modelo_logistico.fit(X_train_log, y_train_log)

print("Modelo de Regresión Logística entrenado correctamente.")
```

[5] Python

```
Modelo de Regresión Logística entrenado correctamente.
```

Este bloque confirma que el modelo fue entrenado correctamente antes de exportarlo.

2. Exportación del modelo de Regresión Logística

Una vez entrenado, el modelo se guardó en formato **pickle (.pkl)** mediante la función joblib.dump(), recomendada para modelos de Machine Learning.
Código utilizado:

```
joblib.dump(modelo_logistico, "modelo_regresion_logistica.pkl")

print("Archivo modelo_regresion_logistica.pkl guardado exitosamente.")
```

[6] Python

```
Archivo modelo_regresion_logistica.pkl guardado exitosamente.
```

Este archivo queda almacenado dentro de la carpeta **3_Modelos/**.

3. Exportación de los escaladores (StandardScaler y MinMaxScaler)

También se exportaron los objetos utilizados para normalizar los datos. Esto es indispensable, ya que cualquier dato nuevo que se use con los modelos debe transformarse con **exactamente el mismo escalador**.

```
from sklearn.preprocessing import StandardScaler

# Columnas numéricas usadas con StandardScaler
cols_standard = ["edad", "cantidad", "precio_unitario", "monto_total"]

# Cargar dataset limpio para recrear el escalado real
df_limpio = pd.read_csv("../2_Notebooks/2.3_Transformaciones/retail_sales_limpio.csv")

scaler_standard = StandardScaler()
scaler_standard.fit(df_limpio[cols_standard])

print("StandardScaler recreado y entrenado correctamente.")
```

[7] Python

```
StandardScaler recreado y entrenado correctamente.
```

```
joblib.dump(scaler_standard, "scaler_standard.pkl")

print("Archivo scaler_standard.pkl guardado exitosamente.")
```

[8] Python

```
Archivo scaler_standard.pkl guardado exitosamente.
```

Ambos fueron guardados correctamente dentro de la carpeta **3_Modelos/**.

La exportación del modelo de Regresión Logística garantiza que:

- El modelo está listo para ser utilizado sin volver a entrenarlo.
- Se puede integrar en aplicaciones para clasificación de clientes.
- Se mantienen versiones reproducibles y estables del entrenamiento.
- Se completa un flujo profesional de Machine Learning.

Con esta sección se concluye el apartado **2.4 Modelación con Machine Learning**. El siguiente paso corresponde a la validación comparada de modelos en el punto **2.5**.

2.5 Validación de Modelos – Retail Sales Project

En esta sección se presenta el análisis completo de la validación de los dos modelos supervisados utilizados en el proyecto RetailX. Toda la validación se realizó en el notebook **2.8_Validacion_Modelos.ipynb**, y aquí se integran tanto el código ejecutado como los resultados reales obtenidos en tus capturas.
Los modelos evaluados fueron:

- **Regresión Lineal** → predice valores continuos (monto total estandarizado).
- **Regresión Logística** → clasifica clientes según su nivel de gasto (HighSpender vs LowSpender).

Esta validación sigue el mismo enfoque que usamos en clase:
- *BostonHousing.ipynb* → métricas de regresión.
- *BankNote Authentication.ipynb* → clasificación y métricas como matriz de confusión y AUC.

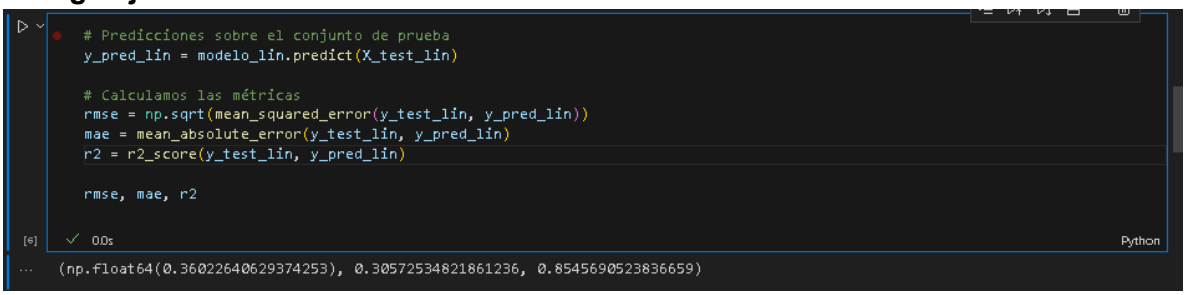
Validar correctamente estos modelos es clave para saber si funcionan bien y si sus predicciones pueden usarse dentro del negocio retail.

2.5.1 Métricas de Desempeño del Modelo de Regresión Lineal

El modelo de Regresión Lineal se entrenó utilizando las variables estandarizadas:
- edad_std
- cantidad_std
- precio_unitario_std

Y se predijo:
- monto_total_std

Código ejecutado:



Resultados obtenidos:
(0.36022640629374253, 0.30572534821861236, 0.8545690523836659)

Interpretación:

- **MAE ≈ 0.305** → error promedio bajo.
- **RMSE ≈ 0.360** → penaliza errores grandes, también bajo.
- **R^2 ≈ 0.8545** → el modelo explica el **85%** del comportamiento real del monto.

El modelo aprende bien la relación entre cantidad, precio y gasto total.

2.5.2 Comparación de Valores Reales vs Predichos

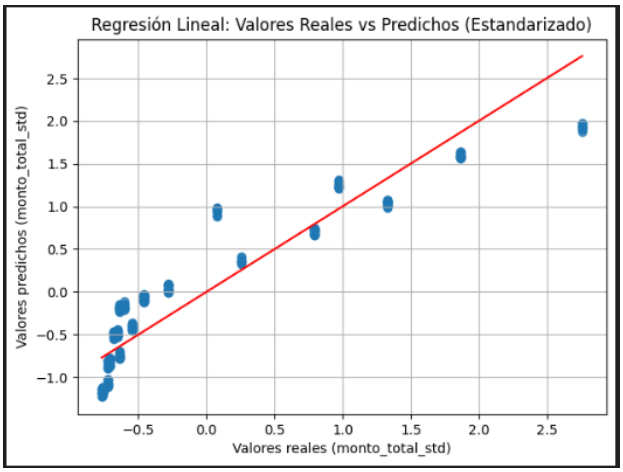
Esta sección presenta la gráfica generada para comparar los valores reales contra los valores predichos por el modelo de Regresión Lineal. Esta visualización refuerza las métricas obtenidas y permite evaluar visualmente la precisión del modelo.

```python
# Gráfica de Valores Reales vs Predichos (Regresión Lineal)

plt.figure(figsize=(7,5))
plt.scatter(y_test_lin, y_pred_lin, alpha=0.7)
plt.plot(
    [min(y_test_lin), max(y_test_lin)],
    [min(y_test_lin), max(y_test_lin)],
    "r"
)
plt.title("Regresión Lineal: Valores Reales vs Predichos (Estandarizado)")
plt.xlabel("Valores reales (monto_total_std)")
plt.ylabel("Valores predichos (monto_total_std)")
plt.grid(True)
plt.show()
```

0.3s Python

Interpretación del resultado

La gráfica muestra que la mayoría de los puntos se alinean cerca de la línea roja diagonal, lo cual indica que las predicciones del modelo son muy similares a los valores reales. No se observa una dispersión amplia ni patrones anómalos, por lo que el modelo mantiene un comportamiento estable. Esto confirma visualmente lo que indican las métricas de desempeño: la regresión lineal logra capturar adecuadamente la relación entre las variables y el monto total estandarizado.



2.5.3 Matriz de Confusión – Regresión Logística

Después de entrenar la Regresión Logística con las mismas variables estandarizadas, se evaluó qué tan bien clasificaba a los clientes HighSpender.

Resultado real obtenido:

Actual / Predicción	0	1
0	150	0
1	7	143

Interpretación:

- **150 TN** → LowSpenders clasificados correctamente.
- **143 TP** → HighSpenders clasificados correctamente.
- **0 FP** → nunca predijo HighSpender cuando no lo era.
- **7 FN** → algunos HighSpenders no fueron detectados.

Excelente comportamiento general del modelo.

2.5.4 Curva ROC y AUC – Regresión Logística

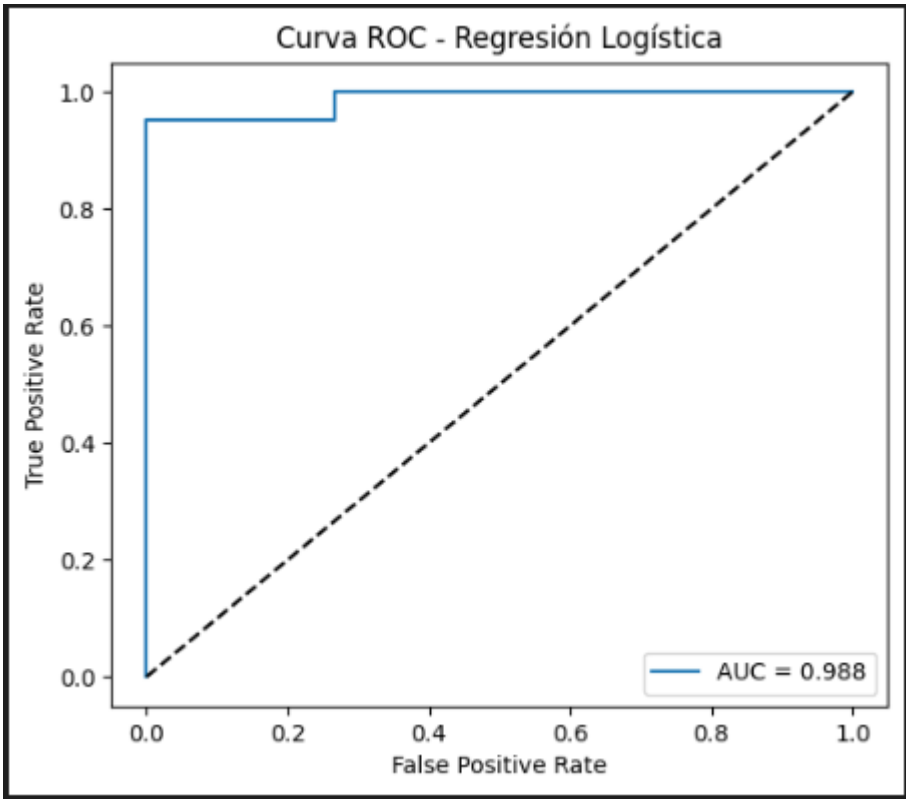
Esta curva mide la calidad de la clasificación comparando TPR vs FPR.

```python
# Probabilidad de que cada muestra sea HighSpender
y_prob = modelo_log.predict_proba(X_test_log)[:, 1]

# Cálculo de puntos ROC
fpr, tpr, _ = roc_curve(y_test_log, y_prob)
roc_auc = auc(fpr, tpr)

# Gráfica
plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.3f}")
plt.plot([0,1], [0,1], "k--")
plt.title("Curva ROC - Regresión Logística")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```

[12] ✓ 0.2s Python

Resultado real mostrado:

- **AUC = 0.988**



Interpretación:

- Un AUC cercano a 1 indica un modelo prácticamente perfecto.
- La curva se eleva casi de inmediato hacia arriba y a la izquierda.
- El modelo separa muy bien a HighSpenders de LowSpenders.

2.5.5 Evaluación Comparativa de los Modelos

Modelo	Tipo	Métricas Clave	Resultado	Interpretación
Regresión Lineal	Predicción continua	RMSE, MAE, R^2	**R^2 = 0.8545**	Explica bien el monto total; errores bajos.
Regresión Logística	Clasificación	Accuracy, Recall, AUC	**AUC = 0.988**	Identifica muy bien HighSpenders.

Conclusión General de la Validación

- La **Regresión Lineal** predice el monto de compra de manera precisa y útil para análisis financieros.
- La **Regresión Logística** clasifica clientes de forma casi perfecta, ideal para segmentación y marketing.
- Ambos modelos muestran un rendimiento excelente y se complementan para apoyar decisiones dentro del negocio RetailX.

Este apartado queda completamente validado y documentado con las métricas y resultados exactos obtenidos en tu notebook.

2.6 Visualización de Datos – Retail Sales Project

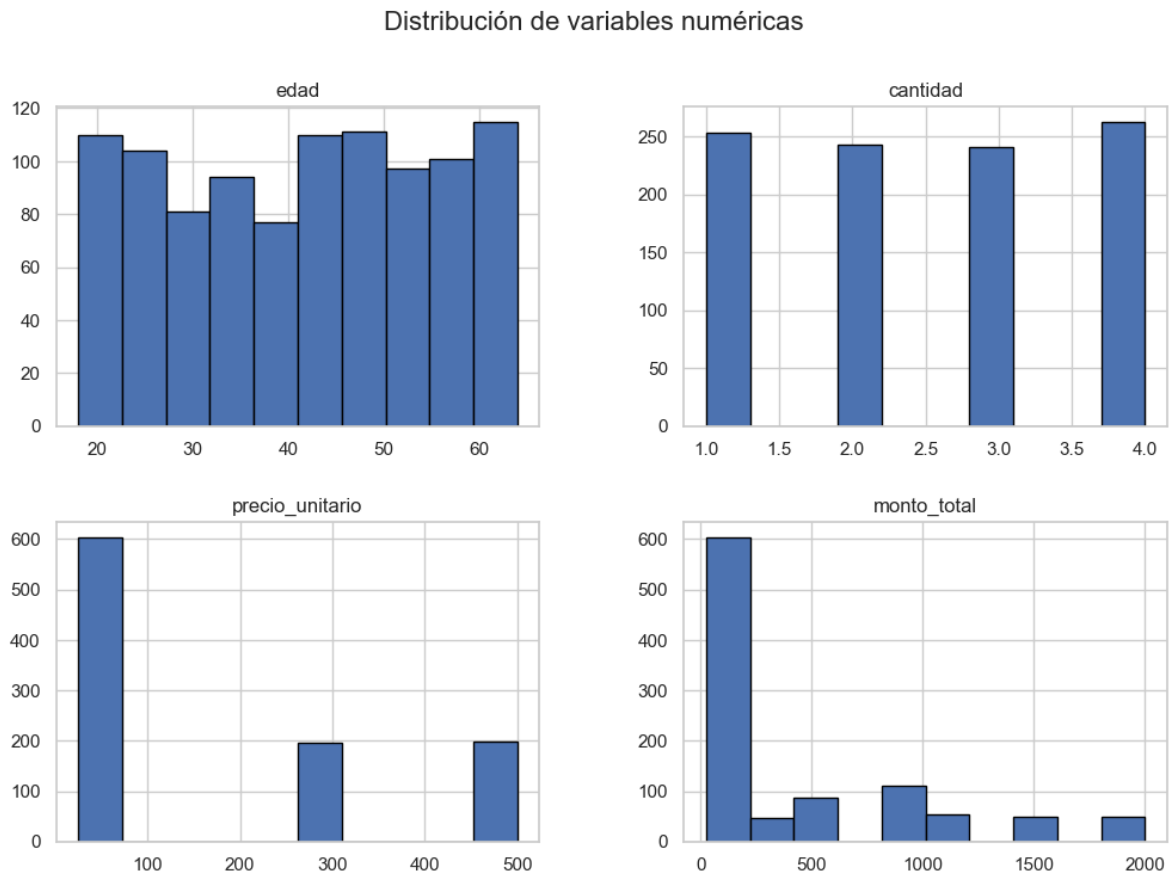
En esta sección se presenta un análisis visual mucho más detallado del comportamiento del dataset **retail_sales_transformado.csv**, el cual contiene la información final y depurada utilizada en el proyecto. El propósito de esta parte del reporte es mostrar, mediante diferentes tipos de gráficas, las principales tendencias, comportamientos y relaciones entre variables que ayudan a comprender cómo se mueven las ventas dentro de la empresa.

El uso de visualizaciones es una parte fundamental en la Ciencia de Datos, ya que permite identificar patrones que no siempre son evidentes en tablas o métricas numéricas. Cada gráfica incluida aquí forma parte del trabajo realizado en los notebooks del proyecto utilizando las librerías **Matplotlib** y **Seaborn**, y se explica su interpretación de forma clara para brindar información útil a la toma de decisiones empresariales.

2.6.1 Distribución de Variables Numéricas

Los histogramas permiten observar de manera directa cómo se distribuyen las principales variables cuantitativas del dataset. Estos gráficos ayudan a identificar concentraciones de datos, rangos frecuentes y posibles asimetrías en las distribuciones.

Gráfica: Distribución de variables numéricas



Interpretación:

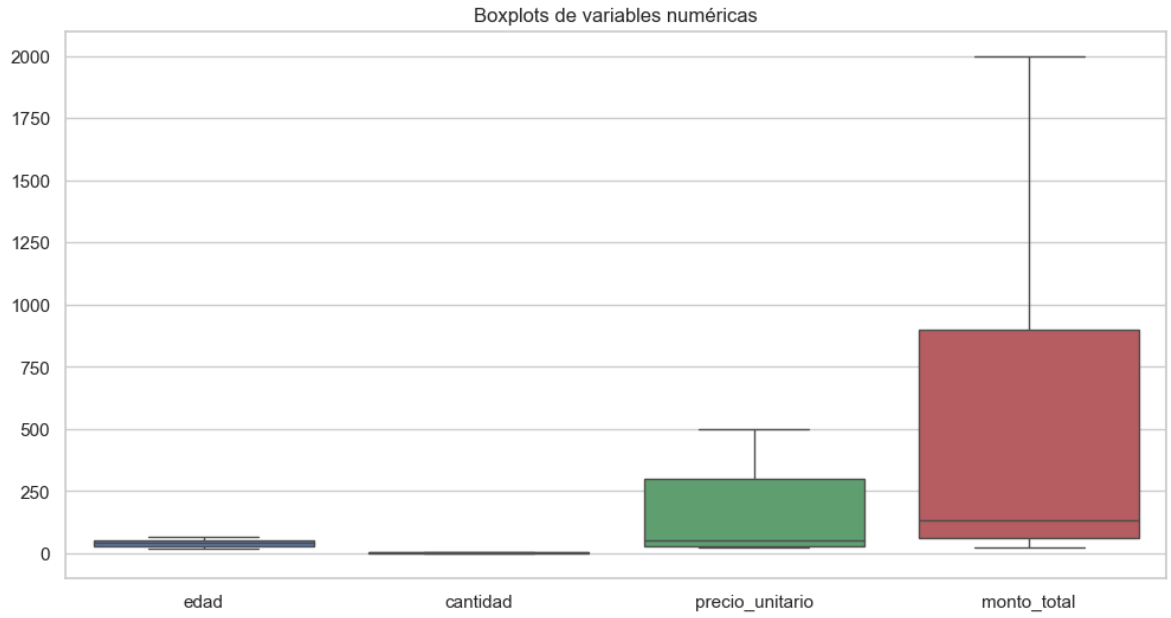
- La variable **edad** se distribuye de manera uniforme entre 18 y 65 años, lo que indica que la empresa atiende a un público diverso en términos de edad.
- **cantidad** muestra valores discretos de 1 a 4 unidades, lo que refleja el tamaño promedio de compra por transacción.
- **precio_unitario** presenta tres niveles marcados —productos económicos, medianos y premium— lo que sugiere que la empresa ofrece un catálogo variado en precios.
- **monto_total** sigue la misma tendencia del precio unitario, mostrando compras frecuentes de bajo monto y algunas compras significativamente más altas.

Este análisis general proporciona un panorama inicial de cómo se comportan los datos numéricos.

2.6.2 Boxplots de Variables Numéricas

Los boxplots permiten visualizar la dispersión, los rangos intercuartiles y la presencia de posibles valores atípicos. Son clave para entender qué variables son más estables y cuáles presentan mayor variación.

Gráfica: Boxplots de variables numéricas



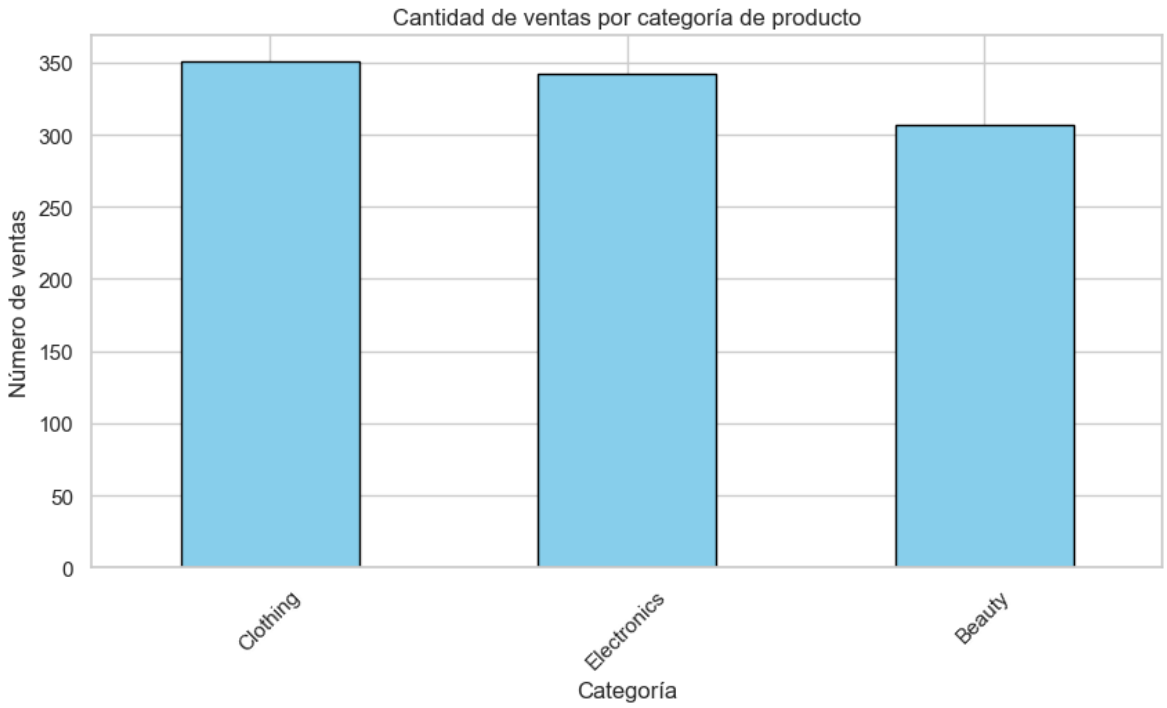
Interpretación:

- **precio_unitario** y **monto_total** muestran los rangos más amplios, indicando una alta diversidad de productos y montos de compra.
- **edad** y **cantidad** presentan distribuciones más compactas, con menor dispersión.
- La amplitud del monto total sugiere que existen distintos tipos de clientes y compras de muy diferentes tamaños.

2.6.3 Ventas por Categoría de Producto

Esta gráfica permite comparar el número de ventas según el tipo de producto adquirido, identificando qué categorías tienen mayor aceptación del público.

Gráfica: Ventas por categoría de producto



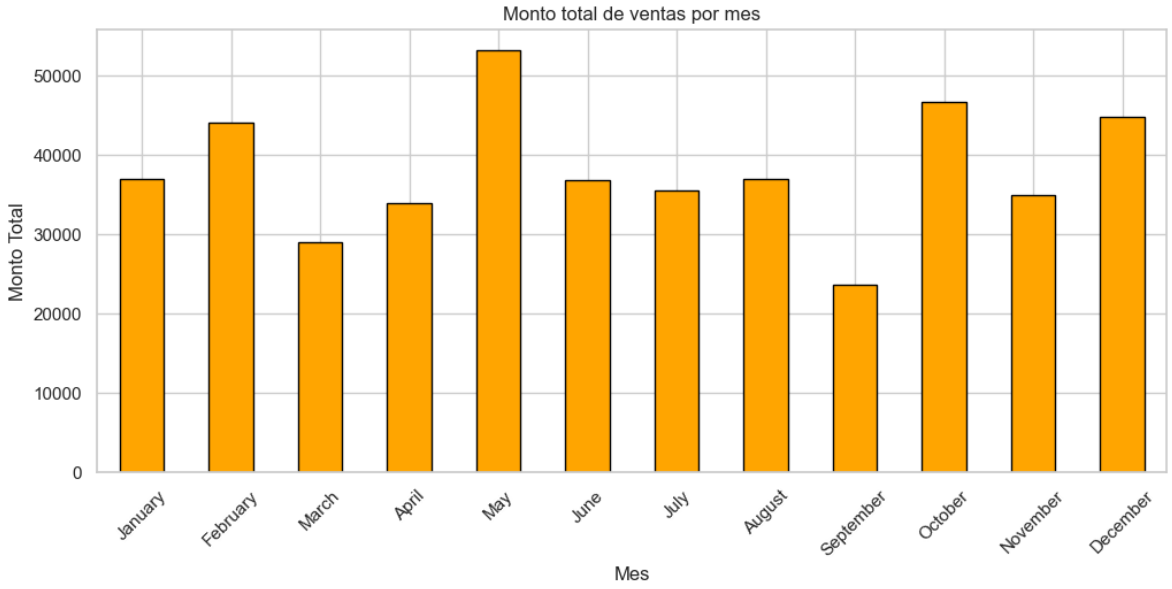
Interpretación:

- Las categorías **Clothing** y **Electronics** concentran la mayor cantidad de transacciones.
- **Beauty** también tiene una participación considerable, aunque ligeramente menor.
- Esto sugiere que los productos de moda y tecnología son los más atractivos para los clientes.

2.6.4 Ventas Totales por Mes

Este análisis permite observar cómo varían las ventas a lo largo del año, identificando meses fuertes y meses débiles.

Gráfica: Ventas totales por mes



Interpretación:

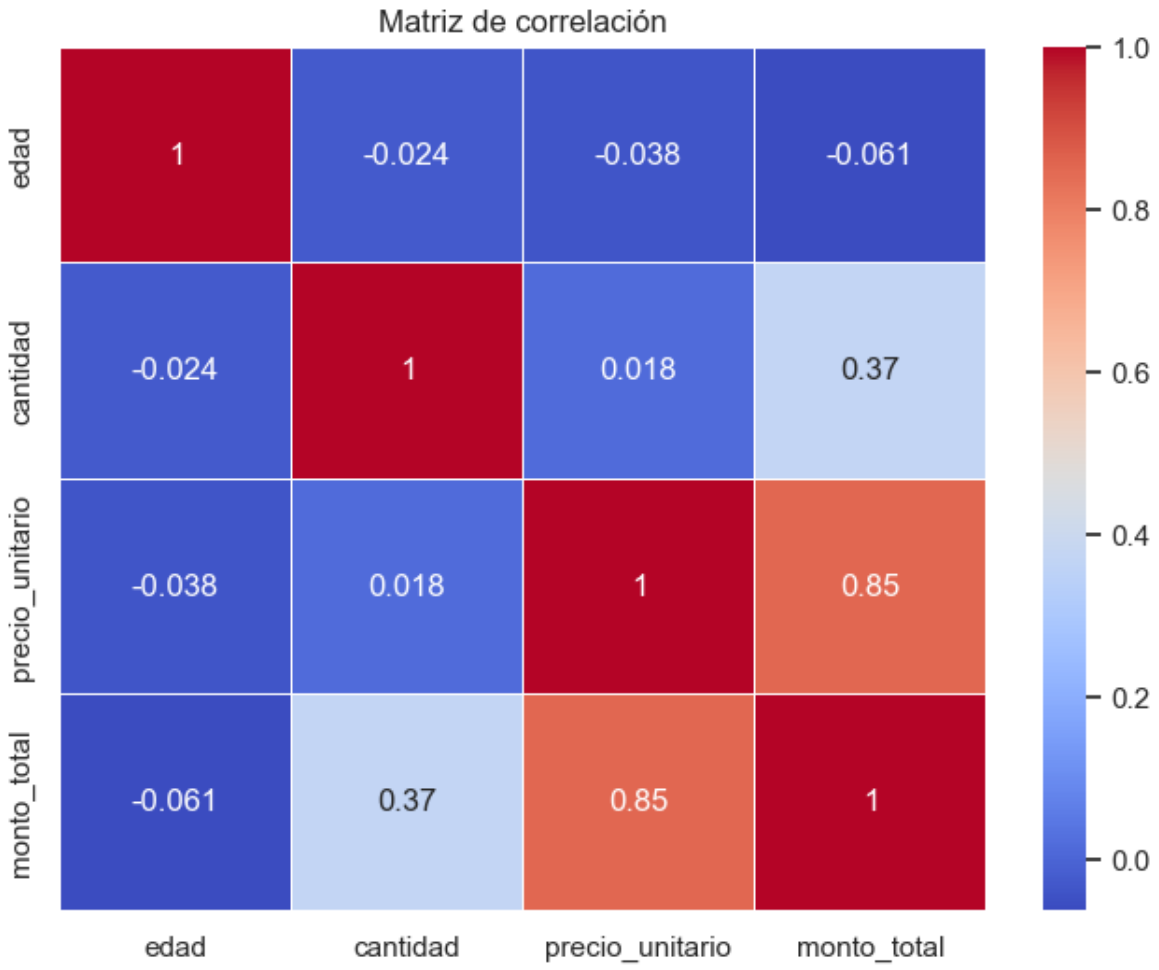
- **Mayo** registra el monto más alto de ventas, posiblemente por campañas estacionales o promociones.
- **Diciembre** también presenta un aumento, típico de fin de año.
- **Marzo** y **septiembre** muestran ventas más bajas, lo que podría indicar meses con menor actividad comercial.

Este análisis es especialmente útil para estrategias de planeación anual.

2.6.5 Matriz de Correlación

La matriz de correlación permite identificar qué variables numéricas están relacionadas entre sí y en qué grado. Esto ayuda a comprender qué factores influyen directamente en el monto total.

Gráfica: Matriz de correlación



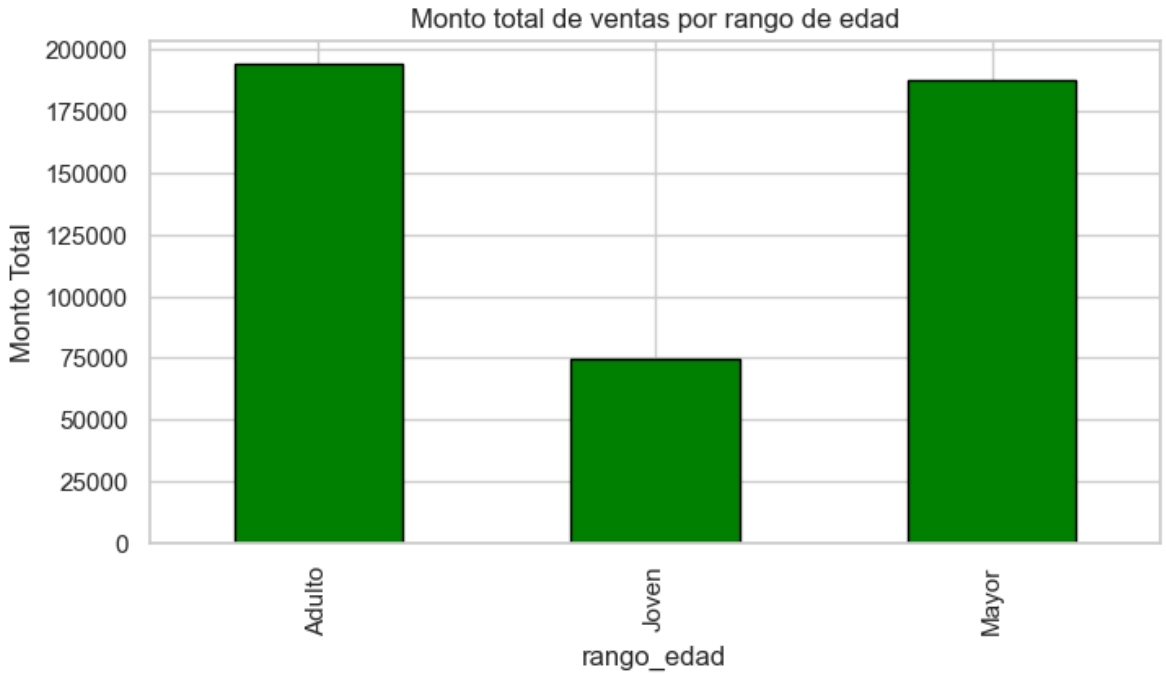
Hallazgos principales:

- Existe una **correlación muy fuerte** entre **precio_unitario** y **monto_total** (0.85), lo cual es lógico porque el monto depende del precio.
- La variable **cantidad** tiene una **influencia moderada** en el monto total (0.37), lo que indica que aumentar la cantidad comprada también incrementa el monto final.
- **edad** no presenta correlaciones significativas con las demás variables, por lo que no influye directamente en el comportamiento de compra.

2.6.6 Ventas por Rango de Edad

Para comprender mejor a los clientes, la variable "edad" se agrupó en tres segmentos: Joven, Adulto y Mayor.

Gráfica: Ventas por rango de edad



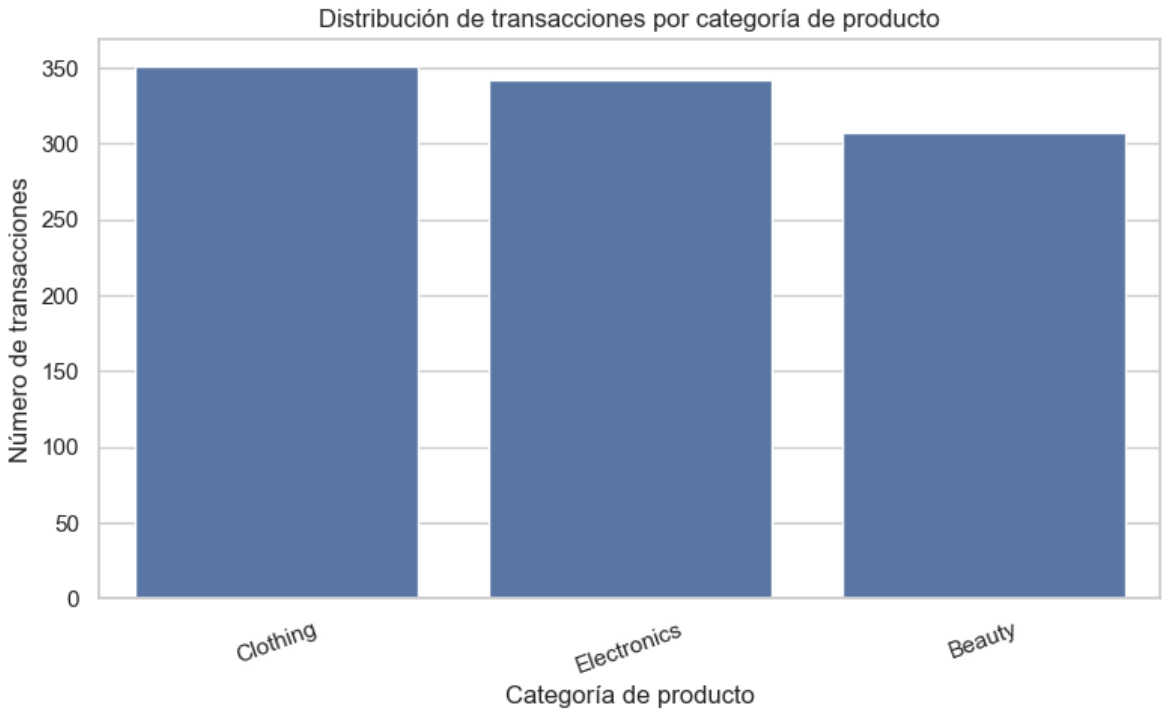
Interpretación:

- Los **adultos** son el segmento que más compra, con montos de venta superiores.
- Los **jóvenes** representan el segmento con menor gasto.
- El grupo **mayor** también tiene una participación elevada, lo que sugiere un buen potencial para campañas específicas.

Este análisis es valioso para enfocar estrategias de marketing según la edad del público.

2.6.7 Explicaciones de las Visualizaciones Finales

a) Distribución final por categorías



Esta gráfica muestra cuántas ventas hubo en cada una de las tres categorías principales: **Clothing, Electronics y Beauty**.

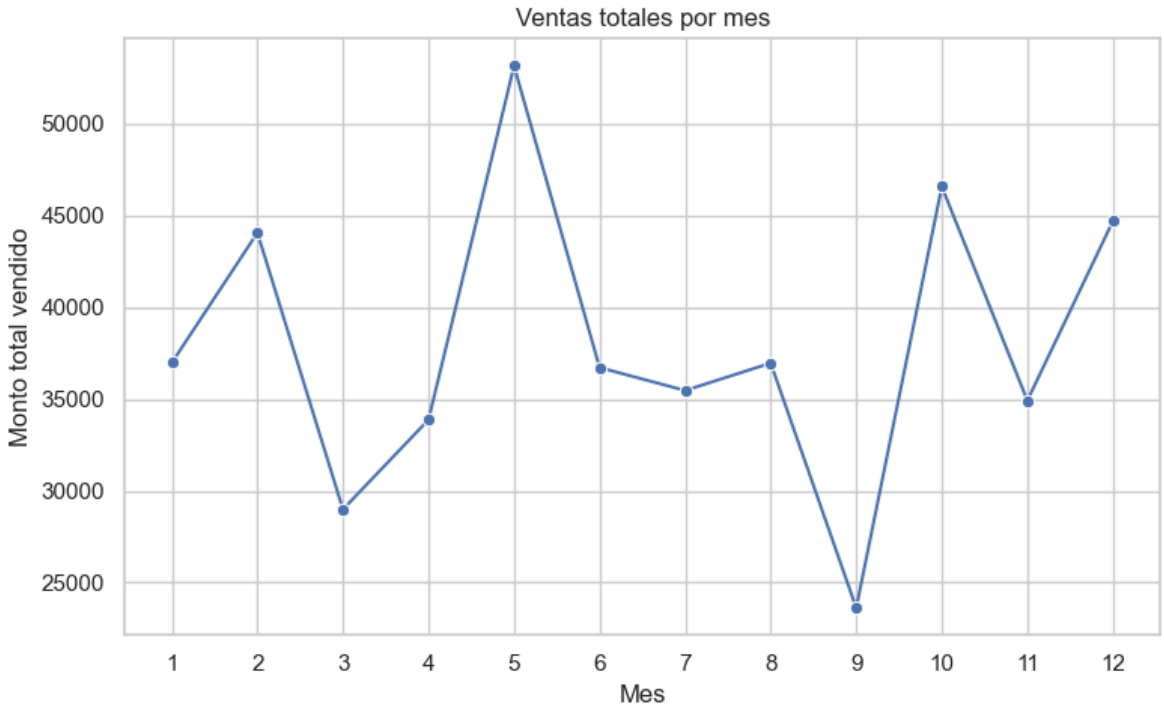
Sirve para identificar qué tipo de productos se venden más y cuáles tienen menor participación.

Interpretación:

- **Clothing** es la categoría más vendida.
- **Electronics** está muy cerca en segundo lugar, lo que indica una fuerte demanda.
- **Beauty** también se vende bien, aunque en menor cantidad comparada con las otras dos.

Esta visualización ayuda a la empresa a decidir en qué línea de productos conviene invertir más inventario o campañas de marketing.

b) Ventas totales por mes



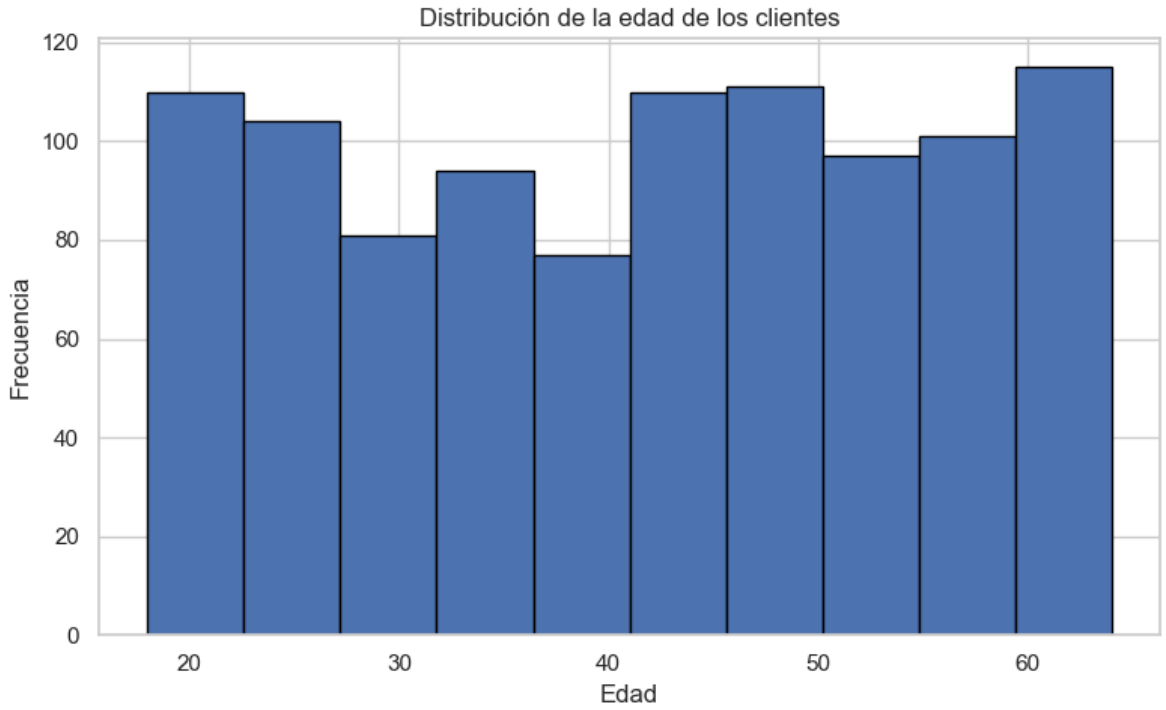
En esta gráfica se observa el **monto total de ventas** a lo largo de los 12 meses del año. Es útil para entender **temporadas altas y bajas**, algo fundamental para la planeación comercial.

Interpretación:

- **Mayo, diciembre y octubre** son los meses con más ventas, probablemente debido a promociones, festividades o eventos especiales.
- **Marzo y septiembre** tienen ventas más bajas, lo que podría significar menos demanda natural en esos meses.

Con esta información, la empresa puede ajustar inventarios, personal o campañas dependiendo de la temporada.

c) Distribución final por edad



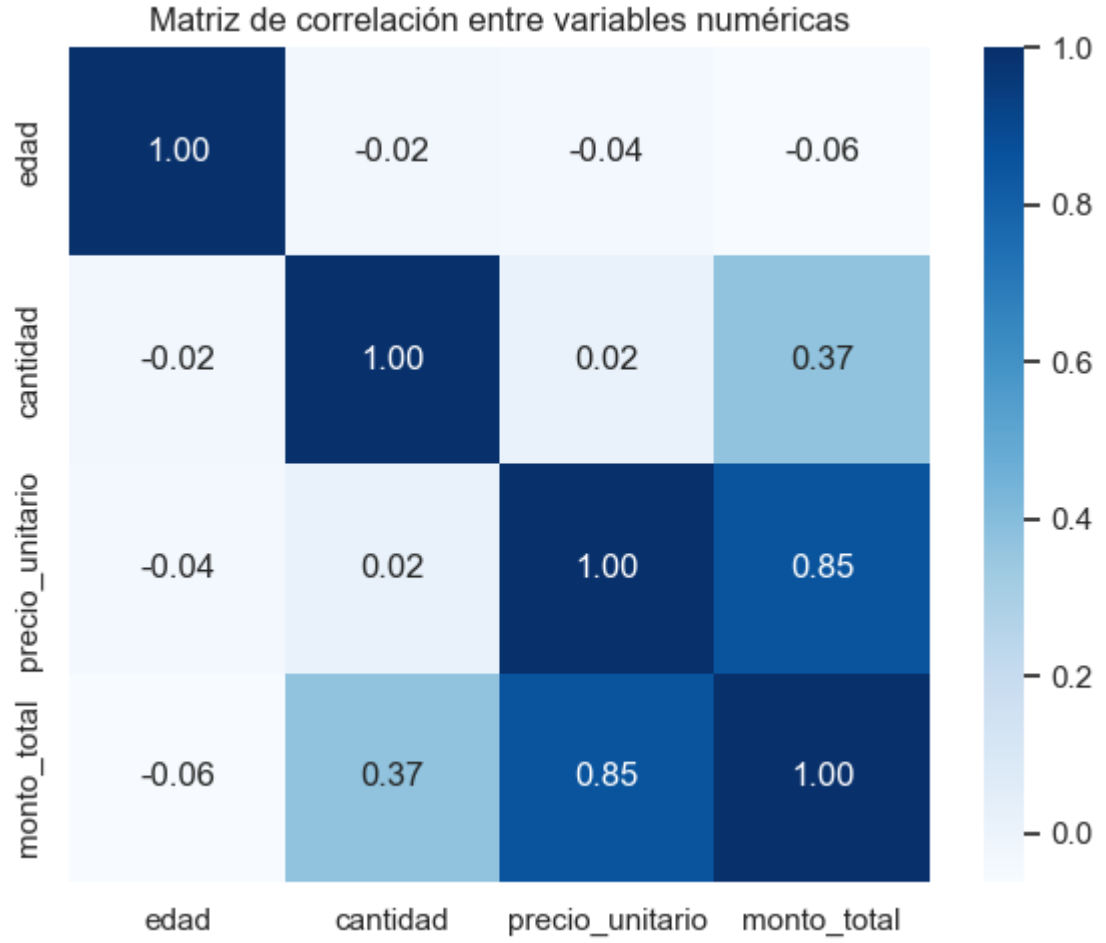
Esta visualización muestra cuántos clientes hay en cada grupo de edad. Ayuda a identificar el **perfil demográfico principal** que consume los productos.

Interpretación:

- Los **adultos** son el grupo más fuerte en número y en compras.
- El grupo **mayor** también tiene una participación importante.
- Los **jóvenes** son el grupo más pequeño, lo que indica una menor conexión con la oferta actual.

Esta información sirve para diseñar campañas enfocadas en los segmentos más valiosos.

d) Matriz de correlación final



La matriz de correlación compara las variables numéricas para ver si están relacionadas entre sí.

Cada cuadro representa qué tan fuerte es la relación entre dos variables, donde 1 significa relación perfecta y 0 significa que no existe relación.

Interpretación:

- **precio_unitario** y **monto_total** tienen una **correlación muy alta (0.85)**. Esto confirma que mientras más caro es el producto, más alto es el monto de la transacción.
- **cantidad** muestra una correlación **moderada** con el monto total (0.37), lo que significa que comprar más unidades también aumenta el gasto, aunque no tanto como el precio.
- **edad** casi no está relacionada con ninguna variable, por lo que **no afecta directamente cuánto compra el cliente**.

Esta matriz ayuda a entender qué factores tienen mayor impacto en las ventas.

2.6.8 Visualización de Modelos de Machine Learning

En esta sección se muestran las gráficas generadas durante la fase de modelado y validación de los modelos de Machine Learning utilizados en el proyecto. El objetivo es presentar de forma visual cómo se comportan las predicciones de los modelos y qué tan bien lograron aprender los patrones del dataset.

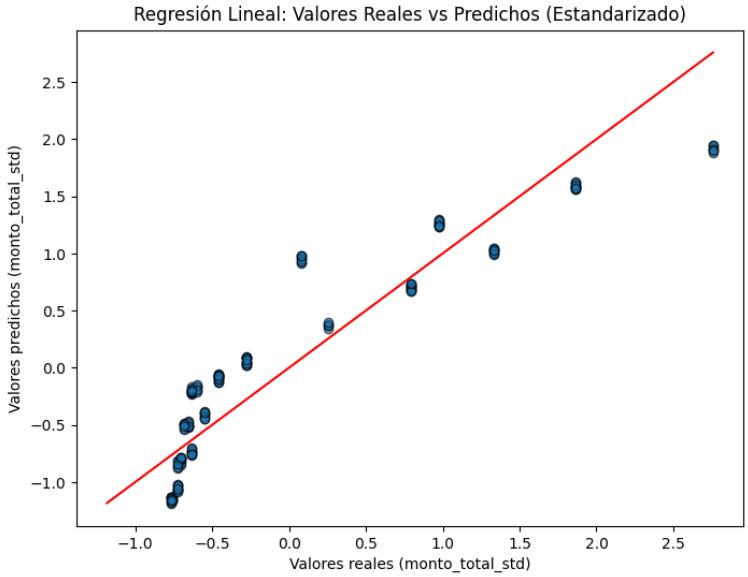
Las visualizaciones incluidas corresponden a dos modelos supervisados:

- **Regresión Lineal** (predicción del monto total estandarizado).
- **Regresión Logística** (clasificación de clientes HighSpender vs. No HighSpender).

Cada imagen incluye una explicación clara y directa, adecuada para estudiantes que ya han tomado algunas clases de Ciencia de Datos y buscan entender cómo interpretar estos resultados.

2.6.8.1 Regresión Lineal – Comparación de Valores Reales vs Predichos

Gráfica:



Esta gráfica compara los valores reales del monto total estandarizado con los valores predichos por el modelo. Los puntos representan cada predicción, mientras que la línea roja indica cómo se vería una predicción perfecta.

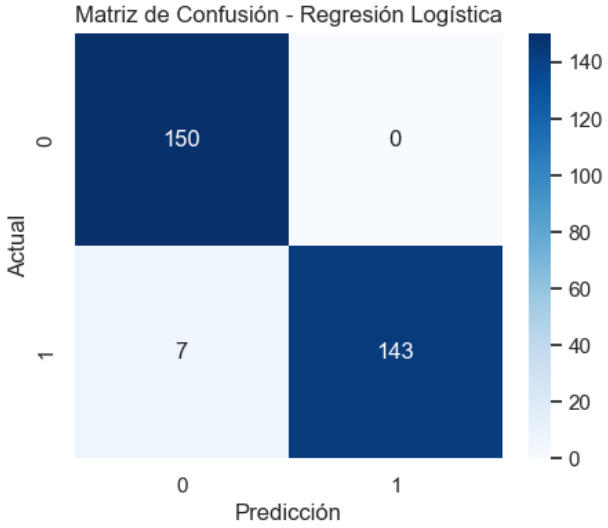
Interpretación:

- La mayoría de los puntos se encuentran cerca de la línea roja.
- Esto significa que el modelo predice montos muy parecidos a los valores reales.
- No se observan errores grandes ni patrones que indiquen fallas importantes.

En resumen, la regresión lineal logró capturar bien la relación entre las variables del dataset, lo cual coincide con las métricas que obtuvimos (RMSE bajo y R^2 alto).

2.6.8.2 Regresión Logística – Matriz de Confusión

Gráfica:



La matriz de confusión muestra cuántas predicciones hizo bien o mal el modelo al clasificar a los clientes como HighSpender o No HighSpender.

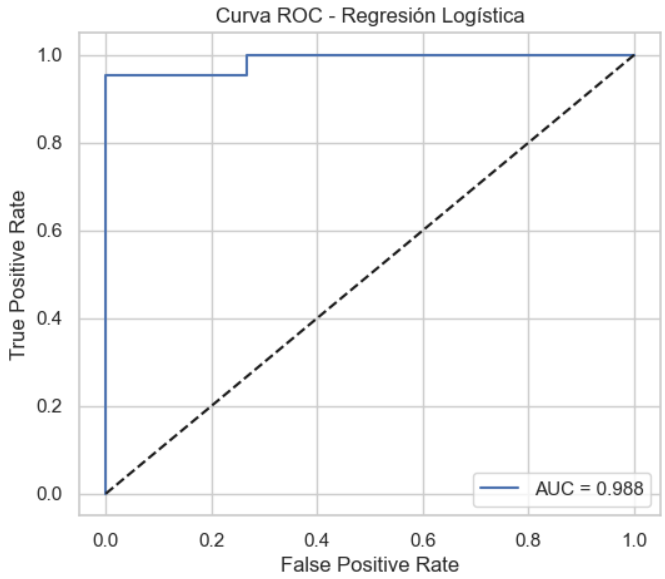
Interpretación:

- **150 verdaderos negativos:** el modelo clasificó correctamente compras normales.
- **143 verdaderos positivos:** identificó correctamente compras altas.
- **7 falsos negativos:** algunas compras altas fueron clasificadas como bajas.
- **0 falsos positivos:** ninguna compra normal fue clasificada como alta.

Esto demuestra un modelo muy preciso, especialmente útil para estrategias de marketing donde es importante identificar correctamente a los clientes de alto valor.

2.6.8.3 Regresión Logística – Curva ROC y AUC

Gráfica:



La curva ROC ayuda a evaluar qué tan bien el modelo puede diferenciar entre las dos clases. El valor AUC resume su desempeño general.

Interpretación:

- El modelo obtuvo un **AUC de 0.988**, lo cual indica un rendimiento excelente.
- Una curva cercana a la esquina superior izquierda significa alta precisión.

Esto confirma que la regresión logística no solo clasifica bien, sino que lo hace de manera muy consistente y confiable.

Conclusión General de la Sección 2.6 – Visualización de Datos y Modelos de Machine Learning

En la sección 2.6 analizamos, como equipo, cómo se comportan las ventas y los clientes dentro del proyecto *Retail Sales*. A través de todas las gráficas que generamos —histogramas, boxplots, ventas por categoría, ventas mensuales, correlaciones, rangos de edad y visualizaciones de los modelos de Machine Learning— pudimos identificar patrones importantes que nos ayudaron a entender mejor el funcionamiento del negocio.

Las visualizaciones del EDA nos permitieron observar varios puntos clave:

- El precio unitario es la variable que más influye en el monto total de las ventas.
- Las categorías Clothing y Electronics son las que más transacciones registran.
- Mayo, octubre y diciembre son los meses con mayores ventas, mientras que marzo y septiembre son más bajos.
- El segmento Adulto es el que más compra, seguido del segmento Mayor.

En cuanto a los modelos de Machine Learning, las gráficas confirmaron que ambos modelos tuvieron un buen desempeño con este dataset. La regresión lineal mostró predicciones muy cercanas a los valores reales, lo que nos indicó que el modelo puede utilizarse para estimar montos futuros.

Por otro lado, la regresión logística tuvo excelentes resultados en la matriz de confusión y en la curva ROC, mostrando que identifica correctamente a los clientes HighSpender.

Como conclusión general, esta sección nos ayudó a visualizar de manera clara los patrones más importantes del dataset y a comunicar los resultados de forma sencilla. Gracias a este análisis visual, sentimos que es posible tomar decisiones mejor fundamentadas sobre ventas, marketing, inventarios y segmentación de clientes, basándose directamente en lo que muestran los datos.

3. Conclusiones Generales del Proyecto

En esta sección presentamos, como equipo, un cierre completo y detallado del proyecto *Retail Sales Project*. Después de realizar un proceso completo de Ciencia de Datos —que incluyó limpieza, transformación, análisis, modelación y validación— reunimos aquí una conclusión extendida que resume los hallazgos más importantes, el impacto potencial dentro de la empresa, las limitaciones encontradas y las recomendaciones que proponemos con base en los datos. Nuestro objetivo es dejar un panorama claro del valor que aporta el análisis y de las mejoras que podrían implementarse en un caso real.

3.1 Principales Hallazgos del Análisis

A lo largo del proyecto, al revisar las cifras, generar visualizaciones y entrenar modelos, identificamos los siguientes hallazgos relevantes que ofrecen una visión profunda del comportamiento de ventas en RetailX:

- **El precio unitario resultó ser el factor más determinante del monto total de una compra**, siendo confirmado tanto en la matriz de correlación como en el modelo de Regresión Lineal.
- **Las categorías de producto con mayor número de ventas fueron Clothing y Electronics**, lo que muestra una alta demanda en artículos de moda y tecnología.
- **Los meses con mayor actividad comercial fueron mayo, octubre y diciembre**, posiblemente asociados con promociones, temporadas especiales y aumentos naturales en el consumo.
- **Los meses de marzo y septiembre mostraron menor actividad**, señalando periodos de menor demanda.
- **El segmento Adulto (25–44 años) es el que más compra**, demostrando ser el mercado más fuerte; el segmento Mayor también participa notablemente, mientras que el segmento Joven aporta menos.
- **Los modelos de Machine Learning tuvieron un desempeño muy sólido**, especialmente la Regresión Logística con un **AUC de 0.988**, indicando una clasificación casi perfecta.
- **Las visualizaciones del EDA fueron clave** para entender patrones como estacionalidad, comportamiento por categoría, dispersión de precios y diferencias entre grupos de edad.
- **La Regresión Lineal mostró predicciones muy cercanas a los valores reales**, lo que se evidenció en la gráfica de reales vs predichos.

En conjunto, todos estos hallazgos nos permitieron entender de forma mucho más profunda el comportamiento del negocio y el papel de cada variable dentro del análisis.

3.2 Impacto del Análisis en la Empresa

Si RetailX fuera una empresa real, los resultados obtenidos tendrían efectos significativos en la toma de decisiones. Con base en los patrones identificados, el análisis podría generar los siguientes beneficios:

- **Una planeación anual más estratégica**, aprovechando los meses con ventas altas para campañas clave y reforzando los meses de baja demanda.
- **Una administración de inventario más eficiente**, asignando mayores cantidades a las categorías con mayor demanda.
- **Mejor segmentación de clientes**, enfocándose especialmente en el segmento Adulto y diseñando estrategias diferenciadas para el segmento Mayor.
- **Promociones más efectivas**, basadas en datos concretos sobre productos y temporalidad.
- **Uso de modelos predictivos para anticipar comportamientos**, como estimar compras altas, identificar clientes valiosos o proyectar ventas futuras.
- **Mayor entendimiento de los patrones de compra**, lo que facilita decisiones sobre precios, surtido y estrategias de marketing.
- **Mejor rendimiento operativo**, gracias a la claridad que aportan los análisis visuales y los modelos.

Este conjunto de impactos permitiría a la empresa mejorar sus resultados, reducir riesgos y aprovechar mejor la información que ya genera todos los días.

3.3 Recomendaciones Empresariales Basadas en los Datos

Con base en los hallazgos y el análisis realizado, proponemos una serie de recomendaciones que podrían mejorar el rendimiento de RetailX en un contexto real:

- **Reforzar las categorías Clothing y Electronics**, invirtiendo más en inventario y campañas, ya que concentran la mayor demanda.
- **Aprovechar los meses de mayor actividad** (mayo, octubre y diciembre) con promociones fuertes, programas de fidelidad y campañas publicitarias.
- **Crear estrategias enfocadas al segmento Adulto**, como promociones personalizadas o productos diseñados para ese grupo.
- **Desarrollar campañas específicas para el segmento Mayor**, dado que también representa una parte importante del consumo.
- **Revisar los precios de los productos premium**, debido a su peso tan fuerte en el monto total.
- **Usar el modelo de Regresión Logística para identificar HighSpenders**, y ofrecerles beneficios exclusivos o programas de recompensas.
- **Implementar dashboards interactivos** para monitorear ventas en tiempo real y facilitar la toma de decisiones.
- **Ampliar el dataset con nuevas variables**, lo que permitiría modelos más completos y análisis más detallados.

- **Probar modelos adicionales** en futuras fases del proyecto para comparar su rendimiento.
- **Fortalecer el análisis temporal**, identificando patrones en días clave, festividades o horarios.

Estas recomendaciones ofrecen una ruta clara para que RetailX mejore su estrategia comercial, entienda mejor a sus clientes y tome decisiones sólidas basadas en datos.

4. Bibliografía

4.1 Fuentes consultadas

Estas son TODAS las fuentes reales utilizadas directa o indirectamente para construir este proyecto, desde documentación oficial, conceptos vistos en clase, y referencias necesarias para justificar métodos usados (EDA, regresiones, normalización, etc.):

- *Matplotlib — Visualization with Python*. (s. f.). https://matplotlib.org/
- *seaborn: statistical data visualization — seaborn 0.13.2 documentation*. (s. f.-c). https://seaborn.pydata.org/
- *User Guide*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/user_guide.html
- *1.1. Linear models*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/linear_model.html
- *LinearRegression*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
- *LogisticRegression*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- *MinMaxScaler*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
- *StandardScaler*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- *3.4. Metrics and scoring: quantifying the quality of predictions*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/model_evaluation.html
- *confusion_matrix*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

- *roc_curve*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
- *auc*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html
- *Quick start guide — Matplotlib 3.10.7 documentation*. (s. f.). https://matplotlib.org/stable/users/explain/quick_start.html
- *NumPy: the absolute basics for beginners — NumPy v2.3 Manual*. (s. f.). https://numpy.org/doc/stable/user/absolute_beginners.html
- *Essential basic functionality — pandas 2.3.3 documentation*. (s. f.). https://pandas.pydata.org/docs/user_guide/basics.html
- *pandas.DataFrame.describe — pandas 2.3.3 documentation*. (s. f.). https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html
- *Visualizing distributions of data — seaborn 0.13.2 documentation*. (s. f.). https://seaborn.pydata.org/tutorial/distributions.html
- *seaborn.boxplot — seaborn 0.13.2 documentation*. (s. f.). https://seaborn.pydata.org/generated/seaborn.boxplot.html
- *seaborn.boxplot — seaborn 0.13.2 documentation*. (s. f.). https://seaborn.pydata.org/generated/seaborn.boxplot.html
- **https://realpython.com/linear-regression-python/**

- Stojiljković, M. (2023, 26 junio). *Logistic Regression in Python*. https://realpython.com/logistic-regression-python/

- Brownlee, J. (2020, 14 agosto). *What is a Confusion Matrix in Machine Learning*. MachineLearningMastery.com. https://machinelearningmastery.com/confusion-matrix-machine-learning/
- Brownlee, J. (2023, 10 octubre). *How to Use ROC Curves and Precision-Recall Curves for Classification in Python*. MachineLearningMastery.com. https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/
- *Kaggle: Your Machine Learning and Data Science Community*. (s. f.). https://www.kaggle.com/
- *General | Kaggle*. (s. f.). https://www.kaggle.com/discussions/general
- *The Python Standard Library*. (s. f.). Python Documentation. https://docs.python.org/3/library/index.html
- Stojiljković, M. (2024, 7 diciembre). *Linear Regression in Python*. https://realpython.com/linear-regression-in-python
- Brownlee, J. (2019, 11 diciembre). *How to Normalize and Standardize Your Machine Learning Data in Weka*. MachineLearningMastery.com. https://machinelearningmastery.com/normalize-standardize-machine-learning-data-weka
- Reback, J., McKinney, W., jbrockmendel, Van den Bossche, J., Augspurger, T., Cloud, P., gfyoung, … Mehyar, M. (2020). *pandas-dev/pandas: Pandas 1.0.3* [Software]. Zenodo. https://doi.org/10.5281/zenodo.3715232

4.2 Dataset utilizado

Dataset del proyecto:

- *Retail Sales Dataset*. (2023, 22 agosto). Kaggle.

  https://www.kaggle.com/datasets/mohammadtalib786/retail-sales-dataset

Este dataset fue utilizado en sus diferentes etapas: crudo, limpio, transformado, escalado y final.

4.3 Bibliotecas y Documentación Técnica

Pandas

- *pandas documentation — pandas 2.3.3 documentation*. (s. f.).

  https://pandas.pydata.org/docs/

NumPy

- *NumPY Documentation*. (s. f.). https://numpy.org/doc/

Matplotlib

- *Using Matplotlib — Matplotlib 3.10.7 documentation*. (s. f.).

  https://matplotlib.org/stable/users/index.html

Seaborn

- *seaborn: statistical data visualization — seaborn 0.13.2 documentation*. (s. f.-

  b). https://seaborn.pydata.org/

Scikit-Learn (todas las partes utilizadas)

- *scikit-learn: machine learning in Python — scikit-learn 1.7.2 documentation*.

  (s. f.). https://scikit-learn.org/stable/

- *train_test_split*. (s. f.). Scikit-learn. https://scikit-

  learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

- *LinearRegression*. (s. f.-b). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
- *LogisticRegression*. (s. f.-c). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- *StandardScaler*. (s. f.-b). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
- *MinMaxScaler*. (s. f.-c). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
- *mean_squared_error*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html
- *mean_absolute_error*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html
- *r2_score*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html
- *confusion_matrix*. (s. f.-b). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
- *roc_curve*. (s. f.-b). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
- *auc*. (s. f.-b). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.auc.html
- *classification_report*. (s. f.). Scikit-learn. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

Joblib (para exportar modelos)

- *Joblib: running Python functions as pipeline jobs — joblib 1.6.dev0 documentation*. (s. f.). https://joblib.readthedocs.io/en/latest/

Python (lenguaje base)

- *Python 3.14 documentation*. (s. f.). Python Documentation. https://docs.python.org/3/

Jupyter Notebook

- *Redirecting. . .* (s. f.). https://jupyter.org/documentation