**Archivo Venezuela**

This is a breakdown of the tasks that can be automated and a proposed plan for one or more Python applications to help:

**1. Data Enrichment & Cleaning:**

* **Task:** Take WorldCat data (which lacks ISBNs and other metadata) and enrich it with information from other sources.
* **Task:** Clean and standardize data across fields (e.g., date formats, consistent language in subject terms).
* **Task:** Separate fields for both English and Spanish, since your metadata scheme uses separate columns.

**2. Translation:**

* **Task:** Translate descriptions, tags, relations, etc., from one language to the other.

**3. Cover Image Retrieval:**

* **Task:** Search for cover images based on the title and/or ISBN.

**4. Tag Management:**

* **Task:** Identify relevant tags based on the description and other fields.
* **Task:** Translate tags between English and Spanish.
* **Task:** Suggest standard subject headings (Library of Congress Subject Headings, etc.).

**Proposed Application Architecture:**

This is a Python-based application with modular components that can be run as separate scripts or integrated into a more comprehensive workflow. Here is a potential structure:

**Application 1: venezuelan\_archive\_toolkit.py (or similar)**

This will be the main script, and it will call the functions from the following modules:

# venezuelan\_archive\_toolkit.py

import csv

import argparse # Allows you to run this script from the command line with arguments

import os

from modules import data\_enrichment, translation\_tools, image\_retrieval, tag\_management

def main():

parser = argparse.ArgumentParser(description="Automates tasks for building a Venezuelan diaspora digital archive.")

parser.add\_argument("input\_csv", help="Path to the input CSV file (WorldCat download)")

parser.add\_argument("output\_csv", help="Path to save the processed CSV file")

parser.add\_argument("--language", choices=["en", "es"], default="en", help="Target language for translation (en or es)") # New argument for language choice

args = parser.parse\_args()

**# 1. Data Enrichment**

print("Enriching data...")

enriched\_data = data\_enrichment.enrich\_data(args.input\_csv) # Simplified call

**# 2. Translation**

print("Translating fields...")

translated\_data = translation\_tools.translate\_data(enriched\_data, args.language) # Added language argument

**# 3. Image Retrieval**

print("Retrieving cover images...")

image\_retrieval.retrieve\_images(translated\_data) # No need to return, saves directly

**# 4. Tag Management**

print("Managing tags...")

tagged\_data = tag\_management.manage\_tags(translated\_data)

**# Write the final processed data to a CSV**

print(f"Writing output to {args.output\_csv}...")

write\_csv(tagged\_data, args.output\_csv)

print("Workflow complete!")

def write\_csv(data, output\_path):

"""Writes the processed data to a CSV file."""

fieldnames = data[0].keys() # Get field names from the first item

with open(output\_path, 'w', encoding='utf-8', newline='') as csvfile:

writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

writer.writeheader()

writer.writerows(data)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Modules (stored in a modules folder):**

* **data\_enrichment.py:**

# modules/data\_enrichment.py

import csv

import requests

import json

def enrich\_data(csv\_path):

"""

Enriches data from a CSV file using external APIs (e.g., Open Library, Google Books).

Args:

csv\_path (str): Path to the input CSV file.

Returns:

list: A list of dictionaries, with enriched data.

"""

enriched\_data = []

with open(csv\_path, 'r', encoding='utf-8') as csvfile:

reader = csv.DictReader(csvfile)

for row in reader:

# Try to enrich with ISBN

if row['Source'] and "ISBN" in row['Source']: # Check for ISBN in 'Source' field

isbn = row['Source'].replace("ISBN: ", "").strip() # Extract ISBN

enriched\_row = enrich\_with\_isbn(row, isbn)

else:

# If no ISBN, try enriching with title and author

enriched\_row = enrich\_with\_title\_author(row)

enriched\_data.append(enriched\_row)

return enriched\_data

def enrich\_with\_isbn(row, isbn):

"""Enriches a row with data from the Open Library and Google Books APIs using ISBN."""

try:

open\_library\_url = f"https://openlibrary.org/isbn/{isbn}.json"

response = requests.get(open\_library\_url)

response.raise\_for\_status() # Raise HTTPError for bad responses (4xx or 5xx)

data = response.json()

row['Title'] = row['Title'] or data.get('title', '') # Update if field is empty

row['Título'] = row['Título'] or data.get('title', '')

if 'authors' in data:

author\_keys = [author['author']['key'] for author in data['authors']]

authors\_data = []

for key in author\_keys:

author\_url = f"https://openlibrary.org{key}.json"

try: # Nested try-except for individual authors

author\_response = requests.get(author\_url)

author\_response.raise\_for\_status()

author\_data = author\_response.json()

authors\_data.append(author\_data.get('name', ''))

except requests.exceptions.RequestException as e:

print(f"Error fetching author data: {e}") # Error for individual author

row['Creator'] = row['Creator'] or ', '.join(authors\_data) # Update if field is empty

# Add description if available from Open Library (may need further processing)

if 'description' in data:

description = data['description']

if isinstance(description, dict):

row['Description (EN)'] = row.get('Description (EN)', '') or description.get('value', '') # Prefer existing

else:

row['Description (EN)'] = row.get('Description (EN)', '') or description

# Try Google Books for more details

try: # Nested try-except for Google Books

google\_books\_url = f"https://www.googleapis.com/books/v1/volumes?q=isbn:{isbn}"

google\_response = requests.get(google\_books\_url)

google\_response.raise\_for\_status()

google\_data = google\_response.json()

if google\_data.get('totalItems', 0) > 0:

volume\_info = google\_data['items'][0]['volumeInfo']

row['Publisher'] = row['Publisher'] or volume\_info.get('publisher', '')

row['Date'] = row['Date'] or volume\_info.get('publishedDate', '')

# Add subjects if available

subjects = volume\_info.get('categories', [])

row['Subject (EN)'] = row['Subject (EN)'] or ', '.join(subjects)

except requests.exceptions.RequestException as e:

print(f"Error fetching Google Books data for ISBN {isbn}: {e}") # Error for Google Books

except requests.exceptions.RequestException as e:

print(f"Error fetching data for ISBN {isbn}: {e}")

except json.JSONDecodeError as e:

print(f"Error decoding JSON for ISBN {isbn}: {e}")

return row

def enrich\_with\_title\_author(row):

"""Enriches a row with data from Google Books API using title and author."""

title = row.get('Title', '')

author = row.get('Creator', '')

if not title or not author:

return row # Can't enrich without title or author

try:

query = f"intitle:{title}+inauthor:{author}"

google\_books\_url = f"https://www.googleapis.com/books/v1/volumes?q={query}"

response = requests.get(google\_books\_url)

response.raise\_for\_status()

google\_data = response.json()

if google\_data.get('totalItems', 0) > 0:

volume\_info = google\_data['items'][0]['volumeInfo']

row['Publisher'] = row['Publisher'] or volume\_info.get('publisher', '')

row['Date'] = row['Date'] or volume\_info.get('publishedDate', '')

# Add subjects if available

subjects = volume\_info.get('categories', [])

row['Subject (EN)'] = row['Subject (EN)'] or ', '.join(subjects)

row['Description (EN)'] = row.get('Description (EN)', '') or volume\_info.get('description', '') # Add Google Books description if none exists

except requests.exceptions.RequestException as e:

print(f"Error fetching Google Books data for title '{title}' and author '{author}': {e}")

except json.JSONDecodeError as e:

print(f"Error decoding Google Books JSON for title '{title}' and author '{author}': {e}")

return row

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487). Python

IGNORE\_WHEN\_COPYING\_END

**translation\_tools.py:**

# modules/translation\_tools.py

from googletrans import Translator # pip install googletrans==4.0.0-rc1

import time # Import time module

def translate\_data(data, target\_language="es"):

"""

Translates specified fields in the data using Google Translate.

Args:

data (list): A list of dictionaries to translate.

target\_language (str): The target language code ("es" for Spanish, "en" for English).

Returns:

list: A list of dictionaries with translated fields.

"""

translator = Translator() # Initialize the translator here

translated\_data = []

for row in data:

translated\_row = row.copy() # Create a copy to avoid modifying original

# Fields to translate based on target language

if target\_language == "es":

fields\_to\_translate = [("Description (EN)", "Description (ES)"),

("Tags (EN)", "Tags (ES)"),

("Relation", "Relación")] # Using tuples

elif target\_language == "en":

fields\_to\_translate = [("Description (ES)", "Description (EN)"),

("Tags (ES)", "Tags (EN)"),

("Relación", "Relation")]

else:

print(f"Unsupported target language: {target\_language}")

return data # Return original data if language is unsupported

for source\_field, dest\_field in fields\_to\_translate:

text\_to\_translate = translated\_row.get(source\_field, "") # Get text from source

if text\_to\_translate:

try:

translation = translator.translate(text\_to\_translate, dest=target\_language)

translated\_row[dest\_field] = translation.text # Set translated text

time.sleep(1) # Be nice to the API - pause for 1 second

except Exception as e:

print(f"Translation error for '{text\_to\_translate}': {e}") # Log errors

translated\_row[dest\_field] = f"Translation Failed: {e}" # Set an error message

translated\_data.append(translated\_row)

return translated\_data

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487). Python

IGNORE\_WHEN\_COPYING\_END

**image\_retrieval.py:**

# modules/image\_retrieval.py

import requests

from io import BytesIO

from PIL import Image

import os

import urllib.parse # Import urllib for URL encoding

def retrieve\_images(data, image\_folder="images"):

"""

Retrieves cover images based on title and saves them to a folder.

Args:

data (list): A list of dictionaries with book information.

image\_folder (str): The folder to save images to.

"""

if not os.path.exists(image\_folder):

os.makedirs(image\_folder)

for row in data:

title = row.get("Title", "").strip()

isbn = row.get("Source", "").replace("ISBN: ", "").strip() # Try to get ISBN

if title:

image\_path = os.path.join(image\_folder, f"{slugify(title)}.jpg") # Create the filepath

if os.path.exists(image\_path):

print(f"Image already exists for '{title}', skipping...")

continue # Skip if the image already exists

image\_url = None # Initialize outside the try block

if isbn:

image\_url = get\_cover\_from\_openlibrary(isbn) # Try Open Library first

if not image\_url: # If no ISBN or Open Library fails, try Google Books

image\_url = get\_cover\_from\_google\_books(title)

if image\_url:

try:

response = requests.get(image\_url, stream=True)

response.raise\_for\_status() # Check for HTTP errors

image = Image.open(BytesIO(response.content))

image.save(image\_path)

print(f"Saved cover image for '{title}' to {image\_path}")

row["Files (if available)"] = os.path.abspath(image\_path) # Save path in the CSV

except requests.exceptions.RequestException as e:

print(f"Error downloading image for '{title}' from {image\_url}: {e}")

except Exception as e: # Catch other image processing errors

print(f"Error processing image for '{title}' from {image\_url}: {e}")

else:

print(f"No cover image found for '{title}'.")

else:

print("No title provided, skipping image retrieval.")

def get\_cover\_from\_openlibrary(isbn):

"""Retrieves a cover image URL from Open Library based on ISBN."""

try:

url = f"https://covers.openlibrary.org/b/isbn/{isbn}-L.jpg" # Large size

response = requests.head(url) # Use HEAD request to check if image exists

if response.status\_code == 200: # 200 OK means the image exists

return url

else:

return None # Image not found on Open Library

except requests.exceptions.RequestException:

return None

def get\_cover\_from\_google\_books(title):

"""Retrieves a cover image URL from Google Books based on title."""

try:

quoted\_title = urllib.parse.quote(title) # URL-encode the title

url = f"https://www.googleapis.com/books/v1/volumes?q=intitle:{quoted\_title}"

response = requests.get(url)

response.raise\_for\_status() # Check for HTTP errors

data = response.json()

if data.get("totalItems", 0) > 0:

image\_links = data["items"][0]["volumeInfo"].get("imageLinks")

if image\_links and "thumbnail" in image\_links:

return image\_links["thumbnail"] # Return thumbnail

return None

except requests.exceptions.RequestException:

return None

except (KeyError, IndexError, TypeError): # Handle cases where the JSON structure is not as expected

return None

def slugify(text):

"""Generates a URL-friendly slug from text."""

text = text.lower()

text = ''.join(c for c in text if c.isalnum() or c == ' ')

return text.replace(' ', '\_') # Replace spaces with underscores

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487). Python

IGNORE\_WHEN\_COPYING\_END

**tag\_management.py:**

# modules/tag\_management.py

import nltk # pip install nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

import re

nltk.download('punkt', quiet=True) # Download tokenizer data if not already present

nltk.download('stopwords', quiet=True) # Download stopwords

def manage\_tags(data, num\_keywords=5):

"""

Manages tags by extracting keywords from descriptions and translating them.

Args:

data (list): A list of dictionaries with item data.

num\_keywords (int): The number of keywords to extract.

Returns:

list: A list of dictionaries with updated tags.

"""

stop\_words = set(stopwords.words('english') + stopwords.words('spanish')) # Combined stopwords

for row in data:

description\_en = row.get('Description (EN)', '')

description\_es = row.get('Description (ES)', '')

# Combine descriptions for more comprehensive tag extraction

combined\_description = f"{description\_en} {description\_es}"

keywords = extract\_keywords(combined\_description, stop\_words, num\_keywords) # Keywords in main language

row['Tags (EN)'] = row.get('Tags (EN)', '') + ', ' + ', '.join(keywords) if keywords else row.get('Tags (EN)', '')

# Attempt to translate keywords to Spanish if 'Tags (ES)' is empty (or vice versa)

if not row.get('Tags (ES)', '') and keywords:

from .translation\_tools import translate\_data # Import locally to avoid circular import

# Create a temporary dictionary for translation

temp\_data = [{'Tags (EN)': ', '.join(keywords)}] # Create a temporary structure for translation

translated\_temp\_data = translate\_data(temp\_data, target\_language="es")

row['Tags (ES)'] = row.get('Tags (ES)', '') + ', ' + translated\_temp\_data[0].get('Tags (ES)', '') # Append to existing tags

return data

def extract\_keywords(text, stop\_words, num\_keywords=5):

"""

Extracts keywords from text using NLTK.

Args:

text (str): The text to extract keywords from.

stop\_words (set): A set of stopwords to exclude.

num\_keywords (int): The number of keywords to extract.

Returns:

list: A list of keywords.

"""

text = re.sub(r'[^\w\s]', '', text) # Remove punctuation

word\_tokens = word\_tokenize(text.lower()) # Tokenize and lowercase

filtered\_words = [w for w in word\_tokens if w not in stop\_words and len(w) > 2] # Remove short words

# Basic frequency analysis for keywords (can be improved with more sophisticated methods)

word\_counts = {}

for word in filtered\_words:

word\_counts[word] = word\_counts.get(word, 0) + 1

sorted\_words = sorted(word\_counts.items(), key=lambda item: item[1], reverse=True)

return [word for word, count in sorted\_words[:num\_keywords]] # Return top N keywords

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487). Python

IGNORE\_WHEN\_COPYING\_END

**To use this code:**

1. **Install Libraries:**
2. pip install requests googletrans==4.0.0-rc1 Pillow nltk

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487). Bash

IGNORE\_WHEN\_COPYING\_END

1. **Create Folder Structure:**
   * Create a main project directory (e.g., venezuelan\_archive).
   * Inside that, create a modules directory.
   * Place the Python files (venezuelan\_archive\_toolkit.py, data\_enrichment.py, translation\_tools.py, image\_retrieval.py, tag\_management.py) in their respective locations.
2. **Run the script:**
3. python venezuelan\_archive\_toolkit.py input.csv output.csv --language es

IGNORE\_WHEN\_COPYING\_START

content\_copy download

Use code [with caution](https://support.google.com/legal/answer/13505487). Bash

IGNORE\_WHEN\_COPYING\_END

* + Replace input.csv with the path to your WorldCat CSV.
  + Replace output.csv with the desired path for the processed CSV.
  + The --language argument is optional. Use --language es to translate to Spanish, or --language en to translate to English (if you have Spanish descriptions you want to translate to English). If omitted, the default language is English.

**Explanation and Key Improvements:**

* **Modular Design:** The code is broken down into modules for better organization and reusability. Each module handles a specific task (data enrichment, translation, etc.).
* **Error Handling:** Includes try-except blocks for handling potential errors during API calls (network issues, invalid JSON, etc.), image processing, and translation. This makes the script more robust. Error messages are printed to the console to help with debugging.
* **API Usage:**
  + Uses Open Library and Google Books APIs to enrich metadata based on ISBN and/or title/author.
  + Uses Google Translate API for translations (remember there are usage limits).
  + Uses Google Books API for cover image retrieval if the ISBN method fails.
* **Image Handling:**
  + Downloads cover images and saves them to an images folder.
  + Includes a slugify function to create safe filenames from titles.
  + Saves the image file path in the "Files (if available)" column of the output CSV.
  + Checks if the image already exists to avoid re-downloading.
* **Tag Management:**
  + Uses NLTK for keyword extraction from descriptions.
  + Removes stop words (common words) to improve keyword relevance.
  + Translates extracted keywords to the target language.
* **Command-line Arguments:** Uses argparse to allow running the script from the command line with input and output file paths as arguments. This makes it easier to use in a workflow.
* **Rate Limiting for Translation:** Added time.sleep(1) to the translation loop to avoid overwhelming the Google Translate API.
* **ISBN Extraction:** Improved ISBN extraction by stripping whitespace and "ISBN: " prefixes.
* **Safeguards for Empty Fields:** The code now handles cases where fields like "Description" or "Title" are empty, preventing errors.
* **Open Library Author Handling:** The Open Library API response structure for authors can be complex. The code now handles this more robustly by fetching author details individually if necessary.
* **Image Retrieval Improvements:** The image retrieval logic is more robust, trying different sources (Open Library, Google Books) and handling potential errors gracefully.
* **Local Imports:** The translation\_tools module is imported locally within the tag\_management module to avoid a circular import error.
* **Clearer Output and Logging:** Added more print statements to give feedback on the script's progress and any errors encountered.

**Further Improvements and Considerations:**

* **API Keys:** For Google Books, consider using an API key for higher rate limits (though it's often not strictly required for basic usage).
* **Configuration File:** Store API keys, preferred languages, and other settings in a configuration file for easier management.
* **More Sophisticated Tagging:** Explore more advanced techniques for tag extraction, such as TF-IDF or topic modeling.
* **Subject Heading Standardization:** Map extracted keywords to standardized subject headings (e.g., Library of Congress Subject Headings) using a controlled vocabulary. You could use a library like skosify for this if you have a SKOS vocabulary file.
* **Database Integration:** For larger archives, consider using a database (e.g., PostgreSQL) to store the data instead of CSV files. This will improve performance and allow for more complex queries.
* **Omeka Classic API:** Automate the upload to Omeka Classic using the Omeka Classic API. You'll need to install the omeka-api Python library.
* **User Interface:** If you want to make this tool more user-friendly, consider building a simple graphical user interface (GUI) using a library like Tkinter or PyQt.
* **Translation API Choice:** Google Translate is convenient, but consider using a paid translation API (e.g., DeepL) for better quality and reliability, especially for large volumes of text.
* **Language Detection:** You could add language detection to automatically determine the language of descriptions and other fields before translation.

This is a solid starting point for automating your workflow. Remember to test the code thoroughly with your data and adapt it as needed. Good luck with your project!