**A Note on the Usage of my program**

The first thing to notice about this document is that it is verbose. This is intentional. I have attempted to outline every detail and every step with great precision, to avoid future pain and confusion. Also, I am assuming that since there are so many documents left to be scanned, the person who will be completing this project should invest some time into learning the fastest way to approach it (given that it is indeed a multi-month project). This document gives my opinion on how to do that, so while it may take a few days to comprehend, it may be worth it in the end. It is worth noting that the benefits of this program come from working on great bunches of documents at once. If you are only scanning a few folders and calling it a day, it may not be worth your time to invest in learning how to use this program.

**Document Automation Technical Specification**

This document is intended to shed light on how to use the Python scripts that I have written to automate redundant tasks when converting physical documents to digital PDF copies. The work can be divided into two distinct phases:

1. Phase 1 is primarily physical: you will be working with the physical copies of the documents to scan them.
2. Phase 2 is primarily digital: you will be working to label the documents on the computer based on the data they contain.

The automation scripts greatly reduce the work required for both categories. However, this will come at the cost of having to perform the work according to a set of rules, some more strict than others.

In this document, I will first cover the technical details of the scripts that I have written, aka what input they take, what modifications they make to the input, and the structure of the output. The second part of this document covers the specific steps of what I would do on a day-to-day basis, aka the most effective way that I found to use the scripts.

\*Note for future scanners, if a folder has a blue number written on the tab, it has already been scanned.

**Part 0: Terminology**

* Batch – You will process *n* number of physical folders at a time. This is referred to as a batch of folders. This typically is from a quarter of a box to half a box of physical folders.
* Blank – A document is considered “blank” if the second page on a scanned sheet is blank. The ScanSnap app will automatically delete blank pages, so these documents are blank. However, some documents do have empty blank pages that were not detected as so, so you will have to mark these.
* Backpage – ScanSnap scans documents one sheet at a time. It will create one pdf document per sheet. Sometimes you want a single pdf document to be composed of multiple sheets of paper, so you will have to merge the individually created pdf documents. A document marked as “Backpage” will be merged into the directly preceding document. If more than one document is marked as “Backpage” subsequently, they will all be merged under the first preceding document that is not marked “Backpage”. For example, if document “#001” is not marked as “Backpage,” but “#002” and “#003” are, then both “#002” and “#003” will be merged under “#001.”
* Form – A form document is a kind of document that almost always encompasses Initial, Update, and Renewal documents.

**Set Up**

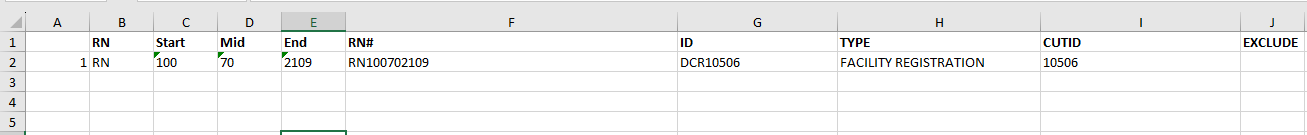
* Have Python and Visual Studio Code from the software center installed on your computer.
* Open Visual Studio Code. Under File, select Open Folder. Select your Workplace folder, under which all the scripts and files are organized.
* Open the terminal. You can do this by going to the Terminal tab at the top. Under it should be an option for a “New Terminal” Select this option.



* Enter this command in the terminal, pressing enter to run it: *pip install -r requirements.txt*
* Wait for this to run. You should only have to do this once per machine.

**Part 1: Instructions**

These are the direct, most simplified instructions for using my tools. If you get stuck, confused, or need additional clarification, see the Part 2: Reference and Documentation section below.

1. Have Visual Studio installed on your computer. Open the “Workplace” folder. This contains all the files you will need. Under the “Terminal” dropdown at the top, select “New Terminal”
2. 
3. Note: For purposes of organization, we will be organizing groups of folders into batches. Batches are sets of *n* folders labeled from 1 to *n*. Following each of these steps will result in one complete batch.
4. Take a pen. Write Batch #*n* on the first folder. (I last finished Batch #16, so you would start at Batch #17)
5. Label each of the folders *1* through *n*, with the first folder labeled *1*.
6. Locate the “ConversionTemplate.xlsx” spreadsheet file. Duplicate it, and rename the copy to “Conversion.xlsx”
7. Open the newly created “Conversion.xlsx” spreadsheet in Excel. Delete every row except for the first two rows: The header row and the first row with data. 
8. Drag out column A so that it contains a list of numbers, labeled from 1 to *n*. The idea is that each row will correspond to the number that you wrote on each of the folders.
9. Drag out the RN column for each of the rows, so that there is an “RN” next to each of the numbers.

A screenshot of a spreadsheet

Description automatically generated

1. For each of the rows, write the first three digits of the RN number in the “Start” column. These should repeat quite a bit, so you can drag-fill the numbers to save time.
2. For each of the folders, you can write the remaining digits in the mid/end columns. You should do this however you see fit. Sometimes, you can save time because the second few digits will repeat quite a bit, so you can drag-fill to same time. However, this will not always be true. The important thing to realize is that the values in the “RN, Start, Mid, End” columns are joined together to create the full RN number.

A screenshot of a computer

Description automatically generated

1. In row number 1, with the data you didn’t delete, there should be Excel formulas in the “RN#, ID, TYPE, CUTID” columns. Drag these formulas out for each of the rows. If you do this correct, they should fill up with values that make sense. The RN# column should fill in based on the values you entered with the Start, Mid, and End columns. The ID and TYPE column should fill in with DCR numbers and Facility Types (and #N/A). The CUTID column should be equivalent to the DCR column but without the letters.

A screenshot of a computer

Description automatically generated

1. It may be the case that some of the values are missing. This is because they were not found in the Historical\_Dry\_Cleaner\_Registrations.xlsx sheet provided. In this case you will have to go to this site and manually find the DCR numbers using the RN number. <https://cf9int.tceq.texas.gov/crintrpt/index.cfm?fuseaction=main.welcome>. DO NOT CHANGE HOW THINGS ARE FORMATTED OR WRITTEN. Everything should be consistent. For the Type, do not just write something like “Facility” or “Drop Station.” You must write the full name: “FACILITY REGISTRATION” or “DROP STATION REGISTRATION.”
2. You may find that some of the columns do not have a valid DCR number even after searching on the website. For each of these rows, write the letter “g” in the EXCLUDE column, and set aside these folders. You are not going to scan these folders. Organize these however you see fit.

A screenshot of a computer

Description automatically generated

1. It is a possibility that you simply mistyped the RN number if it is not found. Be sure to keep a lookout for this throughout these stages.
2. This is the end of preparing the Conversion.xlsx sheet. We now move on to scanning the documents.
3. We will be using the ScanSnap machine to scan documents. You interact with this machine by using the ScanSnap app. ScanSnap has “profiles” to save settings, so you should create a new profile to configure with specific settings. The Scanning Side should be duplex. The image quality should be excellent. The file format should be pdf. In detailed settings, go to the Title section and select “Customize File Names.” Set this to “doc”. It can be either grey or colored, although I scanned mine in color. In pdf settings, set the pdf page split to “create a PDF file per number of specified sheets.” Set this value to one. Once you have created this profile, you will not have to create it again.

A screenshot of a computer program

Description automatically generated

1. Open the terminal. You can do this by going to the Terminal tab at the top. Under it should be an option for a “New Termina.” Select this option.



1. In the terminal, type the following command: *python ImprovedAutoScan.py*. This will run the ImprovedAutoScan.py script. Ensure it starts at Index 1. It should start outputting timestamps at regular intervals.
2. Begin the scanning. For each folder, you should remove staples and papers that should not be scanned. You will then place the stack of papers in the ScanSnap machine. In the ScanSnap app, click the Scan button. This should start the machine. Start with the folder you labeled “1”. You MUST make sure that if you separate staples sheets, you must scan them subsequently and in order. For example, if you separate a stapled sheet, the back sheet must directly follow the front sheet in being scanned.
3. Write the DCR number on the folder (which only has the RN number).
4. Since my Scanning machine would take a while to scan, I would do other tasks concurrently, such as removing staples and writing the DCR number on the folders.
5. After the documents are finished scanning, you should see a dialogue box pop up asking where the folders should be saved to. Make sure to save them in the folder labeled “Scan.” After placing the files here and waiting a few seconds, the files should be automatically transferred from the “Scan” folder to a folder labeled “001” in the folder labeled “Result”.
6. Repeat steps 20 – 23 for every folder in the batch. Periodically check to make sure the number on the digital folders corresponds with the written label on the physical folders. (So, the scanned folder #1 goes into a digital folder labeled “001”, folder #2 goes into “002” etc.)
7. After you have finished Scanning the documents, there should be a digital folder corresponding to each physical folder. Check for this to be the case in the “Result” folder.
8. Click onto the black terminal in VS Code. Press Ctr-c to kill the ImprovedAutoScan.py program.
9. Type this command into the black terminal in VS Code: *python InitialDataPreparation.py*
10. Type this command into the black terminal in VS Code: *python Viewer.py*. This should launch a desktop app.

**An Overview of the Viewer program**

You have 5 goals to accomplish with the Viewer program:

* 1. Mark blank pages for deletion
  2. Merge separated document pages together
  3. Mark the code of each document
  4. Write the date
  5. Correct other errors

If this is all a bit much, feel free to experiment with the program on test data. This will give you a good idea of what is happening.

**REMEMBER, USE THE “s” key to SAVE YOUR WORK. THIS IS THE ONLY WAY YOUR CHANGES ARE SAVED. Save often, there is a bug that might occur that will prevent you from saving, losing all your work from the previous save.**

Here are the tools that allow you to accomplish this:

1. **Blank page marking:**
   1. The “b” key will toggle whether the second page of a document is considered “blank” or not.
   2. The significance of this is that the second page of every document marked “Blank” will be deleted.
   3. ScanSnap will automatically try and detect which pages are blank and delete them. These deleted pages are represented with a blue gradient background and text that says, “Blank Page.” These will be obvious.
      1. The blue gradient image will only show for pages deleted by ScanSnap. Marking something as blank does not cause this page to display.
2. **Merge pages:**
   1. Use the “n” key to toggle whether a pdf is marked as the “Backpage”. Any page marked as the “Backpage” will be merged into the closest preceding document not marked “Backpage.”
      1. Backpages DO NOT HAVE DATES and DO NOT HAVE CODES. This makes sense. These are not individual documents. These pdfs will be appended to another pdf, and as such will not be named individually.
      2. NOTE IMPORTANT: If you activated a filter, the preceding document may have been filtered out. Backpage does NOT consider filters, and will still base the preceding documents on if there were no filters.
      3. If this doesn’t make too much sense, just look at the document number, aka the yellow number at the top. The directly preceding document is one less than that number.
   2. When the final pdfs are created, any document that has the same name, aka the same DCR, same Date, and same Coding, will be merged into one another.
3. **Mark the code of every document**
   1. All documents are marked as Renewal by default
   2. When the final documents are created, any document with the earliest chronological date will be marked initial.
   3. Use the “q” key to mark a doc as Initial
   4. Use “w” key to mark a doc as Update
   5. Use the “e” key to mark a doc as Return Mail
   6. Use the “r” key to mark a doc as GC
   7. Use the “t” key to doc a Renewal
4. **Write the Date of Every document**
   1. Click the “Enter” key to toggle Date mode. Date Mode allows you to enter letters and numbers for each documents Date. The program accepts a wide variety of date formats. If you want to be safe, stick with these: YYYYMMDD. mmmDDYYYY. MM/DD/YYYY.
   2. Date mode means none of the other functions will work. The reason is that you might need those letter keys to type days instead.
      1. Furthermore, “+” and “-“ can’t be used in Date Mode to scroll through documents. Instead use the “Tab” and “~” keys to go forward and backward respectively.
      2. Use arrow keys to scroll through the date input bar.
   3. You can’t save in date mode. You must toggle out of it to save.
5. **Correct Other Errors**
   1. These are very rare errors that you might encounter for one reason or another. Sometimes documents are flipped the wrong way or
   2. Press the “r” key to rotate the first page of a document. There is no functionality to rotate the second page of a document.
   3. Press the “k” key to mark a document for total deletion.

The program also features other very helpful features that you can use:

* **Scrolling**
  + The most basic functionality of the program. Press “+” or “Tab” to scroll forward. Press “-“ or “~” to scroll backwards.
* **Filtering**
  + Filters out any documents that do not meet the criteria.
    - Date filter filters out documents depending on if they have a date written or not.
    - Ignore Date Type. This was for a tool that I used but I excluded for several reasons.
    - TwoPage filter filters out documents depending on whether something is marked “Backpage” or not. The “Is Front Page” option in this context means not marked “Backpage.”
    - Action filter filters out documents based on their coding.
    - Blank filter filters out documents based on if they are marked blank or not. If ScanSnap deleted the back page of one of these documents,
    - Page Num Filter filters out documents based on if the ScanSnap deleted one of their pages or not. Often, you will just use the blank filter instead of this.
  + Click apply to use. You will most often use TwoPage filter, and Blank Filter.
* **Search**
  + This searches for the document number. The document number is displayed in yellow near the top right of the window (Ex. Document 23 in Folder 4. You would search for 23 if you wanted to find that document).
    - Type in the number of the document you want to find in the search bar. Click Go. It will not be found if it does not match the current filter conditions.
  + You can also search for dates that are detected to be invalid (they do not exist). This is good to discover mistakes that you made while writing dates.
    - Click the “Find Invalid Date!” Button. This also will only search within documents that meet the current filter criteria.
    - The trick is that you must filter out documents which intentionally do not have dates, like “Backpage” documents (since they are part of a different document). Empty dates are “Invalid,” so they will be returned.
* **Saving**
  + Saving is vital! You must manually save by clicking “s”. A message will pop yellow if this is successful.
  + Saving does not work in Date Mode. You must toggle off date mode for it to work.
* **Backpage flags**
  + You may notice that sometimes red and green bars will pop up next to your pdf images. These are flags that convey specific information that will help you identify the back pages of GC documents, which often span multiple pages and vary wildly. GC documents are tough because while you want to merge documents that had been stapled together, there isn’t really a consistent rule to do this. Deductive reasoning is required (match staple marks, see if there are page numbers, etc.) However, these flags make it a little bit easier.
    - If there is a red bar, you absolutely do NOT want to mark that document as a Backpage.
      * That bar indicates that the current document is the first in its DCR series. If you were to mark it as a back page, it would be merged into the previous document which belongs to a different DCR, effectively meaning that document is lost.
    - The green bar is a bit complex. If it doesn’t make sense just ignore it.
    - To understand the green bar, you must understand the filter system.
      * The green bar says that the filtered preceding document, aka the one that appears when you click “~” or “-“, is the TRUE preceding document, aka the one that the current document would be merged with if it were to be marked “Backpage.”

The Viewer program is quite open ended on how you might accomplish this, and simply provides a series of tools to accomplish this. I will first give you an overview of the tools that are available to you. I will also give you the strict routine that I followed when using said tools. You can choose to use the tools however you want. For example, you can go document by document correcting everything at once in a document.

**My specific workflow for using the Viewer program**

1. Under “Filters,” set the “Blank Filter” to “Has No Blank Page.” Click apply.
2. Most of the blank second pages have been already detected, but some have not been. You will now go through and mark all the blank pages that the program missed. Note, that “Blank” only refers to if the second (left) page is blank. Marking blank will ensure the second page is deleted, so only use it if you know that there is no important information on the second page.
3. Whether a pdf has a blank second page is indicated clearly to the right of the pdf images. Use the “b” key to toggle whether the second page is blank or not.
4. Use the “+” key and “-“ key to scroll through the pdfs. The “+” key goes forward, the “-“ key goes backward. Notice all the documents are in the order that you scanned them. The documents from folder 1 are the first that you will encounter. Folder and document number are displayed on the righthand column.
5. After you have corrected all the blank pages, you will now proceed to marking back pages. Some of the documents that you scanned were split over two separate pieces of paper. This step concerns merging these papers. If a page is marked as “Backpage,” it will be joined onto the directly preceding pdf document. So if document “#002” is marked as “Backpage”, it will be appended to the back of document “#001”.
6. This instructible generally assumes that you are familiar with the documents you are processing. If so, you should realize that Initial, Update, and Renewal documents tend to be of a specific kind, while GC documents wildly vary in format. I will refer to Initial, Update, and Renewal documents as form documents from now on.
7. Form documents tend to consist of two pages. Many of these form documents are two pages printed on a single *sheet*, front and back, and so will have already been combined in one pdf. However, some of these forms will have been printed over *two sheets* of papers, front on one and back on another. In this case, the ScanSnap program will have stored them in separate pdf files, so you will need to merge them using the “Backpage function.
8. We will ignore merging separated GC sheets for now due to their complexity and variability. The forms are very consistent, and all the back pages look similar. It is also guaranteed that a back page will not have a front page, so you will only need to search through blank pages to find back pages.
9. Set the blank page filter to “Has Blank page.” Click Apply. There should only be documents with blank pages now.
10. Go through these and mark all the documents that seem to be the back page of a form. Whether a page is considered a Backpage is displayed to the right of the pdf images. Toggle this value with the “n” key.
11. After marking all Backpages, set the blank filter to ignore and the TwoPage filter to “Is Front Page.” Click Apply.
12. Go through these documents are mark their coding as either Renewal, Initial, Update, Return Mail, or GC. Use “q” for Initial, “w” for update, “e” for Return Mail, “r” for GC, and “t” for Renewal, though all are set to Renewal by default. Ask around if you don’t know what should be coded.
13. Just remember that at the final step of the process (*Finalizer.py*), the earliest dated form document is force-set to Initial. Don’t set a document to Initial unless you have no other choice.
14. Set the “Backpage” filter to ignore. Set the “Action” filter to GC. Click Apply.
15. Go through the GC documents and mark them as “Backpage.” Setting a document of “Backpage” means that they are not a separate document, but rather a separated page belonging to a previous document. Hence, all documents marked “Backpage” are merged into the first document before them that is not marked “Backpage” (This document represents the first page of the total document). You can also set their Dates at the same time (Press “Enter” to go into Date Mode). I don’t really have a consistent way of finding out which GC documents are the back pages of other documents, so you will have to use your judgment to decide which documents go together and which dates to keep. Another way to merge GC documents is to ensure that they have the same date. Because they have the same DCR and same coding, having the same date will ensure that they have the same name and are merged.
16. If the previous step concerns you, just remember that there is a high degree of error tolerance afforded to this process. Documents that go together look similar. Documents that have the same staple holes were likely stapled together. Even if you fail to merge documents using the “Backpage” method, documents with the same date are merged together in the end (Since they are both GC and belong to the same DCR), so if they really are of the same bundle, they should have the same date.
17. You can also use the Color bars during this step. If there is a Red bar, NEVER USE BACKPAGE. If there is no green bar, YOU SHOULD NOT USE BACKPAGE, unless you intentionally want to merge that document with the preceding form. It is important to realize that green bars indicate the documents were scanned subsequently after the previous, and as such might have been stapled together. If there is no green bar, that means there was a Form document in between them.
18. Stapled documents tend to have very close dates, so even if you fail to merge these documents, two GC documents with close dates (very similar file names) will be created, and nothing much is lost by their lack of merger.
19. After you have processed the GC documents, set the action filter to “Ignore”. Set the TwoPage filter to “Is Front Page.” Set the Date Filter to “No Date.” Click Apply.
20. Press “Enter” to toggle Date Mode. Go through all of these documents and enter the date printed on the stamp. Safest formats are YYYYMMDD, mmmDDYYYY, and MM/DD/YYYY.
21. Set Date Filter to ignore. Set TwoPage filter to “Is Front Page.” Click Apply
22. Click the Find Invalid Date. If it finds a date, correct it. Repeat this until it doesn’t find a date.
23. You are done with the Viewer Section. Save and Close the program.

**Final Steps**

1. Go back to the black terminal console. Run this command: *python Finalizer.py*
2. This creates the files. Look in the folder named “Labeled.” All there, like Magic. Do what you will with them, but that is the end of our process, if you did it correctly.
3. Take a look at prudent measures.
4. If you want to start another batch of folders, delete everything inside of the Result folder. Delete the Conversion.xlsx sheet and the document\_data.csv sheet. These are no longer needed and should be recreated fresh every batch. Move everything outside of the Labeled folder to some place safe.
5. It is perfectly valid to use File Explorer to move and delete files. Don’t stay confined in Visual Studio Code.

**Prudent Measures**

1. The finalizer script should print some output to the terminal. If it says something about a document having an Invalid Date, you did NOT correct all invalid dates. Go to the document it references and correct it. After you have completed this, run *python Finalizer.py* again. It will also make some notes on which files had been date merged. Make sure this makes sense.
2. You can run *python Finalizer.py* as many times as you want, so if you messed something up, you can always open up the viewer again (*python Viewer.py*), edit it, then run *python Finalizer.py*
3. TO BE EXTRA PRUDENT: To be extra safe, I’ve been saving the Result folder, the document\_data.csv, and the conversion sheet after I’ve finished with them. If you saved these three files, you can restore the state of your project by simply moving them back into the Workplace folder, replacing the existing Conversion, document\_data.csv and Result files in there. If you mess up super badly in previous files, this allows you a saving grace. These backups have been saved in a folder named “Environment”.
4. \*There may be a caveat to this, the program *may* be using global file paths, so if you move the workplace folder to another computer or folder, the backups may not work. If you think it prudent, back up these files. If you don’t care, just ignore all this.

**Part 2: Reference and Documentation**

This part is to be used if you get stuck on something and need a deeper understanding of what it does, or if you would just like a deeper understanding in general.

This part goes into a high-level (not too technical) analysis of the scripts function. Although in theory you do not need to know this, and probably could get away with strictly following the direct steps in part 2, it is useful to understand what it is that you are doing and to have a deeper understanding of the tools you are working with. This could possibly benefit you if something goes awry, because you will likely understand *why* it has gone awry. Also, the steps detailed in Part 2 are only one of the many possible ways you can use the software, so if you understand it well, you might be able to find a workflow that you like more.

It might not make too much sense on a first time reading. It may be better to use as a reference if you get confused when using the program.

1. Conversion.xlsx
   1. A spreadsheet that uses excel functions to automate the conversion of RN numbers to DCR. The ID and Type columns are automatically filled based on the RN# column. The RN# column is generated from the “RN, Start, Mid, End” columns being joined together. If you want to exclude a folder from being considered by other scripts, type “g” into the EXCLUDE column. If you know excel stuff, this should be the least obscure thing.
      1. INPUT:
         1. RN numbers written directly on folders
         2. Historical\_Dry\_Cleaner\_Registrations.xlsx (Used for Lookup)
         3. <https://cf9int.tceq.texas.gov/crintrpt/index.cfm?fuseaction=main.welcome> (Use to enter manually if not automatically found in lookup sheet.)
         4. Folders to be excluded
      2. MODIFICATION:
         1. Combines text in the “RN, Start, Mid, End” columns into the RN# column.
         2. Uses the RN# column to perform an excel VLOOKUP on the Historical\_Dry\_Cleaner\_Registrations.xlsx sheet to find the corresponding DCR number and facility type, which will be placed into the ID column and Type column respectively.
         3. Separates the numbers from the letters in for the CUTID column
      3. OUTPUT:
         1. The EXCLUDE column is used by ImprovedAutoScan.py to generate empty dummy folders in their place.
         2. The InitalDataPreparation.py uses the information from this sheet to label documents.
         3. Index column is important (aka the list of numbers before the RN column). It MUST match up with the names of folders in result. Although, if you keep everything simple, numbered from 1 to *n*, this shouldn’t be a problem.

A screenshot of a computer

Description automatically generated

1. ConversionTemplate.xlsx
   1. This is just an old Conversion.xlsx sheet. In order to avoid recreating all of the excel commands and columns, I would just copy this file and rename it to “Conversion.xlsx” You can also use it as an example of what the conversion sheet should look like.
      1. OUTPUT
         1. Use this to recreate a fresh Conversion.xlsx sheet for a new batch.
2. Historical\_Dry\_Cleaner\_Registration.xlsx
   1. This is not my file, but it is used by the Conversion sheet to lookup DCR numbers from RN numbers. It uses an Excel VLOOKUP function.
      1. OUTPUT
         1. Conversion.xlsx uses this sheet via an Excel VLOOKUP.
3. Scan Folder
   1. There is a folder named Scan that is important. This folder is where scanned files should go automatically. Configure your ScanSnap program to send files to this folder by default.
      1. INPUT
         1. ScanSnap Program
      2. OUTPUT
         1. Folders in here are detected by the ImprovedAutoScan.py program.
4. ImprovedAutoScan.py
   1. This is the first of the Python files you will be using, and the only one to be used during the scanning phase. This script provides a simple convenience: after detecting when a folder’s files have been scanned, it will create a numbered folder for it and transfer the files into it. For example, after scanning the documents of a physical folder, it will automatically create a digital folder named “001” and move the scanned pdf files into it. Open scanning the next physical folder, it will create a digital folder named “002” and transfer those files into that.
   2. Technically, you this isn’t really a necessary program. If in the end there are folders in the Result folder that contain labels that match the numbers/indexes in the Conversion.xlsx sheet, then the InitialDataPreparation.py file will still function. All this is a matter of matching the information in the Conversion.xlsx sheet with the scanned pdf data in the Result folder.
      1. INPUT
         1. Scan folder – Every 10 seconds or so, this program will check if there are files in the “Scan” folder.
      2. OUTPUT
         1. Result Folder - This contains the files organized by the folder in which they were scanned. The digital folder names should match the label you placed on the physical folder (1 through *n*)
5. Result Folder
   1. Contains the scanned documents organized into folders. The digital folder names should match the physical label you wrote on the physical folder. Documents should be named something like “doc.002” or similar.
   2. INPUT
      1. ImprovedAutoScan.py – This program sends information from the Scan folder to the Result folder.
   3. OUTPUT
      1. InitialDataPreparation.py – This takes information in the Result folder and matches it with information in the Conversion.xlsx sheet.
6. InitialDataPreparation.py
   1. This script does a great many important things that I will not go into too much length about, but basically it creates the document\_data.csv file that stores all the information about the documents. It is a prerequisite to using the Viewer.py
      1. INPUT
         1. Result Folder – Uses the scanned pdf data inside of here
         2. Conversion.xlsx – Uses the folder specific information to apply to each document
      2. MOD
         1. Basically applies the information in conversion sheet to the Result folder. It matches the folder names to the index numbers in the Conversion sheet.
         2. Prepares and initializes stuff for the Viewer.py
      3. OUTPUT
         1. document\_data.csv – This program actually creates this csv and initializes all its data. It applies folder specific information to this csv, such as DCR number and Facility type.
         2. Viewer.py uses the document\_data.csv to store information.
7. document\_data.csv
   1. This csv contains one row per document. You will not be editing this directly, though I guess you can if you want to. It stores a path to each pdf file. THIS MAY BE A GLOBAL PATH, so take precaution with moving the pdf documents around.
      1. INPUT
         1. InitialDataPreparation.py – This creates this document and initializes its values.
         2. Viewer.py – This is the GUI interface into this csv. Use this to make edits.
      2. OUTPUT
         1. Finalizer.py uses this to create the final documents.
8. Viewer.py
   1. The usage of Viewer.py is discussed in great detail in the instruction section, so I will not repeat myself. Learning this program is like learning any other, say photoshop or teams. There is a degree of effort that is involved. However, its nothing that the average person couldn’t do.
      1. INPUT
         1. document\_data.csv – This program is basically a GUI interface into this file. This file contains the information that is displayed in the Viewer. The edits made in the viewer will reflect upon this file.
         2. To a certain extent, you can consider this program to be an alternate version of Excel. In Excel, you have a visual interface and tools to edit a csv file. In this program you have a visual interface and tools to edit the csv. In fact, you can open the document\_data.csv file in Excel, and edit it from there. The same idea applies to the Viewer.
      2. OUTPUT
         1. This program edits the document\_data.csv
         2. In the grand scheme of things, you are preparing the document\_data.csv for usage by Finalizer.py.
9. Finalizer.py
   1. This is it, the moment we have all been waiting for. This takes the data in document\_data.csv and uses it to generate a bunch of files with the correct names.
   2. INPUT
      1. document\_data.csv – This document contains the path to the pdf, and all the data corresponding to that pdf. Its all you need to recreate the pdf with the correct name
   3. OUTPUT
      1. Labeled folder – Final, correctly named documents are sent to this folder.
10. Labeled Folder
    1. Contains the final, labeled documents
    2. INPUT
       1. Finalizer.py
    3. OUTPUT
       1. Wherever your heart desires. Files are complete.
11. RunCNN.ipynb
    1. This is an automated pair of eyes that is completely optional to use. It adds a bit of complexity to the process, so only use it if you feel comfortable. Based on how a document looks, it categorizes them as either “GC”, “Backpage” or leaves it be. It also guesses if pages are blank that were not caught as blank by the Scan Snap machine
    2. INPUT
       1. document\_data.csv – Only run this program after running InitalDataPreparation.py
    3. OUTPUT
       1. document\_data.csv – Applies its changes to this file.
12. DateConversion.py
    1. This script is not used by the user. This is a script that is used by other scripts. It provides very helpful support for handling multiple date formats. The program provides liberal support for dates because of this file.

**Unused tools and experiments**

I have intended for this document to be an instructible for other people to use in the future. However, it also serves as documentation for what I accomplished during my tenure here. In that case, here are some technologies that I had created here but decided not to include within the technology that future people would use. The reasons for this judgement vary, but one of the main reasons in that I judged the usage of these tools to be overly complicated and would not provide enough benefit to make up for the confusion they would cause. This document is already obstruse as it is, and I don’t want to bloat it even more. In addition to that, some of these tools are technically interesting and were the results of experiments that are valuable to me, but in practical usage they may not be worth the effort it takes to learn them.

With that in mind, I do remain proud of the work done here. These are complex programs that provided me with valuable technical experience that I hope to take forward with me into my future career.

* Preguesser.py – I am very proud of this program. It functions as both an image processor and an OCR program. It uses image processing to isolate and clarify sections of the document, removing any unwanted noise. After it has done so, it uses an OCR program to read the clarified information and apply it to the document\_data.csv. In essence, it is an automated pair of eyes and hands. It was the result of research and numerous experiments to fine tune it. I think it is one of my most technically impressive accomplishments, and I’m glad I could put this on my resume. It was useful to me when I was working with it and saved me time. However, it is a bit complicated to use, and takes a great deal of time to run, and makes mistakes that I believe others may find confusing. For others, I don’t think the confusion is worth the possible benefits. I’ve left it out to streamline the workflow.
* Dataset scripts – There are series of scripts under the name Dataset.py. These were valuable developer tools that I used to create a series of Datasets for machine learning and data analysis based on archived data and documents. These were very helpful in the creation of tools and running of experiments. However, they are not relevant to daily document processing.
  + One particularly interesting dataset script is the Text Dataset script. This was a script to create textual dataset for experimenting and training OCR programs/Text-Based Neural Networks. Since the OCR program outputted locational data in addition to the predicted text, I could very effectively isolate the stamp. I created a significantly large dataset of Stamp specific text, which I consider to a valuable experiment in data gathering. I used this to experiment with fine tuning the OCR model I was using.
* Utility Scripts – I wrote some additional scripts to collect aggregate information about how may files I scanned. These are not too important.
* RunCNN.ipynb – Extremely useful tool that I’ve been using. It uses a convolutional neural network to categorize documents. It would load three, increasingly specific models from the disk and use it to predict on new data. It was very useful to me. Unfortunately, for technical reasons, it cannot be moved from one system to another without hassle, which is why I’ve chosen to exclude it.
* Training Scripts – Here are some of the notebooks that I used to train some Neural Networks on the data collected by some of the Dataset scripts. These are my own CNN models designed for document recognition.
  + There were three main models constructed from this.
    - One of the neural network models was a big success and is used by the RunCNN script documented above. It was very accurate in predicting GC and backpage documents.
    - The other was mildly useful. I didn’t think it provided enough benefits to justify learning it, so I left it out. It attempted to differentiate Initial, Renewal and Update documents. It was very good in predicting Initial documents from 2003, Update Affidavits, and Update forms with yellow highlight, but other than that it wasn’t very good.
    - The other was an attempt to differentiate Updates from Renewals. It failed and was completely inaccurate. The details were to fine to differentiate Renewals from Updates.
  + These models were all saved to disk. The useful one that I’ve left for users is saved as DocCNN.keras.
  + The RunCNN model notebook was designed to load and run these models. The provided RunCNN notebook is cut down from the complete notebook, which was designed to run each of these models in succession. As you can tell, each model attempts to get more and more specific with the documents.
* Testing Scripts – Some very, messy, unkept notebooks that I used to help myself learn and test the skills needed to build the software that I did. I won’t go into detail with these, as it is much too messy, but if you look closely, you can see the prototypes of functions that would later evolve into the core of the Viewer and PreGuesser programs.
  + The most notable experiments/tests were image processing ones, designed to isolate stamps and remove noise. They were very successful and managed to make it into the useful PreGuesser program.
* Old Workflow – Some prototypical tech that would later evolve into the scripts we all love today. A lot of it was reliant on Excel and spreadsheets. The main difference between this workflow and the new one is that the new one utilizes a custom GUI. This previous one relied on entries in Excel. It was a lot slower. If you look closely, you can see some old workflow scripts that have direct descendants in the currently utilized scripts. This folder is kept for archival purposes and is not used.