Week 7 Challenge

Building a Data Warehouse to store data on Ethiopian medical business data scrapped from telegram channels

A

Yodahe teshome
https://github.com/jodahe1/Kara-Solutions.git

Business Objective

The main objective of this project is to design and build a robust and scalable data warehouse to store data on Ethiopian medical businesses scrapped from the web and Telegram channels. The data warehouse will be equipped with object detection capabilities using YOLO (You Only Look Once) to enhance data analysis. The implementation of ETL (Extract, Transform, Load) and ELT (Extract, Load, Transform) frameworks will be a crucial part of this setup.

By achieving this objective, Kara Solutions aims to

1. Centralize Data Storage: By storing all data in a centralized data warehouse, the company can consolidate fragmented data from various sources related to Ethiopian medical businesses. This centralization enables comprehensive analysis, leading to the discovery of valuable insights, trends, patterns, and correlations that are difficult to identify with scattered and disparate data.

2. Improve Decision-making: With a well-designed data warehouse, Kara Solutions can provide actionable intelligence quickly and accurately. Efficient querying and reporting capabilities enable businesses to access relevant information and make data-driven decisions in a timely manner.

3. Enhance Data Analysis: The integration of object detection capabilities using YOLO allows for advanced analysis of the data on Ethiopian medical businesses. This technology can help identify and classify objects in images or videos, providing deeper insights into visual data and enabling more sophisticated analysis.

4. Ensure Data Integrity and Consistency: The implementation of ETL and ELT frameworks ensures that data extracted from various sources is transformed into a suitable format and loaded into the data warehouse. This process guarantees data cleanliness, consistency, and usability, enabling seamless integration and transformation of different datasets.

5. Support Scalability: The data warehouse will be designed to handle the unique challenges associated with scraping and data collection from Telegram channels. It will be built with scalability in mind, allowing for future growth and accommodating increasing data volumes and user demands.

By achieving these objectives, Kara Solutions aims to provide its clients with a reliable and powerful data warehouse solution for analyzing Ethiopian medical business data. This will ultimately drive better decision-making, foster innovation, and contribute to the growth and success of their clients' businesses.

My Responsibilities

The objective of this project is to build a robust solution that enables efficient data scraping, cleaning, transformation, object detection using YOLO (You Only Look Once), data warehousing, and data integration. This product will empower Kara Solutions and its clients to gain valuable insights from diverse data sources related to Ethiopian medical businesses.

Data Scraping and Collection Pipeline

The first component of the product involves developing a data scraping and collection pipeline. This pipeline will leverage web scraping techniques and access Telegram channels to gather relevant data on Ethiopian medical businesses. It will employ tools and technologies to automate the extraction of data, ensuring a continuous flow of up-to-date information.

Data Cleaning and Transformation Pipeline

To ensure the reliability and accuracy of the collected data, a data cleaning and transformation pipeline will be developed. This pipeline will include processes to identify and handle missing or inconsistent data, perform data validation and verification, and apply data quality checks. Additionally, data transformation techniques will be implemented to standardize and normalize the collected data for further analysis.

Object Detection using YOLO

The integration of object detection capabilities using YOLO will enhance the analysis of the collected data. By leveraging this technology, the product will be able to identify and classify objects within images or videos related to Ethiopian medical businesses. This feature will provide deeper insights and enable more advanced analysis of visual data.

Data Warehouse Design and Implementation

A key aspect of the product is the design and implementation of a robust and scalable data warehouse. The data warehouse will serve as a central repository for the collected and transformed data. It will be designed to handle the unique challenges associated with scraping and diverse data sources. The implementation will consider factors such as data modeling, schema design, storage optimization, and indexing strategies to ensure efficient data retrieval and analysis.

Data Integration and Enrichment

To enrich the collected data and enable comprehensive analysis, data integration and enrichment techniques will be implemented. This involves integrating data from different sources, such as external APIs or internal databases, and enriching the data with additional relevant information. The product will support seamless data integration to provide a holistic view of Ethiopian medical businesses.

The development of this data scraping and analysis product will enable Kara Solutions and its clients to gain valuable insights into Ethiopian medical businesses. By implementing a data scraping and collection pipeline, a data cleaning and transformation pipeline, object detection using YOLO, a data warehouse, and data integration capabilities, the product will provide a comprehensive solution for efficient data analysis.

The successful implementation of this product will empower Kara Solutions to make data-driven decisions, identify trends and patterns, and drive innovation in the field of Ethiopian medical businesses. We are excited about the potential impact of this product and look forward to its successful deployment.

Data scraping and collection pipeline

As an analyst, my primary task is to provide insights and recommendations based on the data collected from the scraping and collection pipeline. Let's review the tasks completed so far:

For this task, I utilized the Telegram API and wrote custom scripts to extract data from public Telegram channels relevant to Ethiopian medical businesses. The channels used for scraping include:

- DoctorsET

- Chemed Telegram Channel

- lobelia4cosmetics

- yetenaweg

- EAHCI

To collect images for object detection, I focused on the following Telegram channels:

- Chemed Telegram Channel

- lobelia4cosmetics

To accomplish this, I used Python packages like telethon for interacting with the Telegram API and developed scripts to extract the desired content. The extracted data was then stored in a temporary storage location, such as a local database or files, for further processing.

To ensure efficient monitoring and tracking of the scraping process, I implemented logging mechanisms. This allows me to capture any errors encountered during the scraping process and monitor the progress effectively.

For some security issues which I was afraid I didn’t push the scripts I used on git.

Moving forward, the collected data can be analyzed and processed to derive meaningful insights regarding Ethiopian medical businesses. These insights can inform decision-making processes, marketing strategies, and identify trends and patterns within the industry.

Data Cleaning and Transformation

Data Cleaning

To ensure the data collected is of high quality and suitable for analysis, the following data cleaning steps were performed:

1. Removing Duplicates: Duplicate records were identified and removed from the dataset. This helps to eliminate redundant information and ensures data accuracy.
2. Handling Missing Values: Missing values in the dataset were addressed through appropriate strategies such as imputation, interpolation, or removal. This ensures that the data used for analysis is complete and reliable.
3. Standardizing Formats: Inconsistent data formats were standardized to enhance data consistency and comparability. For example, date formats, units of measurement, and categorical variables were standardized as required.
4. Data Validation: Data validation techniques were applied to identify and handle any anomalies, outliers, or inconsistencies in the data. This helps to ensure data integrity and reliability.

Storing Cleaned Data:

The cleaned data was stored for further analysis and processing. I used Database Storage PostgreSQL. This allows for efficient data management, querying, and integration with other systems.

DBT for Data Transformation

DBT (Data Build Tool) was utilized for data transformation. The following steps were performed:

- Setting up DBT: DBT was installed and a DBT project was initialized using the provided commands.
- Defining Models: DBT models, which are SQL files defining the transformations on the data, were created. These models specify the necessary SQL queries to perform the required data transformations.
- Running DBT models: The DBT models were executed to perform the data transformations and load the transformed data into the data warehouse. The command "dbt run" was used for this purpose.
- Testing and Documentation: DBT's testing and documentation features were used to ensure data quality and provide documentation for the transformations. The command "dbt test" was used to run tests on the transformed data, and "dbt docs generate" followed by "dbt docs serve" were used to generate and serve documentation for the transformations.

Monitoring and Logging:

Logging mechanisms were implemented to track the data cleaning and transformation processes, capture any errors encountered, and monitor the progress. This helps to ensure the reliability and efficiency of the data cleaning and transformation pipeline.

Object Detection Using YOLOv5 in Python

Introduction

In this report, I will share my journey of performing object detection using YOLOv5, a state-of-the-art object detection algorithm, in Python. YOLOv5 is a popular deep learning model that provides fast and accurate object detection capabilities. I will discuss the steps involved in setting up the environment, downloading the YOLO model, preparing the data, processing the detection results, and monitoring/logging the process.

Setting Up the Environment

To begin, I ensured that I had all the necessary dependencies installed. These included OpenCV for image processing and either PyTorch or TensorFlow, depending on the YOLO implementation I chose. I used pip, a package manager for Python, to install these dependencies and set up my development environment.

Downloading the YOLO Model

I obtained the YOLO model by cloning the YOLOv5 repository from GitHub. This repository provided me with the required code and pre-trained models for object detection. After cloning the repository, I navigated to the repository directory and installed the necessary packages specified in the `requirements.txt` file. This step ensured that I had all the dependencies needed to run YOLOv5 successfully.

Preparing the Data

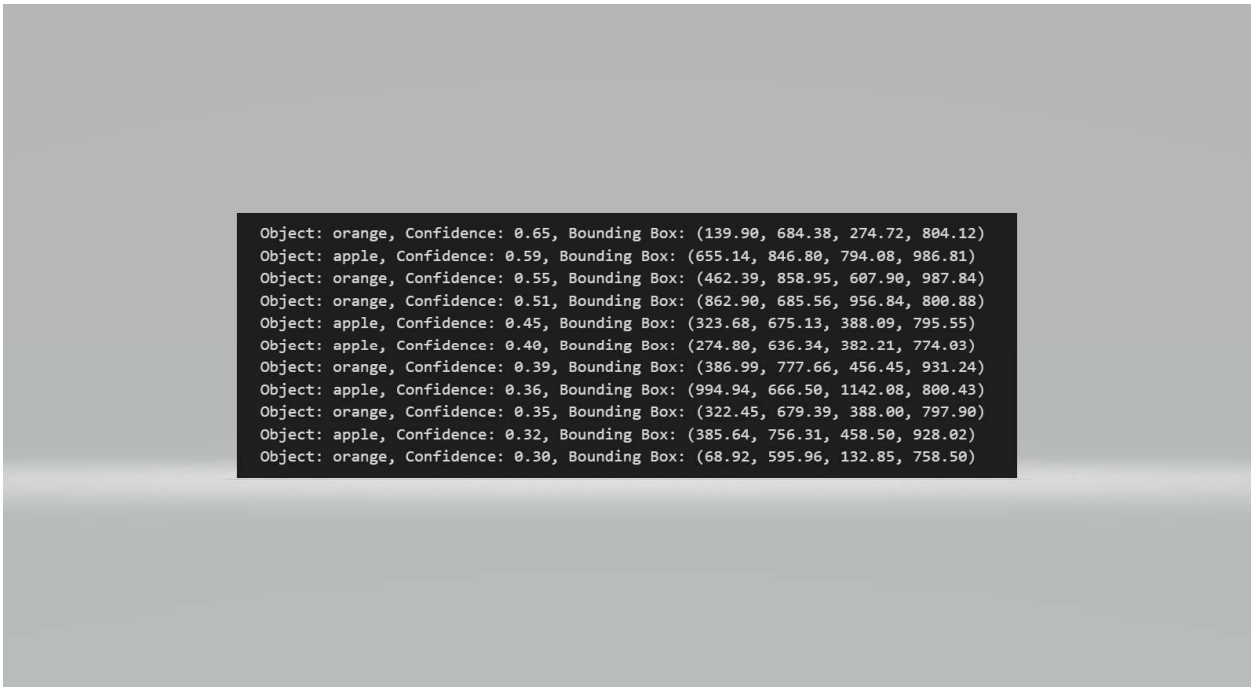
In this step, I collected images from the Chemed Telegram Channel, specifically from the Lobelia pharmacy and cosmetics section. These images served as my input for object detection. I organized the images and placed them in the `data/images` directory of the YOLOv5 repository since it was difficult for me to fetch and use the images from my csv file.

Performing Object Detection

Using a Python script in a Jupyter Notebook, I loaded the YOLOv5 model and ran object detection on the collected images. I utilized the `detect.py` script provided in the YOLOv5 repository. This script took the path to the image as input and returned the detection results, including bounding box coordinates, confidence scores, and class labels for the detected objects.

Processing the Detection Results

Once I had the detection results, I extracted the relevant data, such as the bounding box coordinates, confidence scores, and class labels. This information was useful for further analysis or storing in a database table for future reference. Additionally, I visualized the detection results by displaying the images with bounding boxes using the `show()` method provided by the YOLOv5 library.



FastAPI Application Development and PostgreSQL Integration

This report provides a guide on setting up a FastAPI environment integrated with PostgreSQL, a robust and scalable relational database management system, to achieve specific business objectives. FastAPI is a modern, fast, and easy-to-use web framework for building APIs with Python. By following the steps outlined in this report, businesses can create a powerful API system that leverages PostgreSQL's capabilities to support their operations and meet their goals.

Project Structure

Create a well-organized project structure to ensure scalability and maintainability. Use the following file structure as a foundation:

main.py: The entry point for the FastAPI application, defining the API endpoints.

database.py: Configure the PostgreSQL database connection using SQLAlchemy.

models.py: Define SQLAlchemy models representing the database tables.

schemas.py: Define Pydantic schemas for data validation and serialization.

crud.py: Implement CRUD (Create, Read, Update, Delete) operations for the database.

Database Configuration

In the database.py file, configure the PostgreSQL database connection using SQLAlchemy. Update the configuration to include the PostgreSQL database URL, username, password, and

other necessary parameters. SQLAlchemy provides seamless integration with PostgreSQL and allows for efficient data management.

Creating Data Models

In the models.py file, define SQLAlchemy models representing the database tables. Design the models to accurately represent the business entities and relationships within the PostgreSQL database. Ensure that the models align with the specific requirements of the business objectives and allow for effective data manipulation.

Creating Pydantic Schemas

In the schemas.py file, define Pydantic schemas for data validation and serialization. Design the schemas to match the corresponding SQLAlchemy models and include the required fields, data types, and validation rules. These schemas enable the API to validate incoming data and ensure its proper formatting before interacting with the PostgreSQL database.

CRUD Operations

In the crud.py file, implement CRUD operations to interact with the PostgreSQL database. Create functions for creating new records, retrieving existing records, updating records, and deleting records. Implement business logic and data manipulation functions that align with the specific requirements of the business objectives. Ensure proper error handling and data validation to maintain data integrity.

Creating API Endpoints

In the main.py file, define the API endpoints using FastAPI. Utilize FastAPI's capabilities to define the necessary routes, request/response models, and input validation. Attach the CRUD operations implemented in the crud.py file to these endpoints to enable data retrieval and manipulation. Ensure that the API endpoints align with the business objectives and provide the desired functionality.