Obtencion y Preparacion de Datos

Caso_1: Análisis de Ventas con NumPy Optimizando la Gestión de Ventas en un Retail

La empresa Comercial XYZ tiene tiendas en diferentes ciudades y desea analizar sus ventas de la última semana para optimizar su estrategia comercial. Los datos recopilados incluyen:
• Un vector con los precios de los productos vendidos: [15000, 22000, 18000, 25000, 30000]
• Una matriz que representa la cantidad de unidades vendidas de cada producto en los últimos cinco días:
• [[5, 8, 6, 7, 10],
• [3, 6, 5, 4, 7],
• [8, 12, 10, 9, 11],
• [4, 5, 6, 3, 4],
• [6, 7, 8, 5, 9]]
La gerencia necesita responder las siguientes preguntas clave:
1. **Cálculo de ingresos diarios**: Multiplica la matriz de unidades vendidas por el vector de precios de los productos para obtener los ingresos diarios de cada producto.
2. **Ingresos totales por día**: Suma los ingresos diarios para conocer la recaudación total de la tienda por cada día.
3. **Día con mayores ingresos**: Determina cuál fue el día con la mayor recaudación y cuánto se vendió en total.
4. **Productos más rentables**: Identifica cuáles fueron los dos productos que generaron más ingresos en la semana.
5. **Filtrado de ventas altas**: Utiliza selección condicional para identificar los días en los que las ventas totales superaron los $400.000.

Preguntas Clave:

1. ¿Cuál es el ingreso total de cada producto por día?

   Ingreso total de cada producto por día:
   Producto 1: Día 1: $75,000, Día 2: $45,000, Día 3: $120,000, Día 4: $60,000, Día 5: $90,000
   Producto 2: Día 1: $176,000, Día 2: $132,000, Día 3: $264,000, Día 4: $110,000, Día 5: $154,000
   Producto 3: Día 1: $108,000, Día 2: $90,000, Día 3: $180,000, Día 4: $108,000, Día 5: $144,000
   Producto 4: Día 1: $175,000, Día 2: $100,000, Día 3: $225,000, Día 4: $75,000, Día 5: $125,000
   Producto 5: Día 1: $300,000, Día 2: $210,000, Día 3: $330,000, Día 4: $120,000, Día 5: $270,000

2. ¿Cuál fue el día con mayores ingresos y cuánto se vendió?

Total por día:
Día 1: $834,000
Día 2: $577,000
Día 3: $1,119,000
Día 4: $473,000
Día 5: $783,000

El día con mayores ingresos fue el Día 3 con un total de $1,119,000 en ventas.

3. ¿Cuáles fueron los productos que generaron más ingresos en la semana?

Total semanal por producto:
Producto 1: $390,000
Producto 2: $836,000
Producto 3: $630,000
Producto 4: $700,000
Producto 5: $1,230,000

Productos que generaron más ingresos en la semana:
1. Producto 5 con $1,230,000
2. Producto 2 con $836,000

4. ¿En qué días la tienda logró ventas superiores a $400,000?

Días con ventas superiores a $400,000:
Día 1, Día 2, Día 3, Día 4, Día 5

5. ¿Cómo podría usar la empresa estos datos para mejorar su estrategia comercial**?**

Recomendaciones para mejorar la estrategia comercial:

- Invertir en promoción de los productos más rentables.
- Analizar qué factores ayudaron al éxito del Día 3 y replicarlos.
- Reforzar inventario y personal los días con alta demanda.
- Evaluar la estrategia de productos menos vendidos para optimizar recursos.

E_Colab

```
import numpy as np
 # Datos iniciales
 precios = np.array([15000, 22000, 18000, 25000, 30000])
 ventas = np.array([
 [5, 8, 6, 7, 10],
 [3, 6, 5, 4, 7],
 [8, 12, 10, 9, 11],
 [4, 5, 6, 3, 4],
 [6, 7, 8, 5, 9]
 ])
 # Ingresos por producto y día
 ingresos_por_producto_dia = ventas.T * precios[:, np.newaxis]

# Ingresos totales por día
 ingresos_totales_dia = ingresos_por_producto_dia.sum(axis=0)
 # Ingresos totales por producto
 ingresos_totales_producto = ingresos_por_producto_dia.sum(axis=1)
 # Día con mayores ingresos
 dia_mayor = np.argmax(ingresos_totales_dia) + 1
 monto_mayor = ingresos_totales_dia[dia_mayor - 1]
 # Productos más rentables (Top 2)
 productos_top = np.argsort(ingresos_totales_producto)[-2:][::-1] + 1
 # Días con ingresos > $400.000
 dias_mas_400k = np.where(ingresos_totales_dia > 400000)[0] + 1

# RESULTADOS DETALLADOS
 print("\n INFORME SEMANAL DE VENTAS - COMERCIAL XYZ\n")
 # Detalle por producto y día
 print("Ingreso total de cada producto por día:")
 for i, ingresos in enumerate(ingresos_por_producto_dia):
 print(f"   Producto {i+1}: " + ', '.join([f"Día {j+1}: ${v:,.0f}" for j, v in
enumerate(ingresos)]))

# Detalle de totales por producto
 print("\n Total semanal por producto:")
 for i, total in enumerate(ingresos_totales_producto):
 print(f"   Producto {i+1}: ${total:,.0f}")

# Detalle de totales por día
 print("\n Total por día:")
 for i, total in enumerate(ingresos_totales_dia):
 print(f"   Día {i+1}: ${total:,.0f}")

# Día con mayor ingreso
```

```
 print(f"\n El día con mayores ingresos fue el Día {dia_mayor} con un total de
${monto_mayor:,.0f} en ventas.")

# Productos más rentables
 print("\n Productos que generaron más ingresos en la semana:")
 for i, p in enumerate(productos_top, 1):
 print(f"
{i}. Producto {p} con ${ingresos_totales_producto[p - 1]:,.0f}")

# Días con ventas > $400.000
print("\n Días con ventas superiores a $400,000:")
if dias_mas_400k.size > 0:
    print("   " + ', '.join([f"Día {d}" for d in dias_mas_400k]))
else:
    print("   No hubo días con ingresos mayores a $400,000.")
```

```
INFORME SEMANAL DE VENTAS - COMERCIAL XYZ
 Ingreso total de cada producto por día:
   Producto 1: Día 1: $75,000, Día 2: $45,000, Día 3: $120,000, Día 4: $60,000, Día 5:
$90,000
   Producto 2: Día 1: $176,000, Día 2: $132,000, Día 3: $264,000, Día 4: $110,000, Día 5:
$154,000
   Producto 3: Día 1: $108,000, Día 2: $90,000, Día 3: $180,000, Día 4: $108,000, Día 5:
$144,000
   Producto 4: Día 1: $175,000, Día 2: $100,000, Día 3: $225,000, Día 4: $75,000, Día 5:
$125,000
   Producto 5: Día 1: $300,000, Día 2: $210,000, Día 3: $330,000, Día 4: $120,000, Día 5:
$270,000
Total semanal por producto:
   Producto 1: $390,000
   Producto 2: $836,000
   Producto 3: $630,000
   Producto 4: $700,000
   Producto 5: $1,230,000
 Total por día:
   Día 1: $834,000
   Día 2: $577,000
   Día 3: $1,119,000
   Día 4: $473,000
   Día 5: $783,000
 El día con mayores ingresos fue el Día 3 con un total de $1,119,000 en ventas.
Productos que generaron más ingresos en la semana:
   1. Producto 5 con $1,230,000
   2. Producto 2 con $836,000
 Días con ventas superiores a $400,000:
   Día 1, Día 2, Día 3, Día 4, Día 5
```

Caso_2: Análisis de Datos con Pandas en el Contexto Laboral

Ana es analista de ventas en una empresa de tecnología. Su equipo ha recopilado datos sobre las ventas de productos durante los últimos seis meses y necesita analizarlos para tomar decisiones informadas. Los datos incluyen información sobre los productos vendidos, las fechas de compra y los montos de las transacciones.
Ana recibe el siguiente conjunto de datos en formato de DataFrame:

```
import pandas as pd
datos_ventas = {
'Producto': ['Laptop', 'Mouse', 'Teclado', 'Monitor', 'Mouse', 'Laptop', 'Monitor',
'Teclado',
'Laptop', 'Mouse'],
'Precio': [800, 20, 50, 200, 25, 850, 220, 55, 780, 22],
'Fecha': ['2024-01-05', '2024-01-07', '2024-01-10', '2024-01-15', '2024-02-01', '2024-
02
05', '2024-02-07', '2024-02-10', '2024-02-15', '2024-02-20']
}
df = pd.DataFrame(datos_ventas)
df['Fecha'] = pd.to_datetime(df['Fecha'])
print(df)
```

El equipo de Ana necesita responder las siguientes preguntas clave para presentar un informe al director de ventas.

Preguntas Clave:

1. **Filtrado de datos**: Filtra y muestra solo las ventas de Laptops. ¿Cuántas Laptops se vendieron y cuál fue el monto total generado por su venta?

```
# 1. Filtrar ventas de Laptops
ventas_laptop = df[df['Producto'] == 'Laptop']
cantidad_laptops = ventas_laptop.shape[0]
total_laptops = ventas_laptop['Precio'].sum()

print(ventas_laptop)
print(f"Laptops vendidas: {cantidad_laptops}")
print(f"Total generado por laptops: ${total_laptops}")
```

Salida:

```
  Producto  Precio      Fecha
0   Laptop     800 2024-01-05
5   Laptop     850 2024-02-05
8   Laptop     780 2024-02-15
Laptops vendidas: 3
Total generado por laptops: $2430
```

2. **Selección de datos**: Extrae y muestra solo las columnas de 'Producto' y 'Precio' para analizar la variación de precios de los productos.

```
producto_precio = df[['Producto', 'Precio']]
print(producto_precio)
```

Salida:

```
  Producto  Precio
0   Laptop     800
1    Mouse      20
2  Teclado      50
3  Monitor     200
4    Mouse      25
5   Laptop     850
6  Monitor     220
7  Teclado      55
8   Laptop     780
9    Mouse      22
```

3. **Análisis de fechas**: Identifica en qué mes se realizaron más ventas y cuáles fueron los productos más vendidos en ese periodo.

```
ventas_mes_top = df[df['Mes'] == mes_mas_ventas]
productos_mas_vendidos = ventas_mes_top['Producto'].value_counts()

print(f"Mes con más ventas: {mes_mas_ventas}")
print("Productos más vendidos en ese mes:")
print(productos_mas_vendidos)
```

Salida:

```
Mes con más ventas: 2024-02
Productos más vendidos en ese mes:
Producto
Mouse      2
Laptop     2
Monitor    1
Teclado    1
Name: count, dtype: int64
```

4. **Exploración de datos**: Usa los métodos head(), info() y describe() para describir el DataFrame y explicar qué información proporcionan.

- **head() :**Muestra las primeras filas (vista rápida del DataFrame)
- print(df.head())

Salida:

	Producto	Precio	Fecha	Mes
0	Laptop	800	2024-01-05	2024-01
1	Mouse	20	2024-01-07	2024-01
2	Teclado	50	2024-01-10	2024-01
3	Monitor	200	2024-01-15	2024-01
4	Mouse	25	2024-02-01	2024-02

- **info():** Tipo de datos, cantidad de nulos, estructura general.
- print(df.info())

Salida:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Producto  10 non-null     object
 1   Precio    10 non-null     int64
 2   Fecha     10 non-null     datetime64[ns]
 3   Mes       10 non-null     period[M]
dtypes: datetime64[ns](1), int64(1), object(1), period[M](1)
memory usage: 452.0+ bytes
None
```

- **describe()** :Estadísticas descriptivas (media, min, max, cuartiles) sobre columnas numéricas.
- print(df.describe())

Salida:

	Precio	Fecha
count	10.000000	10
mean	302.200000	2024-01-28 02:24:00
min	20.000000	2024-01-05 00:00:00
25%	31.250000	2024-01-11 06:00:00
50%	127.500000	2024-02-03 00:00:00
75%	640.000000	2024-02-09 06:00:00
max	850.000000	2024-02-20 00:00:00
std	357.939411	NaN

5. **Selección condicional**: Encuentra todas las ventas donde el precio del producto supera los $100. ¿Cuáles son esos productos y cuál es su total de ventas?

```
ventas_mayores_100 = df[df['Precio'] > 100]
total_monto_altos = ventas_mayores_100['Precio'].sum()
print("Ventas Mayores a 100")
print(ventas_mayores_100 [['Producto', 'Precio']])
print()
print(f"Total de ventas > $100: ${total_monto_altos}")
```

Salida:

```
Ventas Mayores a 100
  Producto  Precio
0   Laptop     800
3  Monitor     200
5   Laptop     850
6  Monitor     220
8   Laptop     780

Total de ventas > $100: $2850
```

Caso_3: Extracción y Manipulación de Datos en un Entorno Laboral

María es analista de datos en una empresa de retail que vende productos en línea. Su equipo ha recibido la tarea de analizar las ventas del último trimestre para identificar tendencias y optimizar el inventario. Los datos se encuentran en distintos formatos:

- Un archivo CSV con el registro de ventas detallado.
- Un archivo Excel con reportes financieros generados por el área contable.
- Datos sobre precios de productos extraídos de una página web de la competencia.

Desafío:
María debe consolidar toda esta información en un solo informe que permita visualizar el comportamiento de las ventas y compararlas con los precios de la competencia. Para ello, debe realizar las siguientes tareas:

1. Cargar los datos desde el archivo CSV y analizar el total de ventas por categoría de producto.
2. Leer el archivo Excel y calcular el margen de ganancia promedio de cada categoría.
3. Extraer los precios de productos desde la web de la competencia utilizando herramientas de scraping.
4. Comparar los precios de la competencia con los datos de ventas y margen de ganancia para determinar si es necesario ajustar los precios de la empresa.
5. Generar un archivo Excel con los resultados obtenidos para compartirlo con el equipo de marketing y ventas.

Preguntas Clave:

1. ¿Qué librerías de Python utilizarías para realizar la extracción y manipulación de datos en este caso? Justifica tu respuesta.

LIBRERIA	USO ESPECIFICO
pandas	Cargar, manipular y analizar datos tabulares (CSV, Excel).
openpyxl	Motor para leer/escribir archivos Excel (.xlsx) con pandas.
request	Descargar contenido HTML desde la web para scraping.
BeautifulSoup	Parsear y extraer datos de HTML fácilmente
re	Usar expresiones regulares para limpiar y encontrar patrones.
datetime	Manejar y comparar fechas (por ejemplo, para definir trimestres).
matplotlib o seaborn	Para generar gráficos si se requiere visualización.

2. ¿Cómo podrías leer los archivos CSV y Excel en Python? Explica el procedimiento con ejemplos de código.
   - Leer CSV

```
import pandas as pd

ventas = pd.read_csv('ventas_trimestre.csv')
print(ventas.head())
```

- Leer Excel
  reportes = pd.read_excel('reportes_financieros.xlsx', engine='openpyxl')
  print(reportes.head())

  Tip: puedes usar sheet_name='NombreHoja' si el archivo de Excel tiene múltiples hojas.

3. ¿Qué ventajas ofrece la extracción de datos desde la web en comparación con otras fuentes? ¿Qué desafíos podrías encontrar al hacerlo?

Ventajas:

- Acceso a información actualizada en tiempo real (precios de la competencia, stock, tendencias).
- Fuente gratuita y pública sin depender de proveedores cerrados.

Desafíos:

- Los sitios web pueden tener estructuras dinámicas o cambiar repentinamente.
- Es posible enfrentar restricciones legales o técnicas (robots.txt, CAPTCHAs).
- Scraping requiere validaciones rigurosas para asegurar que los datos son precisos.

4. ¿Cómo podrías automatizar el proceso para que María no tenga que repetir manualmente estos pasos cada trimestre?

Crear un script modular y reutilizable.

```
def cargar_datos_csv(ruta):
    return pd.read_csv(ruta)

def cargar_excel(ruta):
    return pd.read_excel(ruta, engine='openpyxl')

def scraping_precios(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    # Lógica de extracción según estructura HTML
    return precios_dict

def procesar_y_guardar():
    ventas = cargar_datos_csv('ventas.csv')
    reportes = cargar_excel('reportes.xlsx')
    precios = scraping_precios('https://competencia.com/productos')
    # Comparación y análisis...
    resultado.to_excel('reporte_final.xlsx', index=False)
```

También se podria usar Jupyter Notebooks como base de reportes automatizados
y también se podría usar un scheduler como cron o tareas programadas en Windows.

5. ¿Cómo garantizarías la calidad y precisión de los datos extraídos antes de analizarlos?

- Validar tipos: asegurarse de que las fechas sean fechas, precios sean numéricos,etc.
- Detectar duplicados o nulos:
  df.isnull().sum()
  df.duplicated().sum()
- Normalización: estandarizar nombres de productos, categorías o monedas.
- Cruce lógico: verificar si los datos entre fuentes coinciden o tienen anomalías.
- Pruebas unitarias: si se automatiza, escribir pruebas para validar cada paso.

Caso_4: Manejo de Valores Perdidos y Outliers en el Análisis de Datos Análisis de Ventas en una Empresa de Retail

La empresa "Comercial Delta" cuenta con una base de datos de ventas que ha sido recopilada durante el último año. Sin embargo, el equipo de análisis ha identificado algunos problemas en los datos:

- Existen registros de ventas en los que la variable "método de pago" no está especificada.
- En la columna "Edad del Cliente", algunos valores aparecen como nulos.
- Se detectaron valores atípicos en la variable "Monto de Compra", donde la mayoría de las compras oscilan entre $10 y $500, pero hay registros con montos superiores a $10,000.

El equipo directivo ha solicitado un informe que analice la situación y proponga una solución basada en las mejores prácticas para el manejo de datos perdidos y outliers.

Preguntas Clave:

1. ¿Cuáles son las principales problemáticas en la calidad de los datos de la empresa "Comercial Delta"?

- Valores perdidos (missing values) en:
  - "método de pago": puede afectar análisis de comportamiento de pago o detección de fraudes.
  - "Edad del Cliente":limita segmentaciones demográficas.
- Outliers extremos en "Monto de Compra" que podrían distorsionar promedios, desviaciones estándar y decisiones de negocio.
- Posible inconsistencia de formatos o codificaciones (aunque no se detalla explícitamente, suele acompañar este tipo de problemas).

2. ¿Qué técnicas utilizarías para identificar los valores perdidos en la base de datos?

```
# Contar valores nulos por columna
df.isnull().sum()

# Porcentaje de valores nulos
df.isnull().mean() * 100

# Visualizaciones útiles
import seaborn as sns
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
```

Estas técnicas permiten cuantificar y visualizar fácilmente los vacíos para orientar la estrategia de limpieza.

3. ¿Qué estrategias podrías aplicar para tratar los valores perdidos en las variables "método de pago" y "Edad del Cliente"? Justifica tu respuesta.

Método de Pago (Categoría):
Moda (valor más frecuente)

df['Método de Pago'].fillna(df['Método de Pago'].mode()[0], inplace=True)

- Justificación: Es una variable categórica, y asumir el valor más común suele ser una buena aproximación si la cantidad de valores nulos no es muy alta.

Edad del Cliente (Numérica):
- Imputación por mediana

df['Edad del Cliente'].fillna(df['Edad del Cliente'].median(), inplace=True)

-Justificación: La mediana es robusta frente a valores extremos y es ideal si la distribución está sesgada.

4. ¿Cómo identificarías los outliers en la variable "Monto de Compra" y qué acción recomendarías en este caso?

```
Identificación con IQR
q1 = df['Monto de Compra'].quantile(0.25)
q3 = df['Monto de Compra'].quantile(0.75)
iqr = q3 - q1
limite_superior = q3 + 1.5 * iqr

outliers = df[df['Monto de Compra'] > limite_superior]
```

Alternativas de tratamiento:
- Validar con el equipo de negocio si son errores o compras reales.
- Si se confirma que son errores: eliminarlos

df_filtrado = df[df['Monto de Compra'] <= limite_superior]

Si son válidos pero extremos se puede:
- Aplicar técnicas de escalamiento robusto.
- Usar transformaciones logarítmicas para mitigar su efecto en modelos.

5. Propón una metodología para asegurar la calidad de los datos en futuras recopilaciones y evitar estos problemas.

- Validaciones de entrada desde sistemas fuente:
  - Campos obligatorios (por ejemplo: método de pago nunca vacío).
  - Rango de edad razonable (18–100).
  - Límites superiores para montos según productos.
- ETL con limpieza automática periódica:
  - Scripts en Python que detecten, reporten y corrijan valores inconsistentes o nulos.
  - Almacén intermedio para auditoría antes del copiado final.
- Documentación clara del esquema de datos:
  - Definir tipos, rangos válidos, y reglas de negocio por variable.
- Tableros de monitoreo con alertas:
  - Generar visualizaciones automáticas que detecten desviaciones significativas en edad, monto o volumen de ventas.
- Entrenamiento al personal que ingresa los datos:
  - Asegurar una cultura de calidad desde el origen.

Caso_5: Data Wrangling en el Contexto Laboral Optimización de Datos en una Empresa de Retail

La empresa Comercial Express es una cadena de retail que gestiona miles de transacciones diarias en todas sus sucursales. Recientemente, el equipo de análisis de datos ha detectado inconsistencias en la base de datos de clientes y ventas, lo que ha generado reportes inexactos y decisiones erróneas.
Al revisar los datos, encontraron los siguientes problemas:

- Registros duplicados de clientes, con pequeñas variaciones en los nombres.
- Valores nulos en la columna de "Monto de Compra".
- Fechas de transacción en diferentes formatos.
- Categorías de productos inconsistentes (por ejemplo, "Electrónica" y "electronica").
- Datos almacenados como texto en columnas que deberían ser numéricas.

El equipo necesita realizar una limpieza y organización de los datos para garantizar que los análisis reflejen la realidad y permitan tomar decisiones estratégicas adecuadas. Para ello, deberán aplicar técnicas de Data Wrangling, utilizando herramientas como Pandas en Python.

Preguntas Clave:
1. ¿Qué técnicas de limpieza de datos deberían aplicarse para corregir los problemas detectados en la base de datos de Comercial Express?

Para los problemas detectados, se pueden aplicar:

- Normalización de texto: convertir nombres y categorías a minúsculas, eliminar tildes y espacios extra.
- Eliminación de duplicados: usando drop_duplicates() con criterios específicos.
- Imputación de valores nulos: con la media, mediana o incluso modelos predictivos si el contexto lo permite.
- Conversión de tipos de datos: con pd.to_numeric() y pd.to_datetime() para columnas mal tipificadas.
- Estandarización de categorías: usando mapeos (replace()) o funciones personalizadas para unificar etiquetas.

2. ¿Cómo se puede manejar la eliminación de registros duplicados sin afectar la calidad de los datos?

Cuando los duplicados no son idénticos (por ejemplo, "Juan Pérez" vs "Juan Perez"):

- Aplicar normalización de nombres (minúsculas, sin tildes).
- Usar drop_duplicates(subset=['nombre', 'email']) para conservar un solo registro por cliente.
- Si hay variaciones sutiles, puedes usar técnicas de fuzzy matching para detectar similitudes.

Antes de eliminar se debe, agrupa y revisa qué registros aportan más información (por ejemplo, el que tiene más compras o datos completos).

3. ¿Qué estrategia se recomienda para estandarizar las categorías de productos y garantizar coherencia en la clasificación?

Para unificar etiquetas p.ej. como "Electrónica" y "electronica":

- Convertir todo a minúsculas: df['categoria'] = df['categoria'].str.lower()
- Eliminar tildes con unidecode o expresiones regulares.
- Crear un diccionario de mapeo:

```
mapeo = {
        'electronica': 'Electrónica',
         'electrónica': 'Electrónica',
        'electrodomésticos': 'Electrodomésticos'
}
df['categoria'] = df['categoria'].replace(mapeo)
```

- También puedes usar pd.Categorical para definir un conjunto cerrado de categorías válidas.

4. ¿Qué métodos podrían usarse para convertir los valores incorrectos en las columnas numéricas y corregir los formatos de fecha?

- Para columnas numéricas mal tipificadas (como texto):

```
df['monto'] = pd.to_numeric(df['monto'], errors='coerce')
```

- Para fechas en distintos formatos:

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

- Revisa los valores NaT o NaN generados y decide si se imputan o eliminan.

5. ¿Cómo la limpieza y transformación de datos pueden impactar la toma de decisiones en Comercial Express?

Una base de datos limpia permite:

- Análisis precisos: sin duplicados ni errores, los KPIs reflejan la realidad.
- Segmentación efectiva: se puede identificar a los mejores clientes y productos.
- Optimización de campañas: al tener categorías y montos correctos, las estrategias de marketing son más certeras.
- Confianza en los datos: los equipos pueden tomar decisiones sin temor a errores ocultos.
- Se puede agrupar por clientes o sucursales
- Detectar patrones de compra por categorías o rango de fechas
- Automatizar con funciones y programación modular

Ejemplo en Colab:

```
import pandas as pd
import unidecode

# 1. Cargar los datos
df = pd.read_csv("ventas.csv")

# 2. Normalizar texto (nombres, categorías)
df['nombre_cliente'] = df['nombre_cliente'].str.lower().str.strip()
df['nombre_cliente'] = df['nombre_cliente'].apply(lambda x: unidecode.unidecode(x))

df['categoria'] = df['categoria'].str.lower().str.strip()
df['categoria'] = df['categoria'].apply(lambda x: unidecode.unidecode(x))

# 3. Eliminar duplicados (por nombre + email, por ejemplo)
df = df.drop_duplicates(subset=['nombre_cliente', 'email'], keep='first')

# 4. Imputar valores nulos en monto de compra
df['monto'] = pd.to_numeric(df['monto'], errors='coerce')
df['monto'].fillna(df['monto'].median(), inplace=True)

# 5. Corregir formatos de fecha
df['fecha_transaccion']=pd.to_datetime(df['fecha_transaccion'],errors='coerce',
dayfirst=True)

# 6. Estandarizar categorías con un diccionario
mapeo_categorias = {
    'electronica': 'Electrónica',
    'electrodomesticos': 'Electrodomésticos',
    'hogar': 'Hogar',
    'ropa': 'Ropa'
}
df['categoria'] = df['categoria'].replace(mapeo_categorias)

# 7. Validar columnas numéricas
columnas_numericas = ['monto', 'descuento']
for col in columnas_numericas:
    df[col] = pd.to_numeric(df[col], errors='coerce')

# 8. Resultado limpio
df.to_csv("ventas_limpias.csv", index=False)
print(" Datos limpios guardados como 'ventas_limpias.csv'")
```

Caso_6: Análisis y Transformación de Datos con Pandas

Eres analista de datos en una empresa de retail que opera en varios países. La dirección general quiere obtener un análisis claro de las ventas para identificar patrones y mejorar la estrategia comercial. Actualmente, la información está dispersa en diferentes tablas y necesita ser organizada y consolidada.

Datos disponibles:

- Un DataFrame con las ventas de productos por país y continente.
- Un DataFrame con la información de clientes y sus identificadores.
- Un DataFrame con el total de ventas realizadas por cliente.

Tareas a realizar:

1. Indexación jerárquica: Organiza los datos de ventas por continente y país para facilitar su análisis.
2. Groupby: Calcula la suma total de ventas por continente.
3. Pivoteo: Reestructura los datos para mostrar las ventas en un formato de tabla que agrupe por país y continente.
4. Combinación de datos: Fusiona los DataFrames de clientes y ventas para obtener un resumen consolidado.

Preguntas Clave:

1. ¿Cuál es la ventaja de usar indexación jerárquica en este conjunto de datos? Proporciona un ejemplo en código.

Permite organizar los datos en múltiples niveles (por ejemplo, continente → país), lo que facilita filtrado, agregaciones y visualización estructurada.

```
df_ventas = pd.DataFrame({
   'Continente': ['América', 'América', 'Europa', 'Europa'],
   'País': ['Chile', 'México', 'España', 'Francia'],
   'Ventas': [1000, 1500, 2000, 1800]
})

df_ventas.set_index(['Continente', 'País'], inplace=True)
print(df_ventas)
```

Salida:

```
                  Ventas
Continente País
América     Chile   1000
            México  1500
Europa      España  2000
            Francia 1800
```

Esto permite hacer cosas como df_ventas.loc['América'] para ver todos los países de este continente.

```
df_ventas.loc['América']
```

Salida:

```
         Ventas
País
Chile      1000
México     1500
```

2. ¿Qué insights se pueden obtener al agrupar los datos por continente utilizando la función groupby? Explica con un ejemplo.

Agrupar por continente permite ver el rendimiento regional y detectar oportunidades o rezagos.

```
ventas_por_continente = df_ventas.groupby(level='Continente').sum()
print(ventas_por_continente)
```

Insight: Europa tiene más ventas que América, se podría investigar qué productos o estrategias están funcionando mejor allá.

Salida:

```
           Ventas
Continente
América      2500
Europa       3800
```

3. ¿Cómo ayuda el pivoteo a mejorar la visualización y el análisis de los datos? Proporciona un ejemplo de código.

El método pivot() transforma los datos para facilitar comparaciones visuales entre países y continentes.

```
df_reset = df_ventas.reset_index() tabla_pivote = df_reset.pivot(index='País',
columns='Continente', values='Ventas') print(tabla_pivote)
```

Esto genera una tabla donde las columnas son continentes y las filas países, ideal para dashboards o heatmaps.

Salida:

```
Continente  América  Europa
País
Chile        1000.0     NaN
España          NaN  2000.0
Francia         NaN  1800.0
México        500.0     NaN
```

4. ¿Cuál es la diferencia entre merge y concat en la combinación de datos? Justifica tu respuesta con un caso práctico.

Método	Que Hace	Cuando Usarlo
merge	Une DataFrames por columnas clave (como JOIN en SQL)	Cuando hay relaciones entre tablas (p.e.: cliente → ventas)
Concat	Apila DataFrames por filas o columnas	Cuando quieres unir datos similares (por ejemplo, ventas de distintos meses)

```
# Merge: unir ventas con clientes
df_clientes = pd.DataFrame({'ID': [1, 2], 'Nombre': ['Ana', 'Luis']})
df_ventas = pd.DataFrame({'ID': [1, 2], 'Total': [500, 700]})
df_completo = pd.merge(df_clientes, df_ventas, on='ID')
print("<<<< MERGE >>>>>")
print("df_clinets")
print(df_clientes)
print()
print("df_ventas")
print(df_ventas)
print()
print("df_merge")
print(df_completo)

# Concat: unir ventas de enero y febrero
df_enero = pd.DataFrame({'Mes': ['Enero'], 'Ventas': [1000]})
df_febrero = pd.DataFrame({'Mes': ['Febrero'], 'Ventas': [1200]})
df_union = pd.concat([df_enero, df_febrero])
print()
print(">>>>> CONCAT <<<<<")
print("df_enero")
print(df_enero)
print()
print("df_febrero")
print(df_febrero)
print()
print("df_Concat")
print(df_union)
```

Salida:

```
<<<< MERGE >>>>>
df_clinets
   ID Nombre
0   1    Ana
1   2   Luis
```

df_ventas

	ID	Total
0	1	500
1	2	700

df_merge

	ID	Nombre	Total
0	1	Ana	500
1	2	Luis	700

>>>>> CONCAT <<<<<

df_enero

	Mes	Ventas
0	Enero	1000

df_febrero

	Mes	Ventas
0	Febrero	1200

df_Concat

	Mes	Ventas
0	Enero	1000
0	Febrero	1200

5. Después de realizar las transformaciones, ¿cuáles serían los siguientes pasos para extraer más conocimientos de estos datos?

- Segmentación de clientes: quiénes compran más por país o categoría, frecuencia de compra, ticket promedio, etc.
- Análisis temporal: cómo evolucionan las ventas por región
- Detección de outliers: países con ventas inusualmente altas o bajas
- Visualización: gráficos de barras, mapas de calor, dashboards interactivos, gráficos de dispersión, líneas de tiempo.
- Análisis predictivos: predecir ventas futuras, rotación de productos o comportamiento de clientes, optimización de inventario, prever demanda y evitar sobrestock, etc.
- Análisis geoespacial, evaluar rendimiento por ubicación, zonas de influencia, expansión de tiendas.
- Integración con fuentes externas
  - Web scraping para monitorear precios de la competencia.
  - APIs para datos económicos, climáticos o de tráfico que afecten ventas.