Machine Reading New York State's Department of Environmental Conservation (NYS DEC) Construction+Demolition Reports and Mapping the Flow of Construction and Demolition Waste (CDW) Data Final Report

School name
NYU Tandon School of Engineering

Team member

Rui Xue	rx2161@nyu.edu
Tianyi Wu	tw2709@nyu.edu
Ruoan Ni	rn2429@nyu.edu
Yanfeng Xu	yx3104@nyu.edu

Acknowledgments
Christopher Policastro, Industry Assistant Professor
New York University

Terri C. Matthews, Director, Town+Gown
Department of Design and Construction

Contents

1. **Project Background**
   - 1.1 Introduction
   - 1.2 CDW Flow Mapping by CUSP 2021
   - 1.3 Machine Reading by MOT 2023 Spring
   - 1.4 Our Works and Contributions
2. **Methodologies and Tools**
   - 2.1 Documents
   - 2.2 Document AI
   - 2.3 Validation API
   - 2.4 Streamlit Framework
3. **Machine Reading**
   - 2.1 Auto-Extract Programs
   - 2.2 Auto-Validate Program
   - 2.3 Our Results and CSV Files
4. **CDW flow mapping**
   - 3.1 Dashboard
   - 3.2 Database
5. **Future Improvements**
   - 4.1 Correction and Evaluation
   - 4.2 Database Improvements
   - 4.3 Further Model Training
6. **Appendix A: A Simple Guide for GCP Tools**
7. **Appendix B: Document Analysis**

1. Project Background

1.1 Introduction

New York City is working on resource recycling through the reuse of materials collected from construction and demolition waste (C&DW). And get the material back indirectly through processing facilities and the manufacture of materials. This is part of the NYC Resource Recovery Task Force's Closed Loop City Planning Initiative (CLCPI).

This project report is about the Spring 2024 MOT capstone project. The goal of this project is to handle New York State Department of Environmental Conservation (NYS DEC) reports through machine reading and data mapping. This project is based on two previous projects, the Summer 2021 CUSP capstone project and the Spring 2023 MOT capstone project. The CUSP team focused on visualizing CDW flows based on data extracted from documents, while the MOT team developed a methodology to automate the data extraction process. This section provides a comprehensive overview of these prior projects, including the techniques and tools they used, and the progress that has been made. Understanding the foundation of our project is crucial for our work.

1.2 CDW FLow Mapping by CUSP 2021

Overview

The CUSP project emphasized on understanding the flow of Construction and Demolition Waste (CDW) and exploring the recycled materials market. More specifically, the team aimed to develop a tool for visualizing the movement and disposal of CDW, including recycling and landfill paths, and the types of materials involved, which could support policy makers' decision-making and their actions on

waste recycling. The data used for visualization was extracted from handwritten documents provided by the NYS DEP. The results of this work were presented in the form of a web application. In this web application based on the Streamlit framework, the team developed an interactive dashboard that visualizes the CDW flow in various chart views. This tool is useful in policymaking for CDW recycling and reuse, providing insights into the annual flow trends of CDW by material type, transactions, and destinations, and might foster a more sustainable approach to CDW management. Notably, although the team was successful in their work to visualize the data, they faced a lot of challenges to extract data. The CUSP team attempted to automate the data extraction process, but failed. It costs them plenty of time to manually convert data into machine readable format (totally 2543 rows of data in CSV format).

Detailed Analysis of Methodology

The CUSP team utilized Streamlit framework and various APIs to develop the web application. Streamlit is crucial in this project. It is an open source app framework to rapidly build interactive web applications with Python.

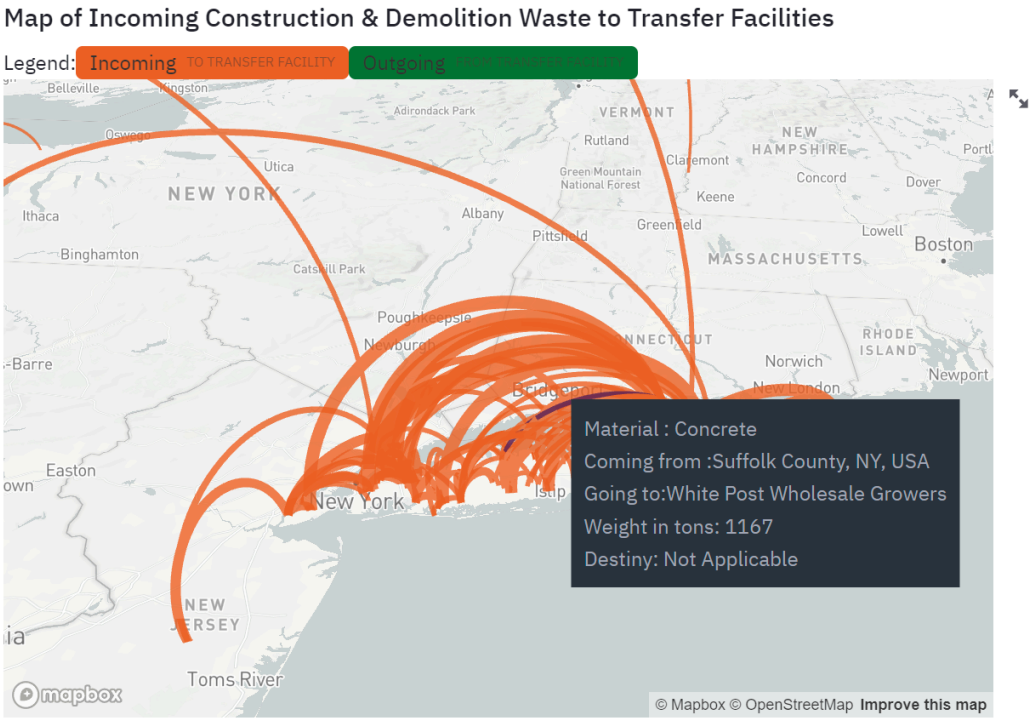


Figure 1: Map View in the Dashboard of CUSP's Web Application

The framework provides various components and web elements for developers to build an app. For visualization, It supports several charting libraries, including interactive charting libraries like PyDec/deck.gl, which they used for the 3D map in the web app. Besides, Google Maps API and Mapbox API provide support in the map content and its rendering.

1.3 Machine Reading by MOT 2023 Spring

Overview

The MOT team picked up from the work of the CUSP project. They expanded on the work, basically focusing on developing machine reading code to automate the data extraction process, which the CUSP team failed to implement. This team used Google Cloud Platform and Cloud Vision API to read data, and build a machine reading website, in which users can upload files and get the extracted data in CSV format. This team successfully developed an efficient and accurate machine reading tool, which can extract data from waste tracking documents. But due to time limits, this tool has not been integrated into the CUSP project. This is an area to be improved.

Detailed Analysis of Methodology

The team's works are presented within a Python program, which shows how the team utilized the Cloud Vision API and other libraries to automate the extraction of information from handwritten documents and convert them into CSV format files. They developed a machine reading system, using Google Cloud Platform and Cloud Vision API to automate data extraction from handwritten documents. Their Python program configures access credentials, converts PDF pages to images, crops images, performs OCR, and extracts specific information using regular expressions. However, the program lacks error handling, efficiency for large document sets, and accuracy assurance. Improvements such as error handling mechanisms, batch processing, and improved accuracy measures are necessary for practical application.

1.4 Our Works and Contributions

We can notice the relevance between the work of the two aforementioned teams: the MOT 2023 team's task is to extract key information from original documents and digitize this information; the CUSP team's task is to visualize the extracted information through a specific framework and tools, eventually presenting it in the form of interactive charts on a web application. The output of the MOT team can be considered as the input for the CUSP team. Thus, our task is to optimize their respective tasks on one hand, and on the other hand, to integrate their work and approaches to achieve a complete workflow, striving to realize a unified process integration from data extraction to data visualization.

After nearly three months of effort, even though there are deficiencies in accuracy, we have successfully automated machine reading. The multiple programs we developed form a system that can automatically extract key information from original Waste Tracking documents and correct and supplement this information. The extracted information can be directly used for the visualization of CDW Flows. Similar to the CUSP team, we use a web application to display our visualization results, which is an interactive map view.

The entire content of our work can be accessed via the following link:

Github Repository: https://github.com/NYU-Tandon-TMI/cdw

Web Application: https://nyc-ddc-cdw.streamlit.app/

Similarly, due to limited time and capabilities, there are many regrets in our work, and there are many areas that need improvement and enhancement. We will explain these at the end of this report.

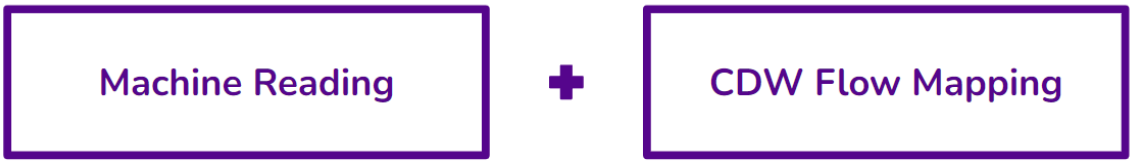


Figure 2: Integrating Machine Reading and CDW Flow Mapping

2. Methodologies and Tools

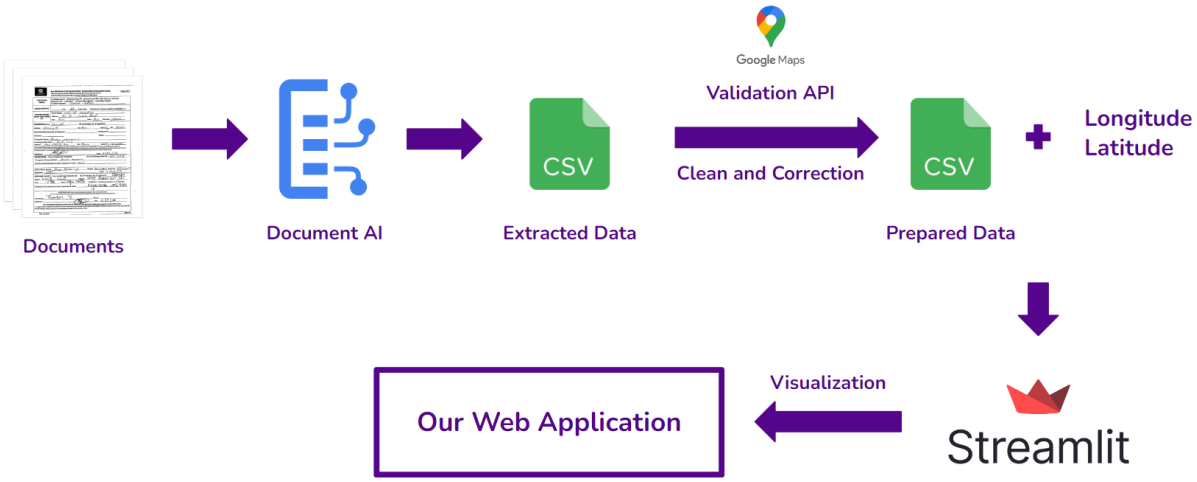
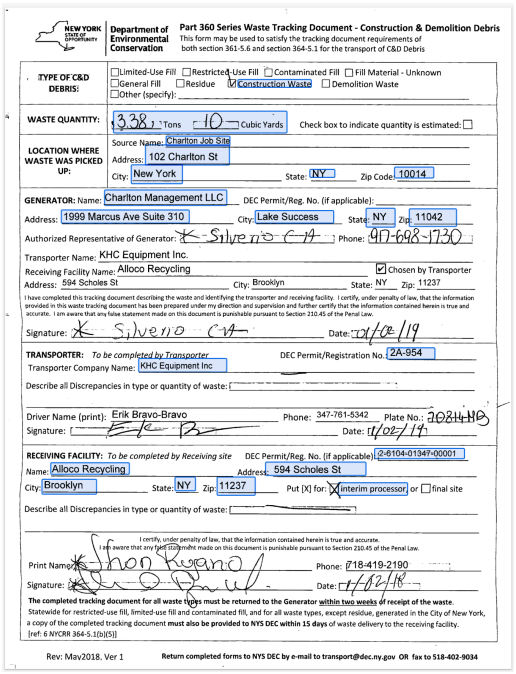


Figure 3: The Complete Workflow/Pipeline after Integration

2.1 Documents



NEW YORK STATE OF OPPORTUNITY. Department of Environmental Conservation

Part 360 Series Waste Tracking Document - Construction & Demolition Debris

This form may be used to satisfy the tracking document requirements of both section 361-5.6 and section 364-5.1 for the transport of C&D Debris

TYPE OF C&D DEBRIS: ☐Limited-Use Fill ☐Restricted-Use Fill ☐Contaminated Fill ☐Fill Material - Unknown ☐General Fill ☐Residue ☑Construction Waste ☐Demolition Waste ☐Other (specify):

WASTE QUANTITY: 3.38 Tons 10 Cubic Yards Check box to indicate quantity is estimated: ☐

LOCATION WHERE WASTE WAS PICKED UP: Source Name: Charlton Job Site Address: 102 Charlton St City: New York State: NY Zip Code: 10014

GENERATOR: Name: Charlton Management LLC DEC Permit/Reg. No. (if applicable): Address: 1999 Marcus Ave Suite 310 City: Lake Success State: NY Zip: 11042 Authorized Representative of Generator: K Silverio C/A Phone: 917-698-1730

Transporter Name: KHC Equipment Inc.

Receiving Facility Name: Alloco Recycling ☑Chosen by Transporter Address: 594 Scholes St City: Brooklyn State: NY Zip: 11237

I have completed this tracking document describing the waste and identifying the transporter and receiving facility. I certify, under penalty of law, that the information provided in this waste tracking document has been prepared under my direction and supervision and further certify that the information contained herein is true and accurate. I am aware that any false statement made on this document is punishable pursuant to Section 210.45 of the Penal Law.

Signature: K Silverio C/A Date: 11/02/19

TRANSPORTER: *To be completed by Transporter* DEC Permit/Registration No.: 2A-954 Transporter Company Name: KHC Equipment Inc

Describe all Discrepancies in type or quantity of waste:

Driver Name (print): Erik Bravo-Bravo Phone: 347-761-5342 Plate No.: 20814MB

Signature: Erik Bravo Date: 11/02/19

RECEIVING FACILITY: *To be completed by Receiving site* DEC Permit/Reg. No. (if applicable): 2-6104-01347-00001 Name: Alloco Recycling Address: 594 Scholes St City: Brooklyn State: NY Zip: 11237 Put [X] for: X interim processor or ☐final site

Describe all Discrepancies in type or quantity of waste:

I certify, under penalty of law, that the information contained herein is true and accurate. I am aware that any false statement made on this document is punishable pursuant to Section 210.45 of the Penal Law.

Print Name: Shon Kwang Phone: 718-419-2190

Signature: Shon D. Kwang Date: 11/02/19

The completed tracking document for all waste types must be returned to the Generator within two weeks of receipt of the waste. Statewide for restricted-use fill, limited-use fill and contaminated fill, and for all waste types, except residue, generated in the City of New York, a copy of the completed tracking document **must also be provided to NYS DEC within 15 days** of waste delivery to the receiving facility. [ref: 6 NYCRR 364-5.1(b)(5)]

Rev: May2018, Ver 1 **Return completed forms to NYS DEC by e-mail to transport@dec.ny.gov OR fax to 518-402-9034**

Figure 4: A Standard DEC Waste Tracking Document

As shown in the figure 4, this is a standard DEC waste tracking document, and the information we need to extract is highlighted in the blue box. In such a document, we focus on the type and quantity of waste, the pickup location, the entity that generated the waste, the transporter, and the facility that receives the waste. We extract this information using tools provided on Google Cloud Platform, specifically Document AI, and then write programs for further processing.

2.2 Document AI

Google's Document AI is an advanced AI solution designed to automate document processing, allowing organizations to extract and digitize data from various document types, such as invoices, receipts, and contracts. It reduces manual data entry, enhances data privacy, and improves document management efficiency with features like form parsing, data extraction, document classification, and entity recognition. By transforming static documents into actionable insights, Google's Document AI helps businesses streamline operations, reduce costs, and make informed decisions, boosting overall productivity.

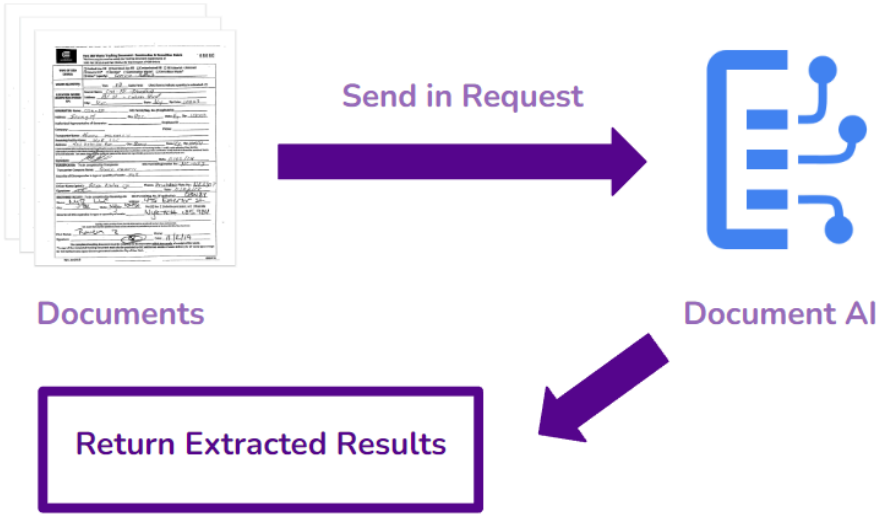


Figure 5: How Document AI Works

Document AI can be used to extract data from our documents. It supports a lot of formats, and uses generative AI to extract and structure data. Document AI has

high-accuracy to extract, classify, and split. Notably, integrated with generative AI, it can be trained to improve accuracy, which also means developers should spend some time on data labeling and model training based on the foundational models. We simply need to establish a connection with GCP, obtain the necessary permissions, and then we can send documents via a request to the client in a local or other environment to call the trained model for processing. After processing, Document AI will return the extracted results.

2.3 Validation API

The information extracted directly through Document AI includes many address details. However, due to various reasons such as errors in the document itself or inaccuracies in model recognition, there can be many issues with the extracted data, necessitating the need for verification and correction of these addresses. Additionally, if we aim to perform CDW flow mapping and visually represent the flow of CDW on a map, we require latitude and longitude information for the respective addresses.

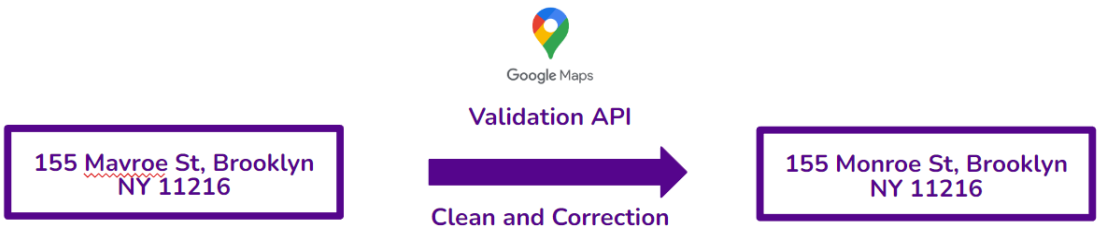


Figure 6: How Validation API Works

This aspect of the project is particularly challenging. After much comparison and decision-making, we ultimately chose to use the validation API provided by Google Maps Platform. Google Map Platform's Validation API is a tool designed to enhance the accuracy and reliability of location data within applications. It enables developers to validate and verify addresses and geographical coordinates provided by users, ensuring that the data corresponds accurately to real-world locations. This API is particularly useful for businesses that require precise location information,

such as logistics, delivery services, and location-based advertising. By integrating the Validation API, developers can reduce errors in location data, improve user experience by guiding correct address inputs, and enhance the efficiency of operational processes that depend on accurate geographical information.

2.4 Streamlit Framework

After processing the data, we visualize it by building a web application, similar to the work done by the CUSP team. We use the Streamlit framework to construct our web app. Streamlit provides an easy and fast way to build web applications and offers free community cloud resources.

Streamlit is an open-source framework designed for creating interactive web applications quickly and with minimal code, primarily for data science and machine learning projects. It allows developers to convert data scripts into shareable web apps by adding simple Streamlit commands to create widgets and visualizations, without needing front-end development skills. Key features include rapid prototyping, easy customization with interactive widgets, and straightforward deployment options. Streamlit's ability to streamline the app development process makes it highly appealing for data professionals who want to showcase their analytical results interactively and effectively. It allows applications to be hosted and run in the cloud, with the code stored in a GitHub repository.

2. Machine Reading

In this section, we will introduce our programs for Machine Reading part of our project, and share information about our results in the past three months. For further details, see these programs and files in: https://github.com/NYU-Tandon-TMI/cdw/tree/main/toolbox

2.1 Auto-Extract Programs

These programs, or more specifically, jupyter notebooks, are used to extract data from documents or document folders.

Document Extractor for One File

The original notebooks can be accessed here: https://github.com/NYU-Tandon-TMI/cdw/blob/main/toolbox/document_extractor_one_file.ipynb

The Jupyter notebook consists of three main code cells that together create a document processing system using Google's Document AI and PyMuPDF (Fitz) library. The first cell imports necessary libraries and sets up configuration parameters for using Google's Document AI. It initializes variables like project ID, location, processor ID, and file paths. This cell also specifies the type of input file (PDF) and the MIME type to be processed. It configures the client for Google's Document AI service and defines a list of columns that appear to be intended for storing extracted data from the documents. The second cell defines several functions to process data extracted from documents. `process_type` function uses the `difflib` library to match text to a list of predefined categories with a similarity threshold. `process_data` function cleans text data by replacing newline characters. `process_page` function processes each page of the PDF, converting it to an image, and using Google Document AI to extract entities which are then mapped to the predefined columns. Finally, `process_pdf` function reads a PDF document page-by-page, processes each page using `process_page`, and collects data into a

DataFrame which is then written to a CSV file. The third cell sets the path for the output CSV file and the input PDF file, then calls the `process_pdf` function to process the specified document and save the extracted data to the CSV file.

It takes 5 to 10 seconds to extract data from one file.

Document Extractor for One Folder

The original notebooks can be accessed here: https://github.com/NYU-Tandon-TMI/cdw/blob/main/toolbox/document_extractor_one_folder.ipynb

This notebook extends the capabilities of the previous one by adding functionality to process multiple PDF files within a folder. Just like the previous notebook, the first cell imports necessary libraries and configures settings for using Google's Document AI. It also defines the project, location, processor, and sets up the client with specific API endpoint configurations. This cell prepares the environment similarly by defining variables and initializing a client for the Document AI service. The second cell also includes function definitions similar to those in the previous notebook. Functions for processing text and images from pages are defined (`process_type`, `process_data`, `process_page`). A new function `process_pdf` is defined to open and process each page of a PDF, similarly converting pages to images, extracting data using Google's Document AI, and appending it to a DataFrame which is saved to a CSV. Additionally, there is a new function `process_folder` that iterates over all PDF files in a specified folder, applying the `process_pdf` function to each. The third cell sets paths for the output CSV file and the folder containing multiple PDF documents. It calls `process_folder` to process each PDF in the specified directory and write the extracted data to the CSV file. This part focuses on batch processing of documents, unlike the single document focus in the previous notebook.

	A	B	C	D	E	F	G	H	I
1	type_debris	waste_quantity	pickup_name	pickup_address	pickup_city	pickup_state	pickup_zip	pickup_lat	pickup_lng
2	Contaminated Fill	12 Cubic Yards	Sisters of Saint Joseph Convent	1725 Brentwood Road	Brentwood	NY	11717		
3	Sand Mud	4.2 Tons	DOS SI Blue West Building	600 West Service Road	Staten Island	NY	10314		
4	Sand Mud	12.89 Tons	DOS SI Blue West Building	600 West Service Road	Staten Island	NY	10314		

Figure 7: The Extracted Results in CSV

2.2 Auto-Validate Program

The original notebooks can be accessed here: https://github.com/NYU-Tandon-TMI/cdw/blob/main/toolbox/document_preprocess.ipynb

The notebook is structured to clean and validate our extracted data. The first two cells define functions to preprocess certain attributes of the data. The first function, `process_type_debris`, handles the 'type_debris' column by filling missing values with "Unknown" and replacing the term "Mix" with "Mix/Other". The second function, `process_waste_quantity`, cleans the 'waste_quantity' column by validating the format (e.g., correct units like Cubic Yards or Tons) and marking entries that do not match the expected pattern as "Unconfirmed", while also filling missing entries with "Unknown". The third cell is more complex and performs several operations.It loads data from a CSV file, applies the previously defined cleaning functions to the data, and defines a function, `validate_address`, to validate and geocode addresses using Google's Address Validation API. This function constructs a JSON payload to send to the API and processes the response to extract geocode data and a confidence status for each address. Another function, `update_address_data`, is defined to update address-related columns in the dataset based on the results from the `validate_address` function. It checks if geocode data already exists and if not, it validates the address and updates the dataset accordingly. The cell concludes by applying the `update_address_data` function to both pickup and receiving addresses in a dataset, updating the latitude, longitude, and geocode confidence status. It also handles missing values by filling them with "Unknown". Finally, the processed dataset is saved to a new CSV file, and a message is printed to indicate the completion of data export.

This program runs fast. It only takes 5 minutes to clean and validate 1000 rows of data extracted by auto-extract programs.

	A	B	C	D	E	F	G	H	I	J
1	type_debris	waste_quantity	pickup_name	pickup_address	pickup_city	pickup_state	pickup_zip	pickup_lat	pickup_lng	pickup_geocode_confidence
2	Contaminated Fill	12 Cubic Yards	Sisters of Saint Joseph Convent	1725 Brentwood Road	Brentwood	NY	11717	40.77358879	-73.24175059	confirmed
3	Sand Mud	4.2 Tons	DOS SI Blue West Building	600 West Service Road	Staten Island	NY	10314	40.5856714	-74.19314981	confirmed
4	Sand Mud	12.89 Tons	DOS SI Blue West Building	600 West Service Road	Staten Island	NY	10314	40.5856714	-74.19314981	confirmed
5	Sand Mud	12.88 Tons	DOS SI Blue West Building	600 West Service Road	Staten Island	NY	10314	40.5856714	-74.19314981	confirmed
6	Demolition Waste	Unknown	Asplundh Yard	294 Old Northport Rd	Kings Park	NY	11754	40.86997063	-73.26886954	confirmed

Figure 8: The Extracted Results after Cleaning and Validation

2.3 Our Results and CSV files

Our machine reading tasks are currently conducted in a Jupyter Notebook, and we've already discussed the technical aspects used in earlier slides. More technical details will be included in our project report. Up to this point, we have extracted data from 22,501 documents, with each data entry originating from a single page document.

Due to the limitations of the validation API, we have chosen to pause further extraction work. The validation API, or the process of address verification and correction, is not only a bottleneck in our work but will also present a significant challenge for future teams responsible for related tasks. We will discuss this further in the 'future improvement' section.

Also, for those who might be confused about why we have so many csv files, here is the explanation. Figure 9 shows how we process the documents now. First, we store the extracted data in the file “cdw_csv_original”. While the next step is not immediately clean and correct these data using Address Validation API. Due to the limitation of Validation API and the running speed (the program runs fast, and our money is spent quickly), we highly recommend that the original data should be processed manually first to improve the effectiveness.

In the cdw_csv_original.csv, there are 22499 rows of data. In the cdw_csv_processed_manully.csv, which serves as the “transfer station”, there are 7218 rows of data. In the cdw_csv_processed_auto.csv, there are 2000 rows of data, all information of which are complete and can be directly used in CDW Flow Mapping.

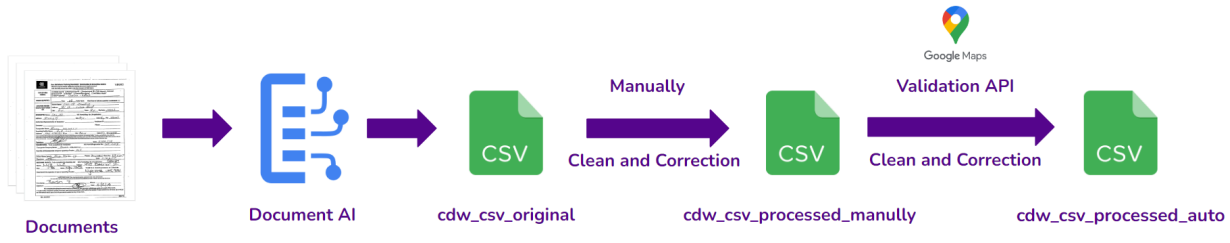


Figure 9: The Extracted Results after Cleaning and Validation

3. CDW Flow Mapping

The CDW Flow Mapping part of our project is presented as a web application developed using Streamlit and Pydeck to visualize the flow of construction and demolition waste (CDW). This program loads data from a CSV file hosted on GitHub, which includes information on fragment types, pick and receive addresses, and geographic coordinates. Users can filter the dataset based on the type of fragmentation and select specific pick-up and receive addresses. This interface also allows users to customize colors to visually distinguish pick-up and drop off routes. This interactive filtering and visualization helps deepen our understanding of waste management logistics.

Using Pydeck's ArcLayer, this application can visualize the route between the picking position and the receiving position. Each route is interactive, displaying tooltips that include detailed information such as fragment type, waste quantity, and address. This feature enhances the user experience by providing instant access to relevant data. The error handling mechanism is in place, ensuring smooth operation even if there is no data matching the selected filter.

The CDW Flow Mapping part not only facilitates effective CDW management, but also supports urban planning and decision-making processes by providing clear, data-driven insights. Future enhancements may include user data upload capabilities, finer grained filtering options, and performance optimization for processing large datasets. This application is a key tool for promoting sustainable urban development and waste management strategies.

3.1 Dashboard

Data Loading and Display

The CDW Flow Mapping part starts by importing the necessary libraries: streamlit for building web applications, pandas for data operations, and pydeck for geographic visualization. The code first defines the title of the application using "st.

title" ('DDC mapper '). The data is loaded from a CSV file hosted on GitHub, with the URL specified as "file_URL". The "pd.read_csv (file_url)" function reads the CSV file into DataFrame "df" and displays the data box to confirm successful loading of data using st.write ("Data loaded successfully!"). Next, the code generates a drop-down menu for filtering data. The unique values of fragment type, pick address, and receive address are extracted from DataFrame "df" using the "Unique()" function and stored in a list prefixed with "All". These lists use "st. selectbox()" to fill the drop-down menu, allowing users to choose the criteria for filtering data. In addition, a color selector is provided for customizing the colors of picking and receiving addresses on the map.

```
import streamlit as st
import pandas as pd
import pydeck as pdk

st.title('DDC Mapping Program')

file_url = 'https://raw.githubusercontent.com/TianyiWuNYU/test/main/data/cdw_csv_processed.csv'
df = pd.read_csv(file_url)

st.write("Data loaded successfully!")

unique_debris_types = ['All types of debris'] + list(df['type_debris'].unique())
selected_debris = st.selectbox('Select Type of Debris:', unique_debris_types)
```

Figure 10: Code Snippet

After initialization, the application reads data from the CSV file, ensuring that all relevant information about fragment types, picking and receiving locations is available for user interaction. This is crucial for the functionality of the application as it forms the foundation for all subsequent data operations and visualization. The filtering function is achieved by comparing user selections with the corresponding columns in the DataFrame. The filtered data is then used to visualize the route between the picking position and the receiving position. This process is dynamic, ensuring that any changes entered by the user will immediately update the displayed data. The visualization component is processed by the draw_routes function, which uses Pydeck to create an interactive map. This function generates a route between coordinates based on filtered data, customizes its appearance according to the color selected by the user, and displays other information through tool prompts. This interactive map provides users with a clear and detailed view of CDW traffic, which helps with effective data analysis and decision-making.

Data Filtering

The data filtering in the CDW Flow Mapping part is achieved by dynamically applying user input to filter the dataset, and then visualizing the results on the map for processing. The core filtering function is implemented using Pandas operations. The filtered dataset is created based on logical conditions determined by the user's selection of fragment type, pickup address, and receive address. The "filtered data" DataFrame is constructed by checking whether each row matches the selected criteria. These conditions are combined using the bitwise AND operator "&". For example, the code checks whether the "type_debris" column matches the selected fragment type, or whether "All types of fragments" is selected. Similar conditions apply to picking and receiving addresses. This ensures that "filtered_data" only contains rows that meet all specified conditions.

```
filtered_data = df[
    ((df['type_debris'] == selected_debris) | (selected_debris == 'All types of debris')) &
    ((df['pickup_address'] == selected_pickup_address) | (selected_pickup_address == 'All pickup addresses')) &
    ((df['receiving_address'] == selected_receiving_address) | (selected_receiving_address == 'All receiving addresses'))
]
```

Figure 11: Code Snippet

Filtering ability is crucial for users to narrow down their data to specific scenarios of interest. By allowing users to choose fragment types and addresses, they can focus on specific aspects of CDW streams. This makes the analysis more targeted and relevant, which helps with the decision-making process. After filtering, the "draw_routes'' function will use the filtered data to create an interactive map. This feature uses Pydeck's ArcLayer to visualize the route between picking and receiving positions. The route will be colored according to user selection, and interactive tooltips will provide additional information about each route, such as fragment type, waste quantity, and address. The map view is centered and scaled based on filtered geographic coordinates, ensuring that all relevant routes are clearly visible.

The dynamic features of filtering and visualization allow users to see real-time updates on the map when adjusting filters. This interactivity enhances the user experience by providing an intuitive understanding of CDW flows, making it easier to identify patterns and trends.

Color Customization

When visualizing CDW flows, we need to use different colors to differentiate the start and destination of each flow. Color customization is achieved through the "color_picker" widget of Streamlight. This allows users to choose colors for picking and receiving addresses. The selected color is stored in a variable and then converted from hexadecimal to RGBA format. This conversion is necessary because Pydeck uses RGBA values to render graphic elements. This program converts hexadecimal color codes to RGBA format, which includes adding an alpha channel for opacity. This ensures that the selected colors are correctly applied to map visualization.

```
pickup_color = st.color_picker('Choose a color for pickup addresses', '#FF6347')
receiving_color = st.color_picker('Choose a color for receiving addresses', '#4682B4')
```

Figure 12: Code Snippet

The color customization feature significantly enhances user interaction and the visual appeal of the application. By allowing users to choose different colors for pickup and delivery addresses, it is easier to visually distinguish different routes on the map. This customization not only makes data easier to access, but also helps to better understand and analyze the flow of construction and demolition waste.

Apply these user selected colors to the visualization, and the route will be displayed in a way that reflects user preferences. This personalized visualization helps to quickly identify patterns and anomalies, making the data more meaningful and easier to interpret.

Route Visualization with Pydeck

The "draw_routes" function is the core of the CDW Flow Mapping part's ability to visualize routes. This function utilizes Pydeck (a powerful library for deck. gl integration) to create interactive and informative maps. The function first checks whether the filtered dataset is non empty. If it contains data, it will build a routing list. Each route is a dictionary that contains source and destination coordinates, as well as other information about fragment types, quantities, and addresses.

```
routes = [
    {
        "from_coordinates": [row['pickup_lng'], row['pickup_lat']],
        "to_coordinates": [row['receiving_lng'], row['receiving_lat']],
        "info": f"Type of Debris: {row['type_debris']}<br>"
                f"Waste Quantity: {row['waste_quantity']}<br>"
                f"Pickup Name: {row['pickup_name']}<br>"
                f"Pickup Address: {row['pickup_address']}<br>"
                f"Receiving Name: {row['generator_name']}<br>"
                f"Receiving Address: {row['generator_address']}"
    }
    for _, row in filtered_data.iterrows()
]
```

Figure 13: Code Snippet

Then use Pydeck's ArcLayer to visualize the route, which aims to draw arcs between coordinate pairs. The source and target colors of arcs are customized based on user input, providing clear visual differences between different routes. This function sets various parameters for ArcLayer, such as the width, tilt, and color of the arc. It also includes a tool tip that displays detailed information for each pipeline when hovering above. Finally, the view state of the map is configured to center and scale based on the average coordinates of filtered data, ensuring that all relevant routes are visible.

```
layer = pdk.Layer(
    "ArcLayer",
    routes,
    get_source_position="from_coordinates",
    get_target_position="to_coordinates",
    get_width=5,
    get_tilt=15,
    get_source_color=pickup_color_rgba,
    get_target_color=receiving_color_rgba,
    pickable=True,
    auto_highlight=True,
)
```

Figure 14: Code Snippet

The route visualization feature allows users to obtain a comprehensive interactive view of CDW traffic. By drawing a route between the pickup and receiving locations, users can easily identify and analyze the logistics of waste management.

Arclayer provides a clear visual representation of connections, and custom colors help distinguish various routes, enhancing readability and comprehension. Interactive tooltips add another layer of functionality by providing real-time access to detailed information about each route. For users who need to analyze specific aspects of waste logistics, such as fragment types or transportation volumes, this feature is crucial. The dynamic nature of visualization means that any changes in user input (such as filters or colors) will be immediately reflected on the map, making the application highly responsive and user-friendly.

Pydeck's route visualization transforms raw data into insightful interactive maps, supporting effective decision-making and policy formulation in CDW management. By providing a detailed visual overview of waste logistics, this application helps users identify patterns, optimize routes, and improve the overall efficiency of waste management processes.

Error Handling and User Feedback

The CDW Flow Mapping part incorporates error handling and user feedback mechanisms to ensure a smooth user experience. In the ‘draw_routes’ function, a check is performed to determine if the filtered dataset is empty. This is done by evaluating the condition ‘if not filtered_data.empty:’. If the dataset is empty, the function does not attempt to draw any routes. Instead, it provides immediate feedback to the user by displaying an error message using ‘st.error('No routes found for the selected options.')’.

```
else:
    st.error('No routes found for the selected options.')
```

Figure 15: Code Snippet

The error handling plays a crucial role in maintaining the robustness of the application. This program can prevent potential runtime errors that may occur when attempting to access non-existent data by checking whether the filtered dataset contains any data before attempting to visualize it. This preemptive check ensures

that the application does not crash and remains responsive even if there is no data that meets the user's standards.

User feedback is an important aspect of application design. When a route cannot be found based on the selected filter, the application will immediately notify the user with a clear error message. This instant feedback helps users understand that their current filter selection has not produced any results, allowing them to adjust the standards accordingly. This feature enhances the user experience by providing clear and actionable information, preventing confusion, and guiding users to achieve meaningful visualization effects.

3.2 Database

This application demonstrates a simple and effective method of managing data and interacting with data using Python libraries such as Streamlit and Pandas. The main features of this application include loading datasets from specified GitHub URLs, displaying data in a user-friendly format, and providing search functionality based on user input filtering of data. This code utilizes Streamlit's caching mechanism to efficiently process data and improve performance.

The application first defines a function to load data from GitHub. Then load the dataset into DataFrame and create a copy of a specific subset of the data for operational and display purposes. Implemented a search function, allowing users to filter data based on the "generatorName" column and display the filtering results in a table. Users can select an index from the filtering results to view detailed information about the selected item.

This application demonstrates the powerful capabilities of combining Streamlit and Pandas to create interactive and responsive data applications, providing users with a good experience of exploring and analyzing data.

Data Loading and Caching

The data loading process in Streamlit applications is initiated by defining the function "load_data_from_github", which takes the URL as input and uses Pandas' pd.read_CSV method to read CSV files from the specified github repository. This

function is decorated with "@ st.cache (allow-output_mutation=True)", which is crucial for optimizing performance. This caching ensures that data is only loaded from the source once and stored in memory, allowing subsequent calls to retrieve data without the need to retrieve it from the URL. This greatly reduces loading time and enhances the responsiveness of the application.

When the application starts, it uses the provided github URL to call "load_data_from_github" to load the dataset. The loaded data is then stored in the DataFrame "data". In order to facilitate user interaction without changing the original dataset, a copy of the relevant data subset was created, especially the column "generator name". Reset this subset using indexes to ensure that each row has a unique identifier, which is very useful for later selection and detailed viewing in the application.

```python
import streamlit as st
import pandas as pd

@st.cache(allow_output_mutation=True)  # This allows the function to mutate the cached data.
def load_data_from_github(url):
    return pd.read_csv(url)

# Load data
github_raw_url = 'https://raw.githubusercontent.com/TianyiWuNYU/test/main/data/cdw_csv_processed.csv'
data = load_data_from_github(github_raw_url)

# We ensure that we are working with a copy of the data for display and manipulation to avoid direct mutation.
small_table = data[['generator_name']].copy()
small_table.reset_index(inplace=True)
```

Figure 16: Code Snippet

Caching functionality is crucial for efficient data management, especially when dealing with large datasets or remote sources. By caching data, this application can avoid unnecessary network calls and reduce latency, providing a smoother user experience. In addition, by using data replicas for display and operation, the integrity of the original dataset can be maintained, preventing accidental modifications during user interaction. The data loading and caching in this Streamlit application ensures efficient data processing and responsive user interface, laying a solid foundation for subsequent search, filtering, and display functions. This method not only improves performance, but also ensures a seamless experience for users when interacting with data.

Search and Filtering Functionality

The search and filtering functions are implemented using Streamlit's "st. text_input" and Pandas' string manipulation function. Provides users with a text input box where they can type a query to search for the "generator name" column of the dataset. This input is captured in the "search_query" variable. The filtering process first checks if "search_query" is empty. If there is a query, the code will filter the "small_table" DataFrame to only include those lines where "generatorName" contains the search string. This is achieved using the "str. contents" method, which is not case sensitive and can handle partial matches.

To handle potential "NaN" values in the "generator name" column that may cause errors during string matching, the code first uses the "fillna()" method to replace any "NaN)" values with empty strings. This ensures that the filtering operation will not fail due to data loss. The filtered results are stored in the "filtered table" DataFrame. If no search query is provided, use the original "small_table" DataFrame.

```
# Implement a search functionality
search_query = st.text_input("Search by generator name:")

# Filter data based on the search input
if search_query:
   filtered_table = small_table[small_table['generator_name'].fillna('').str.contains(search_query, case=False)].copy()

else:
    filtered_table = small_table

st.write("Table for `index` and `generator_name`:")
st.dataframe(filtered_table)
```

Figure 17: Code Snippet

The search and filtering functions significantly enhance user interaction by allowing users to quickly locate specific items in the dataset. By typing a portion of the generator name, users can dynamically filter data and only see relevant results, making exploration of large datasets easier to manage. Case insensitive matching ensures that users do not have to worry about the exact case of search terms, thereby improving the overall user experience. After filtering the data, it will be displayed using the "st. dataframe", which displays the data in a table format, making it easy for users to browse. If the filtered dataset is not empty, users can choose to use the dropdown menu (". selectbox") to select an index from the filtered results.

4. Future improvements

Although our work has concluded, there are still several shortcomings that need to be addressed by future teams, should there be any. Here, we offer some directions for improvement to future teams.

4.1 Correction and Evaluation

As we previously mentioned, the validation API has its limitations, particularly its low tolerance for recognition errors. For example, it can correct a minor spelling mistake such as changing "155 Mavroe St, Brooklyn NY 11216" to "155 Monroe St, Brooklyn, NY 11216."

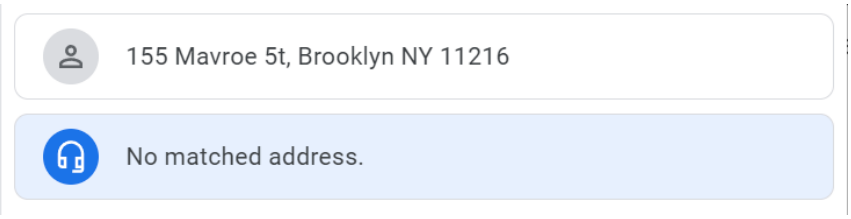


Figure 18: AI Agent Answers Our Question

However, a common recognition error like "155 Mavroe 5t, Brooklyn NY 11216", where 'S' is misrecognized as '5' due to their visual similarity, cannot be corrected by the validation API. We have attempted to integrate Vertex AI with the validation API to address this issue using Generative AI, but the results were unsatisfactory and showed no significant improvement over using the validation API alone (see the AI Agent on the right). Future teams need to consider this direction for improvement.

4.2 Database Improvements

Currently, we don't have a database in the strict sense; instead, we store data in CSV files and extract data from them. It's important to note that our web application runs on code hosted on GitHub, and the CSV files are also hosted there. GitHub does not support changes to its files via non-Git commands, which means

we can't directly modify the backend CSV files through the web app. We originally planned to migrate our data to a cloud database like Google Cloud SQL, but various reasons prevented this from happening. This migration is another improvement direction that future teams should consider.

4.3 Further Model Training

We spent considerable time on model training, but the final results were still not satisfactory. After numerous iterations, the fine-tuned model achieved an F1 score of 0.836, indicating approximately 83.6% accuracy on our training documents. However, the actual performance still fell short of expectations. We hope future teams can further optimize and train the model to enhance its effectiveness.

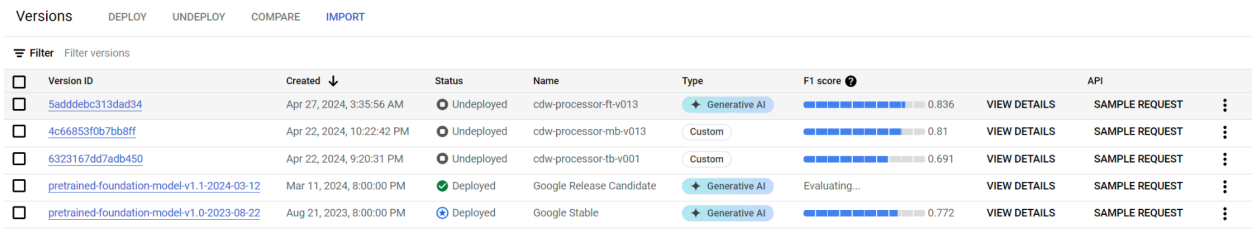


Figure 19: All Versions of Our Models

4.4 Further Web Application Improvements

Advanced filtering options: Adding finer grained filtering features will greatly benefit users. At present, filters are limited to fragment types, pickup addresses, and receive addresses. Future versions may include other attributes such as waste quantity range, specific dates, or carrier details. This can be achieved by extending existing filtering logic to adapt to new conditions, providing users with more precise control over their visualized data.

Performance Optimization

As the dataset grows larger, the demand for optimizing performance becomes crucial. The current implementation can effectively handle small and

medium-sized datasets, but for larger datasets, optimization may be needed, such as asynchronous data loading, caching strategies, and data structures for performance tuning. Streamlit's cache (@ st. cache) can be further utilized to minimize redundant data processing and enhance application responsiveness.

Enhanced Visualization Features

Visualization components can be extended by integrating more advanced features of Pydeck, such as different types of layers (such as ScatterplotLayer, GridLayer), to represent other dimensions of data. Interactive controls such as time based data sliders can also be introduced to visualize the flow of data over time, providing a dynamic view of changes and trends.

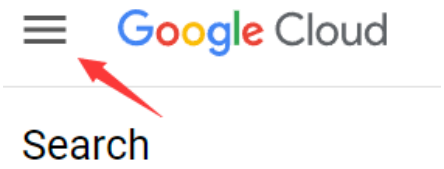
User Interface Improvement

Finally, improving the user interface, including more information panels and user-friendly elements, will enhance usability. This can include better error messages, detailed legends of map layers, and tooltips with richer information. Implementing these enhanced features will make the application more intuitive and engaging, facilitating more insightful analysis and decision-making.

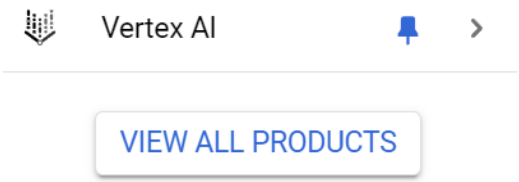
Appendix A
A Simple Guide on GCP Tools

How to Find the Tools

Sign up or Sign in to your Google Cloud Platform. Create a project or select an existing project first (nyc-ddc is our project name here). Then in your Google Cloud Platform Console, go to the right-upper corner, click the sidebar.

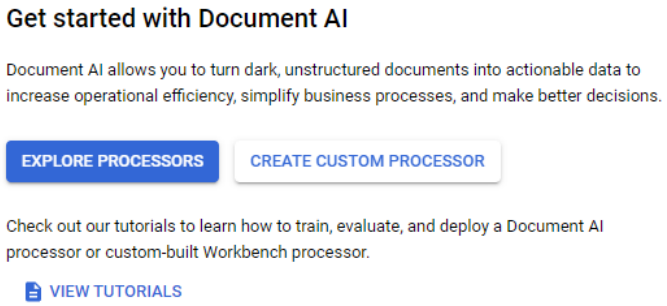


If you have already enabled the products/APIs you want, you can see them in the sidebar. Or click “View All Products” to search and enable them.

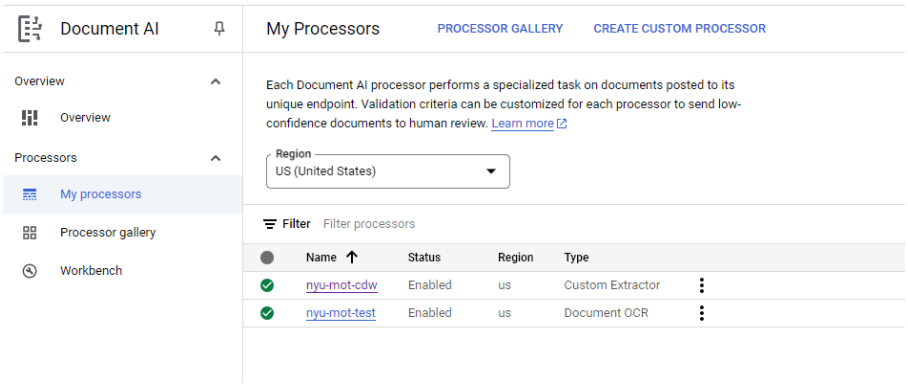


Document AI - Get Started

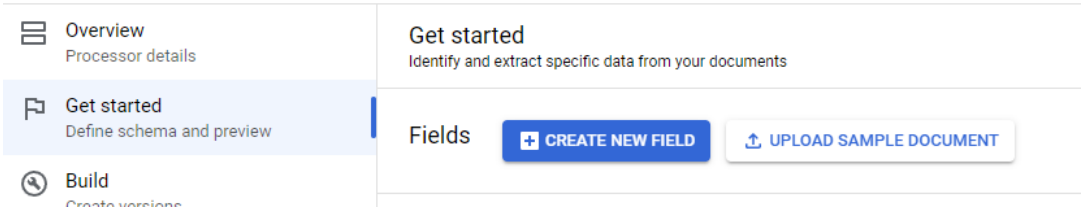
Find and enable the Document AI feature of GCP. Click “Create Custom Processor” to create a new processor, or you can click “Explore Processors” to use the existing models you created.



Select My processors in the sidebar, and select the corresponding processor in the filter. In this project, "nyu-mot-cdw'' is our processor. You can create your own processor in the “Processor Gallery” tab in the sidebar.

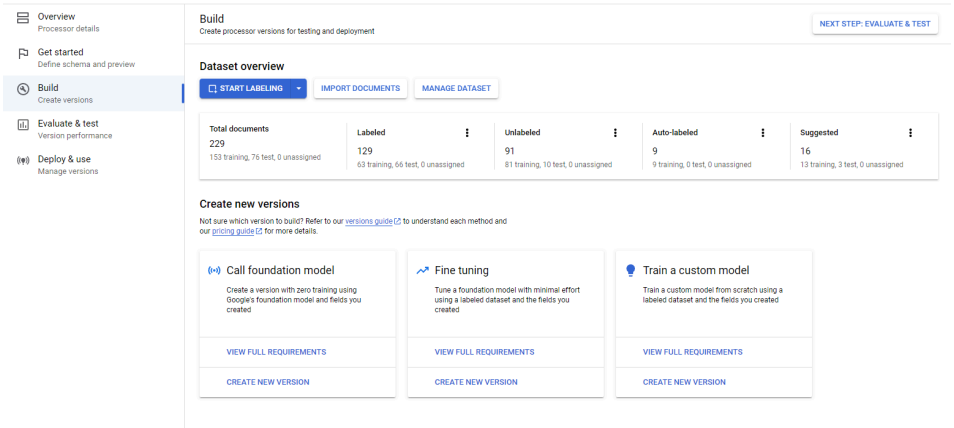


Find Get Started in the sidebar. You can click “Create New Field'' to specify your fields, which are the information you want to extract from the documents. After you define the fields, you can upload the sample file to check your setting/schema.

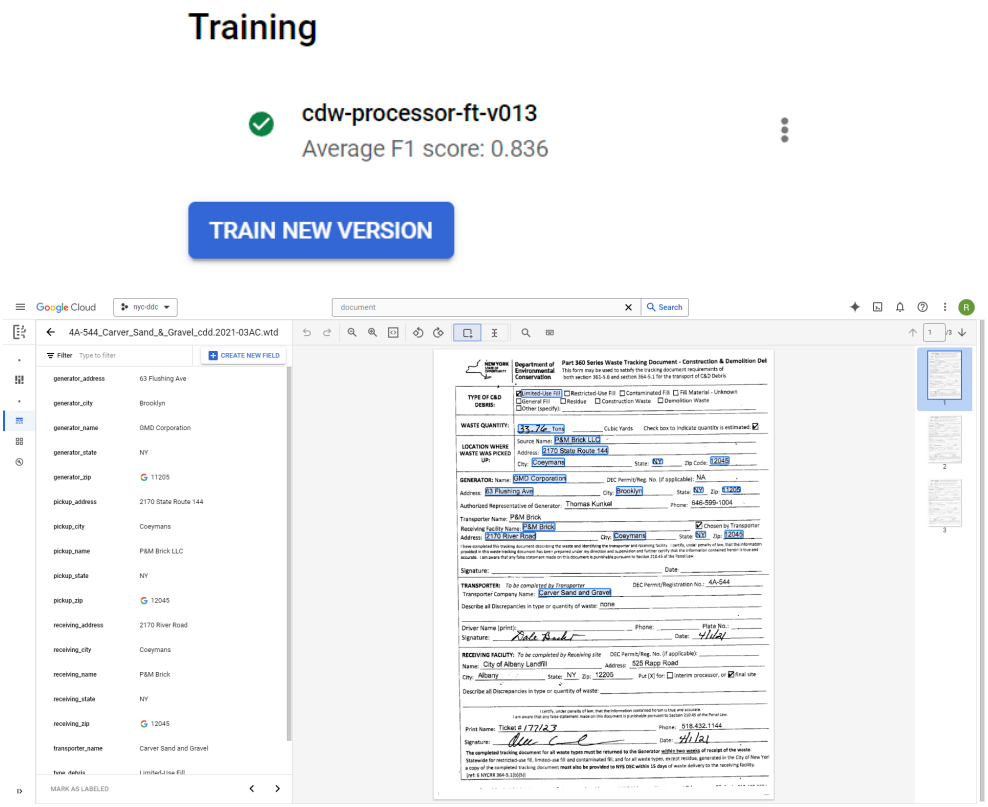


Document AI - Build

The Build page is used to create our model for testing and deployment. You can upload your training data, which should be some documents here. And like most supervised models, you should do the labeling.

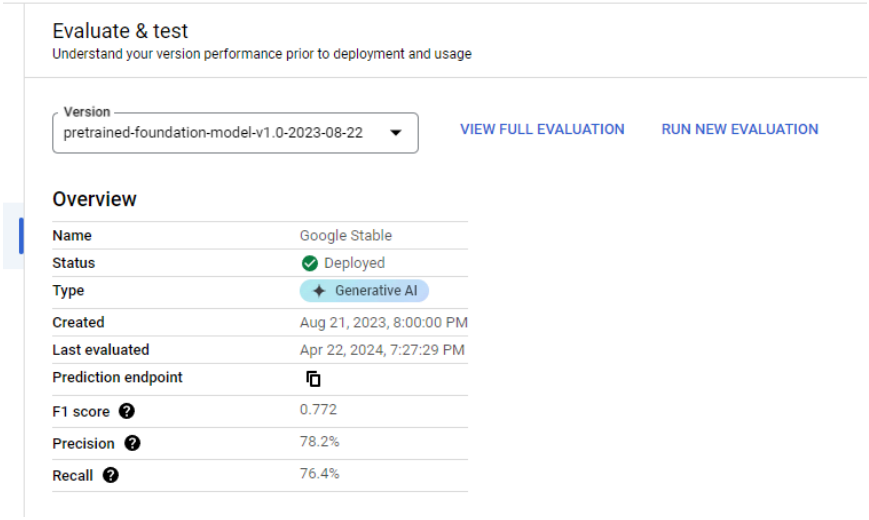


In the “MANAGE DATASET”, you can manage your dataset, do the labeling, upload more documents, and train your model.



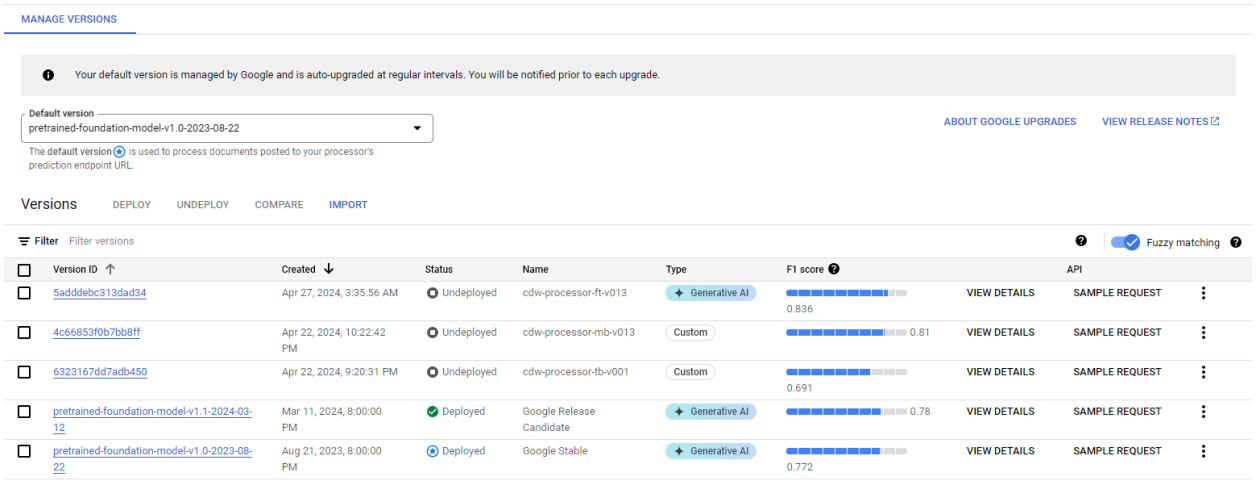
Document AI - Evaluate & Test

You can get an idea of the performance of models before deployment, including the F1 score, Precision, and Recall, all these shows the accuracy of your models.



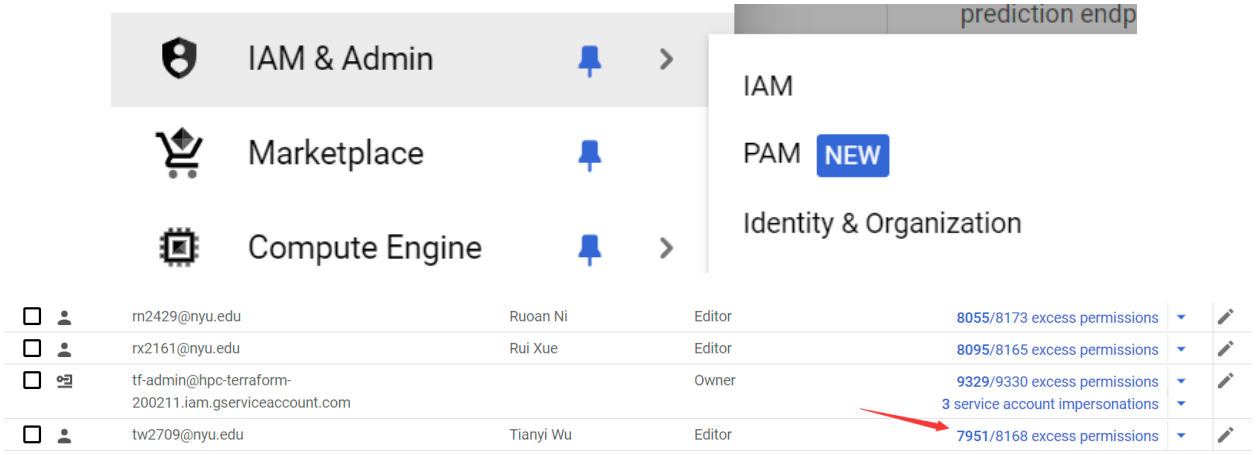
Document AI - Deploy & Use

You can train several versions for models. In this page, you can choose which version is your default version, and which you want to deploy for use, and which version you want to discard.



Document AI - How to Call it via API

First, check your access permission. In most cases, if you have the editor permission in your project, you will also have access to the products/APIs used in the project.



Permissions for tw2709@nyu.edu

Current permissions for Editor role

```
documentai.processors.create
documentai.processors.delete
documentai.processors.fetchHumanReviewDetails
documentai.processors.get
documentai.processors.list
documentai.processors.processBatch
documentai.processors.processOnline
documentai.processors.update
documentai.processorTypes.get
documentai.processorTypes.list
documentai.processorVersions.create
documentai.processorVersions.delete
documentai.processorVersions.get
documentai.processorVersions.list
documentai.processorVersions.processBatch
documentai.processorVersions.processOnline
documentai.processorVersions.update
errorreporting.groups.list
iam.roles.list
```

Once you have confirmed that you have the permission to use Document AI, you should set up authentication in your work environment. The easiest way to do this is use Google Cloud CLI. You can download it here: link.

Then, run the following command in your Command Prompt/Anaconda Prompt/Powershell Prompt :

```
gcloud init
```

And run this command to create local authentication credentials for your Google Account. You will be asked to enter your GCP account and password.

```
gcloud auth application-default login
```

Check these locations for the json file. If the json file exists, you successfully set up authentication.

Linux, macOS: $HOME/.config/gcloud/application_default_credentials.json

Windows: %APPDATA%\gcloud\application_default_credentials.json

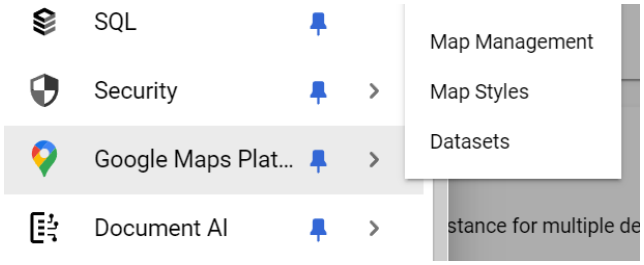
Now you can call Document AI, send it a request, use it to process your documents in your work environment, like your PC.

For code example, check :https://github.com/NYU-Tandon-TMI/cdw/blob/main/toolbox/document_extractor_one_folder.ipynb

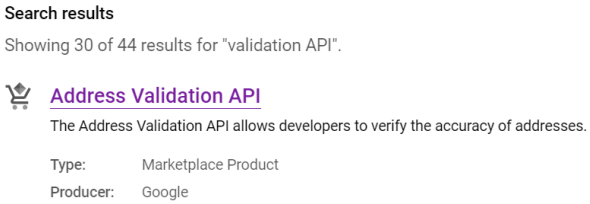
For more details, check: https://cloud.google.com/document-ai/docs/

Validation API

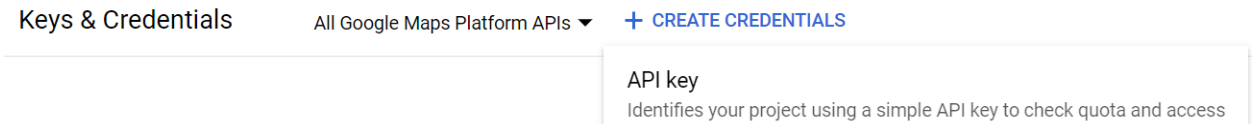
Validation API is among the numerous APIs & Services provided by Google Maps Platform. Similarly, find and enable Google Maps Platforms in the sidebar of your GCP console.



And go to the APIs & Services, find and enable Validation API



This API is easy to use. What you should do is create API keys, and put it in your request code.



For code example, check: https://github.com/NYU-Tandon-TMI/cdw/blob/main/toolbox/document_preprocess.ipynb

For more details, check: https://developers.google.com/maps/documentation/Address-validation

Appendix B
Document Analysis

NYS Tracking Documents	
Folder	**Pages**
R1	47814
R2	44357
R3	9534
R4	9658
R7	138
R8	7
SUM	111508

NYS Tracking Documents				
Folder	**Different**	**Handwriting**	**Printed**	**Missing Info**
R1	0	40666	7148	5000+
R2	2500+	44357	0	1000+
R3	0	9530	4	0
R4	0	0	9658	10+
R7	0	0	138	0
R8	0	0	7	0
SUM	2500+	94553	16955	6100+

* Printed means most of the key information is printed.
* All the values with ”+” are estimations.

Out of State Tracking Documents	
Folder	**Pages**
CT	1
NC	268
NJ	19068
PA	1848
SUM	21185

Out of State Tracking Documents				
Folder	**Different**	**Handwriting**	**Printed**	**Missing Info**
CT	0	0	1	0
NC	0	0	268 (Poor)	0
NJ	500+	18265	803	0
PA	0	1423	425	0
SUM	500+	19688	1496	0

* Printed means most of the key information is printed.
* All the values with ”+” are estimations.