UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

REPRESENTACIÓN AVANZADA DEL CONOCIMIENTO Y RAZONAMIENTO

S15 Tarea: Conversión de Datos Tabulares a RDF para Publicaciones Científicas

Integrantes:

- BERMEO CABRERA DIANA ROCIO

Docente:

Ing. Nelson Piedra.

LOJA

2025

Contenido

Resumen

Este proyecto aborda la necesidad crítica de transformar datos tabulares, comúnmente almacenados en archivos CSV, en un formato de grafo de conocimiento RDF (Resource Description Framework). El problema principal radica en la dificultad de convertir datos "planos" y a menudo desestructurados, especialmente aquellos con columnas multivaluadas, en un modelo semánticamente rico y conectado. El objetivo general es diseñar e implementar un algoritmo robusto que permita esta conversión, aplicando los principios de la Web Semántica y las buenas prácticas de Linked Data. La metodología de desarrollo se basa en un enfoque iterativo y modular, utilizando Python con librerías como Pandas para la manipulación de datos, Streamlit para la interfaz de usuario interactiva y RDFLib para la construcción del grafo. Los resultados incluyen una aplicación web intuitiva que guía al usuario a través de un proceso de dos pasos: limpieza y preparación del CSV, y mapeo dinámico de columnas a propiedades y entidades RDF, permitiendo la generación de grafos de conocimiento estructurados y descargables en formatos estándar como Turtle.

Abstract

This project addresses the critical need to transform tabular data, commonly stored in CSV files, into an RDF (Resource Description Framework) knowledge graph. The main challenge lies in the difficulty of converting "flat" and often unstructured data, especially those with multivalued columns, into a semantically rich and connected model. The general objective is to design and implement a robust algorithm that enables this conversion, applying Semantic Web principles and Linked Data best practices. The development methodology is based on an iterative and modular approach, utilizing Python with libraries such as Pandas for data manipulation, Streamlit for the interactive user interface, and RDFLib for graph construction. The results include an intuitive web application that guides the user through a two-step process: CSV cleaning and preparation, and dynamic mapping of columns to RDF properties and entities, allowing the generation of structured knowledge graphs downloadable in standard formats like Turtle.

Capítulo 1: Introducción

1.1. Planteamiento del problema

En la era actual de la información, grandes volúmenes de datos se generan y almacenan en formatos tabulares simples, como los archivos CSV (Comma Separated Values). Si bien los CSV son eficientes para el almacenamiento y el intercambio básico de datos, carecen de la capacidad inherente para representar relaciones complejas, la semántica subyacente de los datos y la interconexión entre diferentes conjuntos de datos. Esta limitación impide la creación de "conocimiento conectado" que es fundamental para aplicaciones avanzadas como la inteligencia artificial, los sistemas de recomendación, la búsqueda semántica y la integración de datos heterogéneos.

El desafío se agrava con la presencia de "columnas multivaluadas", donde una única celda contiene múltiples valores (ej., varios autores en una columna separados por un punto y coma). Procesar estos datos para extraer entidades individuales y sus relaciones de manera automática y consistente es una tarea compleja que a menudo requiere intervención manual o scripts ad-hoc, lo que es propenso a errores y poco escalable.

1.2. Justificación del proyecto

La transformación de datos tabulares a grafos de conocimiento RDF es crucial para desbloquear el verdadero potencial de los datos. Al convertir datos "planos" en un modelo de triples (sujeto-predicado-objeto), se les dota de significado explícito y se facilita su interconexión con otros conjuntos de datos en la Web Semántica. Esto permite:

- Integración de Datos: Combinar información de diversas fuentes de manera coherente.
- Consultas Semánticas: Realizar consultas más inteligentes que entienden el significado de los datos, no solo su estructura.
- Reutilización de Vocabularios: Aprovechar ontologías y vocabularios estándar (como Schema.org, Dublin Core, FOAF) para describir los datos de forma interoperable.

- Automatización: Reducir la necesidad de intervención manual en la preparación de datos para análisis avanzados.

Este proyecto se justifica por la necesidad de una herramienta accesible y flexible que simplifique este proceso de transformación, permitiendo a usuarios no expertos en programación generar grafos de conocimiento a partir de sus CSVs, superando el reto de las columnas multivaluadas y promoviendo la adopción de principios de Linked Data.

1.3. Objetivos del trabajo

1.3.1. Objetivo general

Diseñar e implementar una aplicación interactiva que permita convertir datos en formato tabular (CSV) a un modelo RDF (Resource Description Framework), aplicando principios de la Web Semántica y buenas prácticas de modelado de datos enlazados (Linked Data), con un enfoque particular en el manejo dinámico de columnas multivaluadas.

1.3.2. Objetivos específicos

- Desarrollar una interfaz de usuario intuitiva que guíe al usuario a través de las fases de carga, limpieza y mapeo de datos CSV.
- Implementar funcionalidades de preprocesamiento de CSV, incluyendo la configuración de delimitadores, el renombramiento de columnas y el manejo heurístico de valores nulos.
- Crear un módulo de mapeo dinámico que permita al usuario definir la clase de la entidad principal y especificar cómo cada columna del CSV se transforma en propiedades literales o relaciones a entidades relacionadas.
- Integrar la capacidad de identificar y dividir columnas multivaluadas, procesando cada sub-valor como un elemento independiente para la construcción del grafo.
- Asegurar la deduplicación de entidades relacionadas (ej., autores, organizaciones) para evitar la creación de nodos duplicados en el grafo.
- Permitir la descarga del grafo de conocimiento resultante en formatos estándar de RDF (ej., Turtle, RDF/XML).

1.4. Alcance y limitaciones

Alcance:

- Transformación CSV a RDF: La aplicación se enfoca exclusivamente en la conversión de datos desde archivos CSV a grafos RDF.
- Interfaz de Usuario Interactiva: Proporciona una interfaz web guiada para la configuración del mapeo.
- Manejo de Columnas Multivaluadas: Permite la especificación de delimitadores para columnas con múltiples valores.
- Deduplicación de Entidades: Realiza la deduplicación de entidades relacionadas (sujetos de propiedades de objeto) para construir un grafo más coherente.
- Uso de Vocabularios Estándar y Personalizados: Permite al usuario especificar URIs de propiedades y clases de vocabularios existentes (ej., Dublin Core, Schema.org, FOAF) y de un namespace personalizado.
- Salida Estándar RDF: Genera el grafo en formatos RDF estándar (Turtle, RDF/XML) para su posterior uso.

Limitaciones:

- No Persistencia de Datos: La aplicación procesa los datos en memoria y no almacena los CSVs ni los grafos generados de forma persistente en una base de datos.
- No Visualización del Grafo: La aplicación no incluye una herramienta integrada para visualizar el grafo RDF generado. Se requiere software externo (ej., Protégé, GraphDB, Gephi) para explorar el grafo.
- No Enriquecimiento de Datos Externos: No se conecta a fuentes de datos externas (ej., APIs de Wikidata, DBpedia) para enriquecer automáticamente las entidades.
- Validación de URIs/Vocabularios: No realiza una validación exhaustiva de la validez semántica o sintáctica de las URIs introducidas por el usuario (más allá de la verificación de cadena vacía).
- Complejidad de Mapeo Avanzado: Aunque es flexible, no soporta reglas de mapeo condicionales complejas basadas en lógica programática (ej., "si el valor es X, entonces mapear con la propiedad Y").

1.5. Metodología de desarrollo

El proyecto se desarrolló siguiendo una metodología ágil e iterativa, con un enfoque en la construcción incremental de funcionalidades y la retroalimentación continua.

- Planificación Inicial: Definición de objetivos generales y requisitos de alto nivel.
- Diseño Modular: Separación de la lógica en módulos (app.py, limpiar_csv.py, convertir_a_rdf.py) para facilitar el desarrollo y la mantenibilidad.
- Iteraciones Cortas: Cada iteración se centró en implementar un conjunto específico de funcionalidades (ej., carga de CSV, renombramiento, mapeo básico, mapeo multivaluado, deduplicación).
- Desarrollo Dirigido por la Interfaz: La interfaz de usuario de Streamlit se construyó de manera progresiva, guiando la implementación de la lógica de backend necesaria.
- Pruebas Continuas: Pruebas manuales y funcionales se realizaron en cada etapa para asegurar la correcta transformación de los datos y la usabilidad de la interfaz.

Capítulo 2: Marco Teórico

2.1. Fundamentos del problema o dominio

El dominio principal de este proyecto es la Gestión del Conocimiento y la Web Semántica. Se busca transformar datos estructurados de forma simple (CSV) en un formato que permita su interpretación y procesamiento por máquinas, construyendo así un "grafo de conocimiento".

Grafos de Conocimiento: Son bases de datos orientadas a grafos que representan información como una red de entidades (nodos) y sus relaciones (aristas). A diferencia de las bases de datos relacionales, los grafos de conocimiento se centran en la interconexión y el significado de los datos, permitiendo consultas más complejas y el descubrimiento de patrones ocultos.

Web Semántica: Una extensión de la World Wide Web que permite a los datos ser leídos y procesados por máquinas. Se basa en estándares como RDF, OWL y SPARQL para añadir significado (semántica) a la información en la web, facilitando la integración y el razonamiento automatizado.

Linked Data (Datos Enlazados): Un conjunto de principios para publicar datos estructurados en la web de manera que puedan ser interconectados y accesibles mediante URIs. Es un componente clave de la Web Semántica, promoviendo la creación de una "web de datos".

2.2. Tecnologías utilizadas

Python: Lenguaje de programación principal. Elegido por su versatilidad, su amplio ecosistema de librerías para ciencia de datos y desarrollo web, y su legibilidad.

Streamlit: Framework de Python para la creación rápida de aplicaciones web interactivas y paneles de control. Permite transformar scripts de Python en aplicaciones web con una mínima configuración de frontend. Es ideal para prototipado y herramientas internas.

Pandas: Librería de Python fundamental para la manipulación y análisis de datos. Proporciona estructuras de datos como DataFrames, que son eficientes para trabajar con datos tabulares y realizar operaciones de limpieza y transformación.

RDFLib: Librería de Python para trabajar con RDF. Permite crear, parsear, serializar y consultar grafos RDF. Es esencial para la construcción del modelo de conocimiento.

2.3. Estudios o trabajos similares

Existen varias herramientas y enfoques para la conversión de datos tabulares a RDF, que van desde soluciones programáticas hasta herramientas visuales.

R2RML / RML: Lenguajes de mapeo estándar (W3C) para transformar datos de bases de datos relacionales (R2RML) o cualquier fuente estructurada (RML) a RDF. Son muy potentes pero requieren un conocimiento técnico considerable para definir los mapeos.

OpenRefine (con extensiones RDF): Una herramienta popular para la limpieza y transformación de datos que puede extenderse para generar RDF. Ofrece una interfaz de usuario, pero el mapeo a RDF puede ser menos flexible o requerir configuraciones adicionales.

Custom Scripts: Muchos proyectos utilizan scripts personalizados en Python, Java u otros lenguajes para realizar conversiones CSV a RDF, adaptados a sus

necesidades específicas. Este proyecto se inspira en la necesidad de automatizar y hacer más accesible este proceso de scripting.

Este proyecto se diferencia al ofrecer una interfaz Streamlit interactiva y simplificada, con heurísticas de auto-sugerencia y un enfoque directo en el manejo de columnas multivaluadas, buscando un equilibrio entre flexibilidad y facilidad de uso sin la complejidad de RML.

2.4. Conceptos clave

RDF (Resource Description Framework): Un marco estándar del W3C para representar información sobre recursos en la Web. Se basa en triples de la forma (Sujeto, Predicado, Objeto), donde cada componente es una URI o un literal.

Triple: La unidad fundamental de información en RDF.

Sujeto: El recurso sobre el que se hace una afirmación (una URI).

Predicado: La propiedad o relación que describe el Sujeto (una URI).

Objeto: El valor de la propiedad, que puede ser un Literal o la URI de otro Recurso.

URI (Uniform Resource Identifier): Una cadena de caracteres que identifica un recurso de forma única en la Web. En RDF, las URIs se utilizan para identificar sujetos, predicados y objetos (si son recursos).

Literal: Un valor de datos, como una cadena de texto, un número, una fecha o un booleano. Los literales pueden tener un tipo de dato XSD (ej., xsd:string, xsd:integer) y/o una etiqueta de idioma (ej., @es).

Namespace (Espacio de Nombres): Un mecanismo para agrupar URIs relacionadas bajo un prefijo común, haciendo que los grafos RDF sean más legibles y concisos (ej., foaf:Person en lugar de http://xmlns.com/foaf/0.1/Person).

Propiedad de Objeto (Object Property): Un tipo de predicado en RDF que relaciona un Sujeto con otro Objeto que también es un Recurso (URI). (ej., dct:creator relaciona un Artículo con una Persona).

Propiedad de Datos (Data Property): Un tipo de predicado en RDF que relaciona un Sujeto con un Literal. (ej., schema:name relaciona una Persona con un string "Juan Pérez").

Deduplicación de Entidades: El proceso de asegurar que cada entidad del mundo real (ej., una persona, una organización) se represente con una única URI en el grafo, incluso si aparece varias veces en los datos de origen. Esto es crucial para la calidad del grafo.

Columnas Multivaluadas: Columnas en un archivo tabular donde una sola celda contiene múltiples valores lógicamente distintos, generalmente separados por un delimitador (ej., "valor1;valor2;valor3").

Capítulo 3: Análisis del Sistema

3.1. Requisitos funcionales y no funcionales

Requisitos Funcionales (RF):

RF1: Carga de Archivo CSV: El sistema debe permitir al usuario cargar un archivo CSV desde su sistema local.

RF2: Configuración de Delimitador CSV: El usuario debe poder especificar el delimitador principal del archivo CSV.

RF3: Previsualización de Datos: El sistema debe mostrar una previsualización de las primeras filas del CSV cargado.

RF4: Renombramiento de Columnas: El usuario debe poder renombrar las columnas del CSV antes del mapeo RDF.

RF5: Configuración de Columnas Multivaluadas: El usuario debe poder identificar columnas multivaluadas y especificar su delimitador interno.

RF6: Definición de Entidad Principal: El usuario debe poder definir la URI de la clase para la entidad principal que representará cada fila del CSV.

RF7: Selección de ID de Entidad Principal: El usuario debe poder seleccionar una columna del CSV para usar como identificador único de la entidad principal.

RF8: Mapeo de Propiedades RDF: Para cada columna, el usuario debe poder definir:

- Si la columna se mapea o no.
- La URI de la propiedad RDF (predicado).
- El tipo de mapeo (literal o entidad relacionada).

RF9: Configuración de Literales: Si el mapeo es literal, el usuario debe poder seleccionar el tipo de dato XSD.

RF10: Configuración de Entidades Relacionadas: Si el mapeo es a una entidad relacionada, el usuario debe poder definir:

- La URI de la clase de la entidad relacionada.
- Una columna opcional para el ID único de la entidad relacionada.

RF11: Generación de Grafo RDF: El sistema debe generar un grafo RDF basado en las configuraciones de mapeo.

RF12: Visualización de Grafo (Texto): El sistema debe mostrar el grafo RDF generado en formato Turtle en la interfaz.

RF13: Descarga de Grafo RDF: El usuario debe poder descargar el grafo RDF generado en formato Turtle.

RF14: Mensajes de Progreso y Error: El sistema debe proporcionar retroalimentación al usuario sobre el progreso de las operaciones y los errores encontrados.

Requisitos No Funcionales (RNF):

RNF1: Usabilidad: La interfaz de usuario debe ser intuitiva y fácil de usar para usuarios con conocimientos básicos de CSV y conceptos de RDF.

RNF2: Rendimiento: La aplicación debe ser capaz de procesar archivos CSV de tamaño moderado (cientos de miles de filas) en un tiempo razonable.

RNF3: Robustez: El sistema debe manejar gracefully errores de formato en el CSV y configuraciones de mapeo inválidas, mostrando mensajes claros.

RNF4: Escalabilidad (Limitada): La solución es adecuada para procesamiento en memoria; para volúmenes de datos muy grandes, se requerirían optimizaciones adicionales.

RNF5: Compatibilidad: La aplicación debe ser compatible con navegadores web modernos.

RNF6: Mantenibilidad: El código debe ser modular, bien comentado y fácil de entender y extender.

3.2. Diagramas

3.3.1. Diagrama de clases

Este diagrama representaría las clases principales del sistema y sus relaciones:

App (en app.py):

- **Atributos:** df_crudo, df_limpio_para_rdf, csv_columns, rename_map, multivalued_delimiters_final, column_rdf_mappings, main_entity_type_uri, main_entity_id_col. **(Estos se gestionan a través de st.session_state).**
- **Métodos**: run() (el flujo principal de Streamlit), add_multivalued_mapping().
- **Relaciones:**

  Usa LimpiadorCSV (la función limpiar_dataframe_generico).

  Usa ConvertidorRDF (la función convertir_dataframe_a_rdf).

LimpiadorCSV (en limpiar_csv.py):

- **Métodos:** limpiar_dataframe_generico(df).
- **Responsabilidades:** Manejo de nulos, inferencia y conversión de tipos.

ConvertidorRDF (en convertir_a_rdf.py):

- **Atributos:** g (rdflib.Graph), global_entity_uris_cache.
- **Métodos:** convertir_dataframe_a_rdf(...), clean_uri_segment(text).

- **Responsabilidades:** Creación de grafo, vinculación de namespaces, creación de entidades principales, mapeo de propiedades literales, mapeo de propiedades de objeto, deduplicación de entidades, manejo de multivaluadas.
- **Relaciones:**

Usa rdflib.Graph, rdflib.Literal, rdflib.URIRef, rdflib.Namespace.

Usa pandas.DataFrame.

3.3.2. Diagrama de Secuencia

Un diagrama de secuencia clave sería el de "Generar Grafo RDF":

Actor (Usuario): Inicia "Generar Grafo RDF" (st.button).

App:

- Recupera df_limpio_para_rdf, main_entity_type_uri, main_entity_id_col, final_column_rdf_mappings, multivalued_delimiters_for_rdf del st.session_state.
- Realiza Validación de Mapeos.
- Llama a convertir_dataframe_a_rdf(df_limpio_para_rdf, ..., final_column_rdf_mappings, ...) en ConvertidorRDF.

ConvertidorRDF:

- Inicializa rdflib.Graph.
- Vincula Namespaces.
- Itera por cada row en df_limpio_para_rdf.
- Crea main_entity_uri.
- Itera por col_name, mapping_config en final_column_rdf_mappings.
- Si mapping_type es "literal":
    - Crea literal_value.
    - Añade triple (main_entity_uri, prop_uri, literal_value).
    - (Si applies_to_entity fuera implementado: buscaría la URI de la entidad relacionada en current_row_entities_cache y la usaría como sujeto).
- Si mapping_type es "object_property":

    - Determina related_entity_uri (usando clean_uri_segment).
    - Verifica global_entity_uris_cache para deduplicación.
    - Si es nueva, añade triples (related_entity_uri, RDF.type, related_entity_type_ref) y (related_entity_uri, RDFS.label, Literal(value)).
    - Añade triple (main_entity_uri, prop_uri, related_entity_uri).
    - Almacena related_entity_uri en current_row_entities_cache.
- Retorna rdf_graph.

App:

- Serializa rdf_graph a Turtle.
- Muestra rdf_output (st.code).
- Prepara st.download_button.

Actor (Usuario): Descarga el archivo.

3.3. Modelo de datos

El sistema maneja dos modelos de datos principales:

Modelo de Datos de Entrada (CSV):

- Representado por un pandas.DataFrame.
- Es una estructura tabular con filas y columnas.
- Las columnas pueden ser de diversos tipos (texto, numérico, fecha) y pueden contener valores únicos o multivaluados (separados por un delimitador).
- No hay relaciones explícitas dentro del CSV más allá de la estructura de tabla.

Modelo de Datos de Salida (Grafo RDF):

- Representado por un objeto rdflib.Graph.
- Estructura basada en triples (sujeto, predicado, objeto).
- Sujetos: URIs que representan recursos (ej., un artículo, una persona, una organización).a
- Predicados: URIs que representan propiedades o relaciones (ej., dct:title, dct:creator, schema:name).

- Objetos: Pueden ser:
- Literales: Valores de datos con tipos XSD (ej., "Mi Título"^^xsd:string, "2023"^^xsd:gYear).
- URIs de Recursos: Enlaces a otros sujetos en el grafo, estableciendo relaciones (ej., dct:creator apunta a una URI de foaf:Person).
- Namespaces: Se utilizan para abreviar URIs y organizar el vocabulario (ej., drber:, schema:, foaf:).
- Deduplicación: Las entidades que se repiten en el CSV (ej., el mismo autor en diferentes artículos) se mapean a una única URI en el grafo, evitando redundancia y permitiendo una conectividad real.

Capítulo 4: Diseño del Sistema

4.1. Arquitectura del software

La aplicación sigue una arquitectura cliente-servidor ligera, típica de las aplicaciones Streamlit:

Cliente (Navegador Web): El usuario interactúa con la interfaz de usuario generada por Streamlit en su navegador. Envía entradas (carga de archivos, configuraciones) al servidor.

Servidor (Aplicación Streamlit en Python): Un proceso de Python que ejecuta el script app.py. Este proceso:

- Recibe las entradas del cliente.
- Maneja la lógica de la aplicación (lectura de CSV, limpieza, mapeo, generación de RDF).
- Utiliza las librerías Pandas y RDFLib para el procesamiento de datos y la construcción del grafo.
- Envía las actualizaciones de la interfaz de usuario (previsualizaciones, mensajes, grafo generado) de vuelta al cliente.
- Además, la arquitectura interna del código es modular, dividida en tres componentes lógicos principales:

Módulo de Interfaz de Usuario (app.py): Maneja la presentación y la interacción con el usuario.

Módulo de Limpieza de Datos (limpiar_csv.py): Encapsula la lógica de preprocesamiento del CSV.

Módulo de Conversión a RDF (convertir_a_rdf.py): Contiene la lógica central para la transformación a grafo.

Esta separación de responsabilidades mejora la mantenibilidad, la reusabilidad del código y facilita futuras extensiones.

4.2. Diseño de la base de datos

La aplicación no utiliza una base de datos persistente tradicional. El procesamiento de datos se realiza en memoria utilizando DataFrames de Pandas.

- pandas.DataFrame: Es la estructura de datos principal para manejar el CSV cargado y sus versiones limpias y renombradas.
- rdflib.Graph: El grafo RDF se construye y manipula en memoria.
- st.session_state: Streamlit utiliza un mecanismo de "estado de sesión" para persistir variables y configuraciones de la interfaz de usuario entre las interacciones del usuario. Esto permite que el DataFrame limpio, los mapeos de columnas y otras configuraciones se mantengan disponibles a medida que el usuario navega entre pestañas o interactúa con los widgets. No es una base de datos en el sentido tradicional, sino un mecanismo de almacenamiento temporal en el servidor para la sesión activa del usuario.

4.3. Diseño de la interfaz de usuario

El diseño de la interfaz de usuario se centra en la simplicidad y la guía paso a paso, facilitando el proceso de conversión a usuarios con diferentes niveles de experiencia.

Diseño de Pestañas: La aplicación se divide en dos pestañas principales ("Limpiar y Preparar CSV" y "Generar Grafo RDF") que representan las fases secuenciales del proceso. Esto ayuda a organizar la información y guiar al usuario.

Widgets Interactivos: Se utilizan componentes estándar de Streamlit:

- st.file_uploader: Para la carga de archivos.
- st.text_input: Para entradas de texto (delimitadores, URIs).

- st.selectbox: Para selecciones de columnas o tipos de datos.
- st.checkbox: Para opciones binarias (mapear/no mapear, multivaluado).
- st.radio: Para selecciones exclusivas (tipo de mapeo).
- st.data_editor: Para la edición interactiva de la tabla de renombramiento de columnas.
- st.button: Para disparar acciones.
- st.progress: Para mostrar el estado de las operaciones.

Mensajes de Retroalimentación: Se utilizan st.info, st.success, st.warning y st.error para proporcionar retroalimentación clara y contextual al usuario.

Ayuda Contextual: Los parámetros help en los widgets proporcionan información adicional al usuario sobre cómo usar cada control.

Organización Visual: El uso de st.subheader, st.markdown y st.columns ayuda a estructurar la información en la página y a alinear los widgets de forma lógica.

4.4. Selección de herramientas de desarrollo

Lenguaje de Programación: Python 3.13.5

Entorno de Desarrollo Integrado (IDE): Visual Studio Code, PyCharm, o cualquier otro IDE compatible con Python.

Control de Versiones: Git (se recomienda su uso para el seguimiento de cambios).

4.5. Seguridad, validaciones y control de errores

Validaciones de Entrada:

Delimitador CSV: Se espera que el usuario introduzca un delimitador válido. pd.read_csv maneja algunos errores de formato con on_bad_lines='skip'.

URIs de Propiedades/Clases: Se valida que las URIs de propiedades y clases de entidades relacionadas no estén vacías si la columna está marcada para mapear. Esto previene errores básicos de rdflib.

Selección de Columnas: Los st.selectbox aseguran que el usuario seleccione una columna existente.

Control de Errores (try-except):

La aplicación utiliza bloques try-except robustos en las operaciones críticas (lectura de CSV, generación de grafo RDF).

Si ocurre un error, se muestra un mensaje de error claro al usuario (st.error) y se imprime la traza completa de la excepción (st.exception) para facilitar la depuración por parte del desarrollador.

Procesamiento en Memoria: Al no almacenar datos de forma persistente ni interactuar con servicios externos, se reduce la superficie de ataque en términos de seguridad de datos. La seguridad se limita principalmente a la ejecución local de la aplicación.

Sanitización de URIs: La función clean_uri_segment en convertir_a_rdf.py es crucial para sanitizar los valores de las celdas antes de usarlos como parte de las URIs, eliminando caracteres no válidos y espacios para asegurar URIs bien formadas.

Capítulo 5: Implementación

5.1. Entorno de desarrollo

Sistema Operativo: Compatible con Windows, macOS, Linux.

Versión de Python: Python 3.7 o superior. Se recomienda Python 3.9 o 3.10 para compatibilidad con las últimas versiones de las librerías.

IDE: Visual Studio Code (con la extensión de Python).

Gestor de Paquetes: pip (integrado con Python).

5.2. Estructura de carpetas del proyecto

La estructura del proyecto es simple y modular:

/RDF7/

├── app.py

├── limpiar_csv.py

├── convertir_a_rdf.py

└── requirements.txt (opcional, para listar dependencias)

app.py: Contiene la lógica de la interfaz de usuario y orquesta las llamadas a los otros módulos.

limpiar_csv.py: Contiene la función para el preprocesamiento y limpieza del DataFrame.

convertir_a_rdf.py: Contiene la lógica para la transformación del DataFrame limpio a un grafo RDF.

5.3. Capturas de pantalla de la aplicación funcionando

Una vista general de la aplicación Streamlit en el navegador, mostrando el título, la descripción y las dos pestañas principales.

Muestra la pestaña "Limpiar y Preparar CSV" con el widget file_uploader, el campo para el delimitador y la tabla de previsualización del CSV.

Muestra la tabla interactiva st.data_editor donde el usuario puede cambiar los nombres de las columnas.

Muestra los selectores y campos de texto para configurar columnas multivaluadas, con la opción de añadir o eliminar configuraciones.

Una vista de la pestaña "Generar Grafo RDF", enfocada en la configuración de una columna individual, mostrando las opciones de URI de propiedad, tipo de mapeo (literal/entidad), tipo de dato XSD o clase de entidad relacionada, y la opción de multivaluado.

Muestra la parte inferior de la pestaña "Generar Grafo RDF" con el área de texto que contiene el grafo RDF en formato Turtle y el botón de descarga.

5.4. Pruebas realizadas

Pruebas Funcionales:

- Carga y Preprocesamiento: Se probaron CSVs con diferentes delimitadores, con valores nulos en varias posiciones, y con columnas que requerían renombramiento o eran multivaluadas para asegurar que la fase de limpieza funcionara correctamente.
- Mapeo RDF: Se probaron diversas configuraciones de mapeo:
- Mapeo de literales con diferentes tipos XSD (string, integer, date).
- Mapeo de entidades relacionadas (object properties) para autores, revistas, organizaciones, verificando la creación de nuevas entidades y la deduplicación.
- Mapeo de columnas multivaluadas a propiedades literales y de objeto, asegurando que todos los sub-valores se procesaran correctamente.

- Generación y Descarga: Se verificó que el grafo RDF se generara sin errores y que el archivo descargado fuera válido y contuviera los triples esperados.
- Pruebas de Robustez: Se introdujeron archivos CSV con formatos incorrectos o valores inesperados para verificar el manejo de errores de la aplicación.

5.5. Problemas encontrados y soluciones aplicadas

- Problema 1: Duplicación de Entidades Relacionadas:

Descripción: Inicialmente, al mapear columnas como "Autor" a entidades relacionadas, si el mismo autor aparecía en múltiples filas, se creaban múltiples URIs para la misma persona en el grafo, resultando en un grafo redundante y menos conectado.

Solución Aplicada: Se implementó un caché global (global_entity_uris_cache en convertir_a_rdf.py) para almacenar las URIs de las entidades relacionadas ya creadas. Antes de crear una nueva entidad, el sistema verifica si ya existe en el caché. Si existe, reutiliza la URI existente; de lo contrario, crea una nueva y la añade al caché.

- Problema 2: Complejidad en el Mapeo de Propiedades a Entidades Secundarias:

Descripción: La necesidad de adjuntar propiedades literales a entidades relacionadas (ej., el país de un autor, no el país del artículo) era un requisito complejo de implementar en la interfaz y la lógica de mapeo.

Solución Aplicada (Parcial): Se diseñó e implementó el widget st.selectbox("¿Esta propiedad... aplica a qué entidad?:") en app.py para permitir al usuario especificar esta relación avanzada. Sin embargo, la lógica completa para procesar este campo en convertir_a_rdf.py y adjuntar el triple al sujeto correcto (la URI de la entidad relacionada) aún está pendiente de implementación. Actualmente, todas las propiedades literales se adjuntan a la entidad principal. Esta es la limitación más significativa del producto final en términos de flexibilidad de mapeo.

- Problema 3: Manejo de Nulos y Tipos de Datos Inconsistentes en CSVs:

Descripción: Los CSVs reales a menudo contienen celdas vacías, valores no numéricos en columnas numéricas, o formatos de fecha inconsistentes, lo que causaba errores al intentar la conversión a RDF.

Solución Aplicada: Se desarrolló el módulo limpiar_csv.py con la función limpiar_dataframe_generico para manejar heurísticamente estos problemas: eliminación de filas con nulos en IDs críticos, relleno inteligente de nulos (vacío para texto, cero para números), e inferencia y conversión robusta de tipos de datos.

Capítulo 6: Resultados y Evaluación

6.1. Comparación con requisitos iniciales

La aplicación ha logrado cumplir la mayoría de los requisitos funcionales y no funcionales establecidos:

Carga y Preprocesamiento de CSV (RF1-RF5): Completamente implementado y funcional, incluyendo la previsualización, el renombramiento y la configuración de multivaluadas.

Definición de Entidad Principal (RF6-RF7): Implementado, permitiendo al usuario definir la clase y el ID de la entidad principal.

Mapeo de Propiedades RDF (RF8-RF10): Implementado de manera robusta para propiedades literales y de objeto, con opciones para tipos XSD, clases de entidades relacionadas e IDs.

Generación y Descarga de Grafo RDF (RF12-RF14): Completamente funcional, generando el grafo en Turtle y permitiendo su descarga.

Mensajes de Progreso y Error (RF15): La aplicación proporciona retroalimentación clara al usuario.

Usabilidad (RNF1): La interfaz de pestañas y los mensajes guiados contribuyen a una buena experiencia de usuario.

Robustez (RNF3): El manejo de errores con try-except y la función de limpieza mejoran la robustez.

6.2. Evaluación del desempeño

Velocidad: Para archivos CSV de tamaño pequeño a mediano (hasta unas pocas decenas de miles de filas y decenas de columnas), la aplicación funciona de manera fluida y rápida, con la generación del grafo tomando solo unos segundos.

Carga de Memoria: El procesamiento se realiza en memoria. Para archivos CSV muy grandes (cientos de MB o GB), la aplicación podría consumir una cantidad significativa de RAM, lo que podría llevar a un rendimiento degradado o errores por falta de memoria en sistemas con recursos limitados.

Usabilidad: La interfaz guiada y las auto-sugerencias hacen que el proceso sea accesible incluso para usuarios sin experiencia profunda en RDF, reduciendo la curva de aprendizaje.

6.3. Beneficios del sistema desarrollado

Facilidad de Uso: Simplifica el complejo proceso de conversión de CSV a RDF a través de una interfaz gráfica intuitiva.

Flexibilidad de Mapeo: Permite una personalización detallada de cómo los datos se transforman en triples RDF, incluyendo la definición de vocabularios y relaciones.

Generación de Conocimiento Conectado: Facilita la creación de grafos de conocimiento que son más ricos semánticamente y mejor interconectados que los datos tabulares originales.

Deduplicación Automática: Evita la redundancia en el grafo al asegurar que las entidades únicas se representen una sola vez.

Manejo de Multivaluadas: Aborda un desafío común en los datos tabulares, permitiendo que múltiples valores en una celda se conviertan en elementos RDF individuales.

Reutilización: Los módulos de limpieza y conversión son genéricos y podrían ser reutilizados en otros proyectos de procesamiento de datos.

6.4. Limitaciones del producto final

Las limitaciones más destacadas del producto final, que también representan oportunidades de mejora, son:

Ausencia de Visualización de Grafo: No hay una representación visual del grafo dentro de la aplicación, lo que dificulta la verificación inmediata de la estructura generada.

No Conexión a Fuentes Externas: No permite enriquecer el grafo con datos de APIs o bases de conocimiento externas.

Mapeo Avanzado de Propiedades Literales (RF11): La capacidad de adjuntar propiedades literales a entidades relacionadas (no solo a la entidad principal) está presente en la UI, pero su implementación lógica en el backend aún no está completa.

Validación Semántica Limitada: La validación de las URIs y la coherencia semántica del mapeo recae en gran medida en el usuario.

Capítulo 7: Conclusiones y Recomendaciones

7.1. Conclusiones generales del proyecto

El proyecto ha logrado exitosamente el objetivo de diseñar e implementar una aplicación interactiva para la conversión de CSV a grafo de conocimiento RDF. La arquitectura modular, el uso de Streamlit para una interfaz de usuario amigable y la integración de Pandas y RDFLib han demostrado ser una combinación efectiva para abordar el desafío de transformar datos tabulares en un formato semánticamente rico. La aplicación proporciona una solución valiosa para usuarios que necesitan crear grafos de conocimiento a partir de sus datos, superando obstáculos comunes como las columnas multivaluadas y la deduplicación de entidades. Aunque existen áreas para futuras mejoras, el prototipo funcional valida la viabilidad del enfoque y su utilidad práctica.

7.2. Recomendaciones para trabajos futuros

Completar la Lógica de applies_to_entity: Priorizar la implementación de la lógica en convertir_a_rdf.py para que las propiedades literales puedan adjuntarse

correctamente a entidades relacionadas específicas, no solo a la entidad principal. Esto desbloqueará un nivel mucho más profundo de detalle en el grafo.

Integración de Visualización de Grafo: Explorar librerías de visualización de grafos (ej., pyvis, networkx con adaptadores, o integrar un visor JavaScript como vis.js o D3.js con un backend Flask/FastAPI para Streamlit) para permitir a los usuarios inspeccionar visualmente el grafo generado.

Soporte para Otros Formatos de Entrada: Ampliar la capacidad para leer otros formatos de datos estructurados (ej., JSON, XML, Excel) como fuentes de entrada.

Validación de Esquemas/Ontologías: Permitir al usuario cargar un archivo de ontología (OWL) o un esquema (SHACL) para validar el grafo generado contra un modelo predefinido.

Mapeo Basado en Reglas: Explorar la integración de un motor de mapeo basado en reglas (ej., RML) para permitir transformaciones más complejas y condicionales.

7.3. Posibilidades de mejora o ampliación

Enriquecimiento de Datos: Integrar funcionalidades para consultar APIs externas (ej., Wikidata, GeoNames) y enriquecer automáticamente las entidades del grafo con información adicional.

Persistencia de Mapeos: Permitir al usuario guardar y cargar configuraciones de mapeo para reutilizarlas en futuros archivos CSV con estructuras similares.

Serialización a Múltiples Formatos: Añadir opciones para descargar el grafo en otros formatos RDF populares como RDF/XML, JSON-LD, N-Triples.

Interfaz de Usuario Mejorada: Mejorar la experiencia de usuario con arrastrar y soltar más avanzado, feedback visual en tiempo real y manejo de grandes conjuntos de datos en la UI.

Manejo de Errores Semánticos: Implementar chequeos más avanzados para detectar posibles errores semánticos en el mapeo (ej., propiedades aplicadas a tipos de entidades incorrectos según una ontología).

Referencias bibliográficas

RDF 1.1 primer. (2014, 24 junio). https://www.w3.org/TR/rdf11-primer/

Dublin CoreTM Metadata Element Set, Version 1.1: Reference Description. (s. f.). DCMI. https://www.dublincore.org/specifications/dublin-core/dces/

Schema.org - schema.org. (s. f.). Schema.org. https://schema.org/

FOAF Vocabulary Specification. (s. f.). http://xmlns.com/foaf/spec/

Ontospy > http://purl.org/ontology/bibo/. (s. f.). http://purl.org/ontology/bibo/

SKOS Simple Knowledge Organization System Reference. (s. f.). https://www.w3.org/TR/skos-reference/

Streamlit Docs. (s. f.). https://docs.streamlit.io/

pandas documentation — pandas 2.3.1 documentation. (s. f.). https://pandas.pydata.org/docs/

https://www.easyrdf.org/converter

https://www.w3.org/RDF/Validator/

Anexos

Enlaces a repositorio de Git que contiene el código fuente completo de app.py, limpiar_csv.py, y convertir_a_rdf.py, grafos generados en .rdf/xml y .ttl y una visualización del grafo en .png

Enlace: https://github.com/drbermeo/TabuladorRDF.git