Actualizador POSManager

El Actualizador POSManager es un conjunto de scripts diseñados para facilitar el procesamiento y actualización de archivos relacionados con productos, ofertas y códigos de barras. Su objetivo es automatizar la integración de datos provenientes de diferentes fuentes, como archivos CSV y Excel, realizar cálculos de precios basados en descuentos y generar archivos finales organizados y listos para ser utilizados en el sistema POSManager.

Este conjunto de herramientas permite la fusión de datos entre múltiples archivos, la normalización de información, la asignación de valores calculados y la creación de archivos de salida en formatos específicos, como CSV o TXT. Además, garantiza que los archivos se guarden con nombres basados en la fecha, asegurando la trazabilidad de los procesos.

Con la interfaz gráfica de Tkinter, el usuario puede interactuar fácilmente con el sistema para seleccionar archivos de entrada y definir la ubicación de los archivos de salida, mientras que el manejo de errores y las validaciones aseguran que el proceso se realice de manera segura y sin contratiempos.

Main.py

Se define una aplicación gráfica utilizando `tkinter` para gestionar la selección de varios archivos (CSV, TXT, Excel), procesarlos mediante funciones externas y mostrar mensajes de éxito o error. Permite al usuario cargar archivos de entrada, realizar cálculos y generar archivos de resultados, todo a través de una interfaz sencilla con botones y cuadros de texto.

1. **Importación de librerías y funciones:**

```
import tkinter as tk

from tkinter import filedialog, messagebox

from tkinter import ttk

from libs.update_normalizer import procesar_archivos

from libs.offer_calculator import calcular_ofertas

from libs.barcode_selector import seleccionar_barcodes
```

- Se importan las librerías necesarias para crear la interfaz gráfica de usuario (`tkinter`), para manejar cuadros de diálogo de archivos (`filedialog`), mostrar mensajes (`messagebox`), y utilizar widgets mejorados (`ttk`).
- También se importan las funciones específicas que se usarán en el procesamiento de archivos (`procesar_archivos`, `calcular_ofertas`, `seleccionar_barcodes`).

2. **Funciones para seleccionar archivos:** Las siguientes funciones permiten abrir un cuadro de diálogo para seleccionar archivos y luego insertar la ruta del archivo seleccionado en un campo de texto en la interfaz.

```
def seleccionar_archivo_entrada1():

    file_path = filedialog.askopenfilename(title="Seleccionar la
consulta de la Base de Datos", filetypes=[("Archivos CSV", "*.csv"),
("Archivos Excel", "*.xlsx;*.xls")])

    entry_archivo1.delete(0, tk.END)

    entry_archivo1.insert(0, file_path)
```

- **seleccionar_archivo_entrada1**: Permite al usuario seleccionar un archivo CSV o Excel. El archivo seleccionado se muestra en `entry_archivo1`.

def seleccionar_archivo_entrada2():

```
 file_path = filedialog.askopenfilename(title="Seleccionar la lista de
Items de POSManager", filetypes=[("Archivos TXT", "*.txt"), ("Archivos
CSV", "*.csv"), ("Archivos Excel", "*.xlsx;*.xls")])

    entry_archivo2.delete(0, tk.END)

    entry_archivo2.insert(0, file_path)
```

- **seleccionar_archivo_entrada2**: Permite seleccionar archivos TXT, CSV o Excel y muestra la ruta seleccionada en `entry_archivo2`.

```
def seleccionar_archivo_propuesta():

    file_path = filedialog.askopenfilename(title="Seleccionar archivo de
Propuesta", filetypes=[("Archivos Excel", "*.xlsx;*.xls")])

    entry_propuesta.delete(0, tk.END)
```

```
    entry_propuesta.insert(0, file_path)
```

- **seleccionar_archivo_propuesta**: Permite seleccionar un archivo Excel y mostrar su ruta en `entry_propuesta`.

```
def seleccionar_archivo_codebars():

    file_path = filedialog.askopenfilename(title="Seleccionar archivo de codigo de barras", filetypes=[("Archivos CSV", "*.csv"),("Archivos Excel", "*.xlsx;*.xls")])

    entry_codebars.delete(0, tk.END)

    entry_codebars.insert(0, file_path)
```

- **seleccionar_archivo_codebars**: Permite seleccionar un archivo CSV o Excel para los códigos de barras y muestra su ruta en `entry_codebars`.

3. **Función para procesar los archivos:**

```
def procesar():

    file_path1 = entry_archivo1.get()

    file_path2 = entry_archivo2.get()

    file_propuesta = entry_propuesta.get()

    file_codebars = entry_codebars.get()

    if not file_path1 or not file_path2 or not file_propuesta:

        messagebox.showwarning("Advertencia", "Por favor, seleccione todos los archivos antes de continuar.")

        return
```

- **procesar()**: Obtiene las rutas de los archivos seleccionados. Si algún archivo no ha sido seleccionado, muestra una advertencia.

```
  try:
```

```
        output_file = procesar_archivos(file_path1, file_path2)

        items_file = calcular_ofertas(output_file, file_propuesta)

        codebars_file = seleccionar_barcodes(output_file, file_codebars)



        if items_file:

            messagebox.showinfo("Éxito", f"El archivo de resultados ha
sido guardado exitosamente")

    except ValueError as e:

        messagebox.showerror("Error", str(e))
```

- Procesa los archivos usando las funciones importadas: `procesar_archivos`, `calcular_ofertas` y `seleccionar_barcodes`. Si todo va bien, muestra un mensaje de éxito. Si ocurre un error, muestra un mensaje de error.

4. **Creación de la ventana principal:**

```
root = tk.Tk()

root.title("Procesamiento de Archivos para POSManager")
```

- Crea una ventana principal con el título "Procesamiento de Archivos para POSManager".

5. **Creación de etiquetas y campos de entrada para cada archivo:**

- Para cada archivo (consulta de base de datos, lista de items, archivo de propuesta, archivo de códigos de barras), se crean etiquetas y campos de texto (`ttk.Entry`) para que el usuario pueda ver la ruta del archivo seleccionado.

```
label_archivo1 = ttk.Label(root, text="Seleccionar la consulta de la
Base de Datos (CSV delimitado por punto y coma):")

label_archivo1.grid(row=0, column=0, padx=10, pady=10)

entry_archivo1 = ttk.Entry(root, width=50)
```

```
entry_archivo1.grid(row=0, column=1, padx=10, pady=10)

button_archivo1 = ttk.Button(root, text="Buscar",
command=seleccionar_archivo_entrada1)

button_archivo1.grid(row=0, column=2, padx=10, pady=10)
```

- Este bloque se repite para cada tipo de archivo, cambiando las etiquetas y los botones correspondientes.

6. **Botón para procesar los archivos:**

```
button_procesar = ttk.Button(root, text="Procesar Archivos",
command=procesar)

button_procesar.grid(row=4, column=0, columnspan=3, pady=20)
```

- Crea un botón que llama a la función `procesar` al hacer clic, para iniciar el procesamiento de los archivos seleccionados.

7. **Iniciar el bucle principal de la interfaz gráfica:**

```
root.mainloop()
```

- Inicia el bucle principal de `Tkinter`, lo que permite que la interfaz se muestre y sea interactiva.

Este código define una aplicación gráfica que permite seleccionar varios archivos, procesarlos y mostrar mensajes sobre el estado del procesamiento.

Normalizer.py

Este bloque de código procesa dos archivos CSV/TXT, realiza un "merge" entre ellos, genera nuevos valores para las columnas necesarias, normaliza los datos y guarda el resultado en un archivo CSV con un nombre que incluye la fecha actual.

Importación de librerías:

```
import pandas as pd
```

```
import os
from datetime import datetime
```

1.
    - Se importa `pandas` para la manipulación de datos tabulares.
    - Se importa `os` para manejar la creación y verificación de directorios.
    - Se importa `datetime` para obtener la fecha actual.

Definición de la función `procesar_archivos`:

```
 def procesar_archivos(file_path1, file_path2):
```

2.
    - Define una función llamada `procesar_archivos` que recibe dos rutas de archivos como parámetros (`file_path1` y `file_path2`).

Lectura del primer archivo (df1):

```
 try:
    df1 = pd.read_csv(file_path1, sep=';', encoding='utf-8-sig',
on_bad_lines='skip')
except pd.errors.ParserError as e:
    print(f"Error de análisis al leer el archivo {file_path1}: {e}")
    exit()
except FileNotFoundError as e:
    print(f"Archivo no encontrado: {file_path1}")
    exit()
except Exception as e:
    print(f"Hubo un error al leer el archivo {file_path1}: {e}")
    exit()
```

3.
    - Intenta leer el archivo CSV `file_path1` usando `pandas`. El archivo está delimitado por punto y coma (`;`) y usa la codificación `utf-8-sig`.
    - Si ocurre un error (por ejemplo, de análisis o archivo no encontrado), se muestra un mensaje de error y se termina la ejecución del programa.

Lectura del segundo archivo (df2):

```
 try:
    df2 = pd.read_csv(file_path2, sep='\t', encoding='latin1',
```

```
on_bad_lines='skip')
except pd.errors.ParserError as e:
    print(f"Error de análisis al leer el archivo {file_path2}: {e}")
    exit()
except FileNotFoundError as e:
    print(f"Archivo no encontrado: {file_path2}")
    exit()
except Exception as e:
    print(f"Hubo un error al leer el archivo {file_path2}: {e}")
    exit()
```

4.
    - Intenta leer el archivo `file_path2` de la misma forma, pero este archivo está delimitado por tabulaciones (`\t`) y usa la codificación `latin1`.

Normalización de nombres de columnas:

```
 df1.columns = df1.columns.str.strip()
df2.columns = df2.columns.str.strip()
```

5.
    - Elimina los espacios en blanco alrededor de los nombres de las columnas en ambos DataFrames.

Verificación de las columnas necesarias:

```
 required_columns_df1 = ['IDProducto']
required_columns_df2 = ['CodigoERP']

missing_columns_df1 = [col for col in required_columns_df1 if col not in
df1.columns]
missing_columns_df2 = [col for col in required_columns_df2 if col not in
df2.columns]

if missing_columns_df1:
    print(f"Error: Las siguientes columnas no se encuentran en df1: {',
'.join(missing_columns_df1)}")
    exit()

if missing_columns_df2:
    print(f"Error: Las siguientes columnas no se encuentran en df2: {',
'.join(missing_columns_df2)}")
    exit()
```

6.
    - Verifica que los archivos contengan las columnas necesarias (`IDProducto` en `df1` y `CodigoERP` en `df2`).
    - Si alguna columna falta, muestra un mensaje de error y termina la ejecución.

Renombrar columnas para unificar:

```
df2 = df2.rename(columns={'CodigoERP': 'IDProducto'})
```

7.
    - Renombra la columna `CodigoERP` de `df2` a `IDProducto` para poder hacer el "merge" con `df1`.

Lista de columnas a mostrar:

```
columnas_a_mostrar = [...]
```

8.
    - Define una lista de columnas que se incluirán en el DataFrame final después del "merge".

Realizar el "merge" entre los DataFrames:

```
merged_df = pd.merge(df1, df2[['IDProducto', 'IdItem']],
on='IDProducto', how='left')
```

9.
    - Realiza una fusión (`merge`) entre `df1` y `df2` usando la columna `IDProducto` como clave.
    - El `how='left'` asegura que se conserven todas las filas de `df1`, incluso si no hay coincidencias en `df2`.

Reordenar las columnas:

```
result = merged_df[columnas_a_mostrar]
```

10.
    - Reorganiza las columnas del DataFrame fusionado según el orden especificado en `columnas_a_mostrar`.

Generar nuevos valores para `IdItem`:

```
 existing_ids = set(df2['IdItem'].dropna().astype(int))
empty_id_indices = result[result['IdItem'].isna()].index
required_ids = len(empty_id_indices)
new_ids = (id for id in range(8000, 8000 + required_ids +
len(existing_ids)) if id not in existing_ids)
```

11.
    - Crea un conjunto de valores existentes en `IdItem` para evitar duplicados.
    - Encuentra las filas que no tienen valor en `IdItem` y calcula cuántos valores nuevos se necesitan.
    - Crea un generador para asignar nuevos valores de ID únicos, comenzando desde 8000.

Asignar nuevos valores de `IdItem`:

```
 for index in empty_id_indices:
    codigo_interno = result.at[index, 'codigoInterno']
    if pd.notna(codigo_interno) and int(codigo_interno) < 8000:
        result.at[index, 'IdItem'] = int(codigo_interno)
    else:
        result.at[index, 'IdItem'] = next(new_ids)
result['IdItem'] = result['IdItem'].fillna(0).astype(int)
```

12.
    - Asigna un nuevo valor de `IdItem` a las filas que no tienen uno. Si `codigoInterno` es menor que 8000, se usa ese valor; de lo contrario, se asigna un nuevo valor generado.

Renombrar columna y eliminar `codigoInterno`:

```
 result = result.drop(columns=['codigoInterno'])
result.rename(columns={'IdItem': 'codigoInterno'}, inplace=True)
```

13.
    - Elimina la columna `codigoInterno` y renombra `IdItem` como `codigoInterno` para ajustarse al formato final.

Guardar el archivo de salida:

```
 output_dir =
os.path.expanduser('~\\Documents\\PM-offer-updater\\processed-files')
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
fecha_hoy = datetime.today().strftime('%Y-%m-%d')
output_file = os.path.join(output_dir, f"Items-{fecha_hoy}.csv")
result.to_csv(output_file, index=False)
return output_file
```

14.
    - Crea la carpeta de salida si no existe.
    - Genera un nombre de archivo con la fecha actual.
    - Guarda el DataFrame final como un archivo CSV en la carpeta especificada y retorna la ruta del archivo guardado.

Resumen:

Este bloque de código procesa dos archivos de entrada, realiza una fusión de datos entre ellos, genera nuevos valores para ciertas columnas, reordena las columnas y guarda el resultado en un archivo CSV con un nombre basado en la fecha actual.

Calculator.py

Este bloque de código permite procesar y calcular descuentos sobre productos, leer archivos de datos (CSV y Excel), normalizar texto, realizar cálculos con precios, agregar nuevas columnas a los datos, y finalmente exportar un archivo con los resultados calculados en formato de texto.

Importación de librerías:

```
 import pandas as pd
import tkinter as tk
from tkinter import filedialog, messagebox
import unicodedata
```

1.
    - Se importa `pandas` para la manipulación de datos tabulares.
    - Se importa `tkinter` y sus submódulos para la interfaz gráfica (diálogos de archivo y mensajes de error).
    - Se importa `unicodedata` para la normalización de texto y eliminación de caracteres no deseados.

Definición de la función `normalizar_texto`:

```
def normalizar_texto(texto):
    if isinstance(texto, str):
        return ''.join(
            c for c in unicodedata.normalize('NFKD', texto)
            if not unicodedata.combining(c)
        )
    return texto
```

2.
    - Esta función toma un texto y lo normaliza eliminando los caracteres diacríticos (acentos y signos) para asegurar que se manejen de manera uniforme.

Definición de la función `calcular_ofertas`:

```
def calcular_ofertas(output_file, archivo_propuesta):
```

3.
    - Define la función principal que recibe como parámetros el archivo de salida `output_file` y el archivo de propuesta `archivo_propuesta`.

Manejo de excepciones al cargar archivos:

```
try:
    items_df = pd.read_csv(output_file)
    propuesta_df = pd.read_excel(archivo_propuesta)
except Exception as e:
    messagebox.showerror("Error", f"Error al leer los archivos: {e}")
    return
```

4.
    - Intenta cargar los archivos `CSV` y `Excel`. Si ocurre un error, muestra un mensaje de error y termina la ejecución de la función.

Normalización de texto en todas las columnas:

```
items_df = items_df.applymap(normalizar_texto)
propuesta_df = propuesta_df.applymap(normalizar_texto)
```

5.
    - Aplica la normalización del texto a todas las columnas de ambos DataFrames `items_df` y `propuesta_df`.

Conversión de la columna de precios a `float`:

```
 items_df[items_df.columns[8]] =
items_df[items_df.columns[8]].replace({',': '.'},
regex=True).astype(float)
```

6.
    - Reemplaza las comas por puntos en la columna de precios y convierte la columna en tipo `float` para poder realizar operaciones matemáticas.

Verificación de la existencia de columnas necesarias:

```
 if "IDProducto" not in items_df.columns or propuesta_df.columns[0] !=
"Id Quantio":
    messagebox.showerror("Error", "¡Faltan las columnas IDProducto o
codigo_ERP!")
    return
```

7.
    - Verifica que las columnas `IDProducto` y `Id Quantio` estén presentes en los DataFrames. Si falta alguna, muestra un mensaje de error y termina la ejecución.

Verificación de la columna `Precio_de_Oferta_Etiquetas`:

```
 if "Precio_de_Oferta_Etiquetas" not in items_df.columns:
    messagebox.showerror("Error", "¡La columna
Precio_de_Oferta_Etiquetas no existe en el archivo Items!")
    return
```

8.
    - Verifica que la columna `Precio_de_Oferta_Etiquetas` esté presente en `items_df`. Si no, muestra un mensaje de error.

Definición de la función `calcular_precio_final`:

```
def calcular_precio_final(id_producto, precio_final):
```

...

9.
    - Define una función interna para calcular el precio final de un producto, aplicando un descuento basado en el archivo de propuesta.

Aplicación de la función de cálculo de precio final:

```
descuentos = items_df.apply(
    lambda row: calcular_precio_final(row["IDProducto"],
row[items_df.columns[8]]), axis=1
)
```

10.
    - Aplica la función `calcular_precio_final` a cada fila del DataFrame `items_df` para calcular el precio final con el descuento correspondiente.

Agregar la columna de descuento (`Precio_de_Oferta_Etiquetas`):

```
items_df.iloc[:, 23] = [precio for precio, _ in descuentos]
```

11.
    - Asigna los precios calculados en la columna `Precio_de_Oferta_Etiquetas` en el índice 23 del DataFrame.

Agregar columna `Es Oferta`:

```
if len(items_df.columns) <= 35:
    items_df.insert(35, 'Es Oferta', "")
items_df.iloc[:, 35] = ["S" if descuento else "N" for _, descuento in
descuentos]
```

12.
    - Verifica si la columna `Es Oferta` existe. Si no, la agrega en la posición 35 y asigna un valor de "S" o "N" dependiendo de si hay descuento.

Conversión de las columnas al formato adecuado:

```
items_df.iloc[:, [8, 22, 23]] = items_df.iloc[:, [8, 22,
```

```
23]].apply(pd.to_numeric, errors='coerce')
items_df.iloc[:, [8, 22, 23]] = items_df.iloc[:, [8, 22, 23]].fillna(0)
# Opcional: Manejo de NaN
```

13.
    - Convierte las columnas relevantes a formato numérico y maneja los valores `NaN` con un valor de 0.

Guardar archivo actualizado:

```
 ruta_guardado = filedialog.asksaveasfilename(
    title="Guardar archivo Items actualizado",
    defaultextension=".txt",
    filetypes=[("Archivos de Texto", "*.txt")]
)
if ruta_guardado:
    items_df.to_csv(ruta_guardado, index=False, header=False, sep='\t',
encoding='utf-16', float_format="%.2f")
    return True
else:
    messagebox.showwarning("Cancelado", "La exportación fue cancelada.")
    return False
```

14.
    - Abre un cuadro de diálogo para que el usuario elija dónde guardar el archivo. Si se elige una ruta, guarda el DataFrame como un archivo `.txt` en formato `UTF-16`. Si no, muestra una advertencia de cancelación.

Manejo de excepciones:

```
 except Exception as e:
    messagebox.showerror("Error", f"Ocurrió un error: {e}")
    return False
```

15.
    - Si ocurre cualquier otro error durante el proceso, se muestra un mensaje de error.

Resumen:

Este bloque de código gestiona el procesamiento de dos archivos (uno CSV y otro Excel), normaliza texto, calcula descuentos sobre precios de productos y agrega nuevas columnas a los datos. Después, guarda el archivo actualizado en formato TXT con los cálculos

realizados. La interfaz gráfica permite al usuario elegir el lugar para guardar el archivo resultante.

Selector.py

Este bloque de código permite leer dos archivos CSV, limpiar y validar sus columnas, realizar una unión de los datos, y finalmente exportar los resultados de los códigos de barras asociados con productos a un archivo de texto.

Importación de librerías:

```
 import pandas as pd
import os
from datetime import datetime
from tkinter import filedialog, messagebox
import unicodedata
```

1.
    - Se importa `pandas` para la manipulación de datos tabulares.
    - Se importa `os` para interactuar con el sistema de archivos (crear carpetas, manejar rutas).
    - Se importa `datetime` para trabajar con fechas.
    - Se importa `tkinter` para mostrar mensajes de error y diálogos de archivo.
    - Se importa `unicodedata` aunque no se usa en este bloque.

Definición de la función `seleccionar_barcodes`:

```
def seleccionar_barcodes(output_file, barcode_query):
```

2.
    - Define la función principal `seleccionar_barcodes`, que recibe dos parámetros: `output_file` (ruta del archivo de salida) y `barcode_query` (archivo CSV con códigos de barras).

Manejo de excepciones al leer los archivos:

```
 try:
    df1 = pd.read_csv(output_file)
    df2 = pd.read_csv(barcode_query, sep=';', encoding='utf-8-sig',
on_bad_lines='skip')
except Exception as e:
    messagebox.showerror("Error", f"Error al leer los archivos: {e}")
    return
```

3.
    - Intenta leer dos archivos CSV (`output_file` y `barcode_query`) y manejar posibles errores al hacerlo. Si hay un error, muestra un mensaje de error.

Eliminación de espacios en blanco en los nombres de las columnas:

```
 df1.columns = df1.columns.str.strip()
df2.columns = df2.columns.str.strip()
```

4.
    - Elimina cualquier espacio en blanco adicional en los nombres de las columnas de ambos DataFrames (`df1` y `df2`).

Verificación de las columnas necesarias en ambos DataFrames:

```
 required_columns_df1 = ['IDProducto']
required_columns_df2 = ['IDProducto']

missing_columns_df1 = [col for col in required_columns_df1 if col not in
df1.columns]
missing_columns_df2 = [col for col in required_columns_df2 if col not in
df2.columns]

if missing_columns_df1:
    print(f"Error: Las siguientes columnas no se encuentran en df1: {',
'.join(missing_columns_df1)}")
    exit()

if missing_columns_df2:
    print(f"Error: Las siguientes columnas no se encuentran en df2: {',
'.join(missing_columns_df2)}")
    exit()
```

5.
    - Verifica que ambos DataFrames contengan la columna `IDProducto`. Si alguna falta, muestra un mensaje de error y termina la ejecución.

Realización del "merge" entre los DataFrames:

```
 merged_df = pd.merge(df1[['IDProducto', 'codigoInterno']],
df2[['IDProducto', 'Codebar']], on='IDProducto', how='left')
```

6.
    - Realiza un `merge` entre los dos DataFrames (`df1` y `df2`) usando la columna `IDProducto` como clave. Se hace un "left join", lo que significa que se mantendrán todas las filas de `df1` y se agregarán las filas de `df2` que coincidan.

Reordenar las columnas según la solicitud:

```
 result = merged_df[['codigoInterno', 'Codebar']]
```

7.
    - Selecciona solo las columnas `codigoInterno` y `Codebar` del DataFrame combinado para el resultado final.

Manejo de la carpeta de salida:

```
 output_dir =
os.path.expanduser('~\\Documents\\PM-offer-updater\\codebars')

if not os.path.exists(output_dir):
    os.makedirs(output_dir)
```

8.
    - Define la carpeta donde se guardará el archivo de salida. Si la carpeta no existe, se crea.

Generación del nombre del archivo de salida con la fecha actual:

```
 fecha_hoy = datetime.today().strftime('%Y-%m-%d')
output_file = os.path.join(output_dir, f"CodBarras-{fecha_hoy}.txt")
```

9.

- Crea el nombre del archivo de salida utilizando la fecha actual en formato `YYYY-MM-DD`.

Guardar el resultado en un archivo de texto:

```
result.to_csv(output_file, sep='\t', index=False, header=False)
```

10.
    - Guarda el DataFrame resultante (`result`) en un archivo de texto con formato tabulado (`sep='\t'`), sin los índices ni los encabezados.

Retornar la ruta del archivo guardado:

```
return output_file
```

11.
    - Devuelve la ruta del archivo de salida guardado.

Manejo de excepciones generales:

```
except Exception as e:
   messagebox.showerror("Error", f"Ocurrió un error: {e}")
   return False
```

12.
    - Si ocurre cualquier error durante el proceso, se muestra un mensaje de error y se devuelve `False`.

Resumen:

Este bloque de código lee dos archivos CSV, verifica que contengan las columnas necesarias, realiza una unión entre ellos usando `IDProducto`, y luego guarda los resultados (códigos internos y de barras) en un archivo de texto en una carpeta específica. Además, si falta alguna columna o ocurre un error durante el proceso, muestra mensajes de error adecuados.

Resumen general de la funcionalidad:

Los bloques de código proporcionados están orientados a la manipulación y procesamiento de archivos de datos relacionados con productos, ofertas y códigos de barras. A continuación, se describe la funcionalidad general de todos ellos:

1. **Lectura y procesamiento de archivos:**
    - **Archivos CSV y Excel**: Los códigos leen archivos en formatos CSV y Excel, realizando verificaciones de existencia de columnas necesarias y manejo de errores. En algunos casos, también se realizan conversiones de formato de texto (por ejemplo, reemplazo de comas por puntos decimales) y normalización de texto para eliminar caracteres especiales.
2. **Unificación y fusión de datos:**
    - Se realiza una **unión (merge)** de diferentes conjuntos de datos basándose en columnas comunes, como `IDProducto` o `CodigoERP`, para combinar información relevante de varios archivos en un único DataFrame. Este proceso incluye uniones tipo "left join", donde se mantienen todas las filas de un archivo principal, añadiendo la información correspondiente de los archivos secundarios.
3. **Cálculo y generación de nuevos valores:**
    - En varios de los bloques, se calculan nuevos valores a partir de las columnas existentes. Por ejemplo:
        - **Precios de ofertas**: Se calculan precios finales para productos a partir de descuentos (ya sean porcentuales o fijos), y se agregan columnas indicadoras como "Es Oferta".
        - **Generación de códigos de barras**: Se asignan códigos de barras a los productos mediante una combinación de columnas de códigos internos y productos, para generar un archivo final con los códigos necesarios.
4. **Transformación y formato de datos:**
    - Se realizan transformaciones para asegurar que los datos estén en el formato correcto, como convertir precios y otros valores numéricos a tipo `float`, asegurando la correcta visualización y cálculo. Se eliminan espacios innecesarios en los nombres de columnas y se aseguran las posiciones correctas de las columnas importantes.
5. **Exportación de datos:**
    - Después de procesar y modificar los datos, los resultados se exportan a archivos de salida, como **archivos CSV o TXT**, en formato tabulado (`\t`). Se generan nombres de archivo automáticamente basados en la fecha actual para mantener un registro organizado de los archivos generados.
    - Las exportaciones incluyen la opción de guardar los resultados en una carpeta específica, como `Documents\PM-offer-updater\processed-files` o `Documents\PM-offer-updater\codebars`.

6. **Interfaz de usuario y manejo de errores:**

    - Se utiliza **Tkinter** para mostrar cuadros de mensaje, lo que permite alertar al usuario sobre errores en el proceso, como la falta de columnas necesarias o problemas al guardar archivos.
    - Los archivos de salida se pueden guardar en directorios específicos que se crean automáticamente si no existen, y se permiten interacciones con el usuario a través de diálogos de selección de archivos de entrada y salida.