Bootcamp Data Engineering 2

#Week 2

Ejercicio 1: **CSV**

Archivo ventas_home.csv

Archivo ventas_home_limpio.csv

ventas_home_limpio.csv

outputs > ventas_home_limpio.csv > data

```
producto,categoria,precio,cantidad,ingresos
Manzana,Fruta,100.0,5,500.0
Banana,Fruta,80.0,0,0.0
Pera,Fruta,90.0,10,900.0
Notebook,Electrónica,50000.0,2,100000.0
Auriculares,Electrónica,1500.0,4,6000.0
Remera,Indumentaria,2000.0,3,6000.0
Pantalón,Indumentaria,3500.0,1,3500.0
Campera,Indumentaria,8000.0,2,16000.0
Tablet,Electrónica,30000.0,1,30000.0
Teclado,Electrónica,2500.0,0,0.0
Camisaco,Indumentaria,2000.0,3,6000.0
camisaco,Indumentaria,3000.0,1,3000.0
Jeans,Indumentaria,4000.0,0,0.0
Tablet,Electrónica,30000.0,15,450000.0

```

Archivo resumen_ventas.csv

resumen_ventas.csv

outputs > resumen_ventas.csv > data

```
categoria,ingresos_total,productos_distintos
Electrónica,586000.0,4
Indumentaria,34500.0,6
Fruta,1400.0,3

```

Scripts:

cargar_csv.py

cargar_csv.py > ...

```python
# cargar_csv.py
from pathlib import Path
import pandas as pd

ROOT = Path(__file__).resolve().parent
CSV = ROOT / "data" / "ventas_home.csv"

if not CSV.exists():
    raise FileNotFoundError(f"No encuentro el archivo: {CSV}")

# UTF-8 primero; si falla, probamos latin-1 (útil por tildes)
try:
    df = pd.read_csv(CSV, encoding="utf-8")
except UnicodeDecodeError:
    df = pd.read_csv(CSV, encoding="latin-1")

print("✅ Cargado OK")
print("Ruta:", CSV)
print("Shape:", df.shape)              # (filas, columnas)
print("\nPrimeras filas:")
print(df.head(10).to_string(index=False))
print("\nInfo del DataFrame:")
print(df.info())

```

limpiar.py

```python
                .str.lower()
                .str.replace(r"\s+", "_", regex=True)
)

# Chequeo mínimo
required = {"producto", "categoria", "precio", "cantidad"}
faltan = required - set(df.columns)
if faltan:
    raise KeyError(f"Faltan columnas en el CSV: {sorted(faltan)}. Encabezados: {list(df.columns)}")

# 2.1 Quitar espacios extra en producto
df["producto"] = (
    df["producto"].astype(str)
                  .str.replace(r"\s+", " ", regex=True)
                  .str.strip()
)

# 2.2 Uniformar categoria (Fruta, Electrónica, Indumentaria)
def sin_tildes(s: str) -> str:
    s = str(s)
    return "".join(c for c in unicodedata.normalize("NFKD", s) if not unicodedata.combining(c))

cat_norm = (df["categoria"].astype(str)
            .str.strip().str.lower()
            .apply(sin_tildes))

mapa = {
    "fruta": "Fruta",
    "frutas": "Fruta",
    "electronica": "Electrónica",
    "electronic": "Electrónica",
    "indumentaria": "Indumentaria",
}
df["categoria"] = cat_norm.apply(lambda x: mapa.get(x, "Otros"))

# 2.3 Reemplazar precios en texto por números
def to_number(x):
    if pd.isna(x):
        return pd.NA
    s = str(x).strip()
    s = s.replace("$", "").replace("USD", "").replace("ARS", "")
    s = re.sub(r"[.\s](?=\d{3}(?:\D|$))", "", s)  # quita miles
    s = s.replace(",", ".")                       # coma -> punto
    try:
        return float(s)
    except Exception:
        return pd.NA
```

```python

# ← línea equivalente a tu 49, usando apply para evitar warnings del linter
df["precio"] = df["precio"].apply(to_number)

# 2.4 Rellenar nulos en cantidad con 0 (asegurar int)  ← tu línea 50
df["cantidad"] = pd.to_numeric(df["cantidad"], errors="coerce").fillna(0).astype(int)

# Guardar dataset limpio SIN ingresos (solo Parte 2)
df.to_csv(CSV_OUT, index=False, encoding="utf-8")
print(f"Archivo limpio (sin ingresos) guardado en: {CSV_OUT}")

```

columna_ingresos.py

```python
# columna_ingresos.py
from pathlib import Path
import pandas as pd

ROOT = Path(__file__).resolve().parent
IN_OUT = ROOT / "outputs" / "ventas_home_limpio.csv"  # viene del Paso 2 y se actualiza acá

if not IN_OUT.exists():
    raise FileNotFoundError(f"No encuentro el archivo limpio del Paso 2: {IN_OUT}")

# Cargar robusto
try:
    df = pd.read_csv(IN_OUT, encoding="utf-8")
except UnicodeDecodeError:
    df = pd.read_csv(IN_OUT, encoding="latin-1")

# Asegurar tipos (por si el Paso 2 no forzó numéricos)
df["precio"] = pd.to_numeric(df["precio"], errors="coerce").fillna(0.0)
df["cantidad"] = pd.to_numeric(df["cantidad"], errors="coerce").fillna(0).astype(int)

# PASO 3: crear ingresos = precio * cantidad
df["ingresos"] = df["precio"] * df["cantidad"]

# Guardar (sobrescribe el limpio con la nueva columna)
df.to_csv(IN_OUT, index=False, encoding="utf-8")

print("✅ Paso 3 OK — columna 'ingresos' creada")
print(f"Archivo actualizado: {IN_OUT}")
print(df.head(10).to_string(index=False))

```

agrupar_resumen.py

```python
# agrupar_resumen.py
from pathlib import Path
import pandas as pd

ROOT = Path(__file__).resolve().parent
INP  = ROOT / "outputs" / "ventas_home_limpio.csv"   # viene de los pasos 2 y 3
OUT  = ROOT / "outputs" / "resumen_ventas.csv"

if not INP.exists():
    raise FileNotFoundError(f"Falta el archivo limpio con 'ingresos': {INP}")

# Cargar
try:
    df = pd.read_csv(INP, encoding="utf-8")
except UnicodeDecodeError:
    df = pd.read_csv(INP, encoding="latin-1")

# Si por algún motivo no existe 'ingresos', lo calculo en memoria (sin modificar el archivo)
if "ingresos" not in df.columns:
    df["precio"] = pd.to_numeric(df.get("precio"), errors="coerce").fillna(0.0)
    df["cantidad"] = pd.to_numeric(df.get("cantidad"), errors="coerce").fillna(0).astype(int)
    df["ingresos"] = df["precio"] * df["cantidad"]

# Paso 4: agrupar por categoría
resumen = (
    df.groupby("categoria", dropna=False)
      .agg(
          ingresos_total=("ingresos", "sum"),
          productos_distintos=("producto", "nunique")
      )
      .reset_index()
      .sort_values("ingresos_total", ascending=False)
)

# Paso 5: guardar
OUT.parent.mkdir(parents=True, exist_ok=True)
resumen.to_csv(OUT, index=False, encoding="utf-8")
print(f"✅ Resumen generado: {OUT}")
print(resumen)

```

Ejercicio 2: **Diccionarios y JSON**

Archivo JSON

stock.json

outputs > stock.json > ...

```
{
    "Manzana": 50,
    "Banana": 40,
    "Naranja": 25
}
```

Scripts:

stock_json.py

```python
# stock_json.py
from pathlib import Path
import json

ROOT = Path(__file__).resolve().parent
OUT_DIR = ROOT / "outputs"
OUT_DIR.mkdir(parents=True, exist_ok=True)
JSON_PATH = OUT_DIR / "stock.json"

# 1) Diccionario inicial
stock = {
    "Manzana": 50,
    "Banana": 30,
    "Pera": 20
}

# 2) Agregar producto nuevo
stock["Naranja"] = 25

# 3) Actualizar cantidad existente (ej: +10 bananas)
stock["Banana"] = stock.get("Banana", 0) + 10  # 40

# 4) Eliminar un producto (ej: Pera)
stock.pop("Pera", None)

# 5) Guardar a JSON (pretty)
with open(JSON_PATH, "w", encoding="utf-8") as f:
    json.dump(stock, f, ensure_ascii=False, indent=2)

# 6) Leer y mostrar contenido
with open(JSON_PATH, "r", encoding="utf-8") as f:
    data = json.load(f)

print("✅ stock.json generado en:", JSON_PATH)
print(json.dumps(data, ensure_ascii=False, indent=2))

```

Ejercicio 3: **APIs públicas** *(opcional)*

JSON final

todos_completados.json

outputs > todos_completados.json > ...


```
[
  {
    "userId": 1,
    "id": 4,
    "title": "et porro tempora",
    "completed": true
  },
  {
    "userId": 1,
    "id": 8,
    "title": "quo adipisci enim quam ut ab",
    "completed": true
  },
  {
    "userId": 1,
    "id": 10,
    "title": "illo est ratione doloremque quia maiores aut",
    "completed": true
  },
  {
    "userId": 1,
    "id": 11,
    "title": "vero rerum temporibus dolor",
    "completed": true
  },
  {
    "userId": 1,
    "id": 12,
    "title": "ipsa repellendus fugit nisi",
    "completed": true
  },
  {
    "userId": 1,
    "id": 14,
    "title": "repellendus sunt dolores architecto voluptatum",
    "completed": true
  },
  {
    "userId": 1,
    "id": 15,
    "title": "ab voluptatum amet voluptas",
    "completed": true
  },
  {
    "userId": 1,
    "id": 16,
    "title": "accusamus eos facilis sint et aut voluptatem",
    "completed": true
```


Scripts:

todos_api.py

```python
# todos_api.py
from pathlib import Path
import json
import requests

ROOT = Path(__file__).resolve().parent
OUT_DIR = ROOT / "outputs"
OUT_DIR.mkdir(parents=True, exist_ok=True)
OUT_PATH = OUT_DIR / "todos_completados.json"

URL = "https://jsonplaceholder.typicode.com/todos"  # endpoint público

# 1) Hacer la solicitud HTTP GET
resp = requests.get(URL, timeout=30)  # timeout por si no responde
resp.raise_for_status()               # si no es 200 OK, lanza error

# 2) Convertir la respuesta JSON (texto) a objetos Python (lista/dict)
todos = resp.json()                   # ahora es una lista de dicts

# 3) Filtrar solo los completados
completados = [t for t in todos if t.get("completed") is True]

# 4) Guardar el resultado en disco como JSON bonito
with open(OUT_PATH, "w", encoding="utf-8") as f:
    json.dump(completados, f, ensure_ascii=False, indent=2)

print(f"OK → completados: {len(completados)}")
print(f"Guardado en: {OUT_PATH}")

```