[Insert Sheet.ipynb](https://colab.research.google.com/drive/1GNZ6cTdXmtqWWu0XZNyW51W7azFsBVKz?usp=sharing)

* This program converts selected spreadsheet to dataframe and loops through the [Manual Indexing](https://drive.google.com/drive/folders/1NJIZFtBPyqF1q6GxPL3rNPadsobSryDD?usp=drive_link) folder placing the selected spreadsheet within each workbook via the Google auth and gspread packages
* Current Runtime: 15+ minutes(170+ loans)

[Master Merge.ipynb](https://colab.research.google.com/drive/1uSq9lziHU-WPqyX8Gk8iqSJ7i520HZ_y)

* This program loops through the [Manual Indexing](https://drive.google.com/drive/folders/1NJIZFtBPyqF1q6GxPL3rNPadsobSryDD?usp=drive_link) folder via the Google auth and gspread packages
* It concatenates each loan data frame and outputs one merged CSV
* Current Runtime: 4 minutes(143 loans = 80,000+ rows)

[Package Pages.py](https://drive.google.com/file/d/1AnhpuINkk32YKl5IWKAFuJii7CiTYarh/view?usp=sharing)

* This program loops through a directory to find the amount of pages for each PDF within the directory
* It adds the number of pages to a dictionary. The file name is the key, and the number of pages is the value.
* This dictionary is then converted to a data frame and outputted to a CSV[(“Total.csv”)](https://docs.google.com/spreadsheets/u/0/d/1AOcfPRAno0a6W62fp1NtWxOM-FlCHaBI6Oo_Wyjp6hY/edit)
* IDP use case: This program was created to identify the length of each IDP-indexed package.
* Current Runtime: 4-5 minutes(148 files)

[Page Compare.py](https://drive.google.com/file/d/1MbmFMFQ75BJbrSU2jZE_cBK_djrs0SL4/view?usp=sharing)

* This program compares the pages of two PDFs.
* It has a method get\_page that takes in two inputs. It then iterates through the two documents and finds which pages are equivalent.
  + Inputs: f1 and f2
  + Output: pages(a data frame with the matching page assignments)
* The data frame is outputted to a CSV[(“page compare.csv”)](https://docs.google.com/spreadsheets/u/0/d/154aOP3Fr1M7z3eqW5Yg6Mn6xEgDV6rS3vVucjqSieEg/edit)
* IDP use case: This program was created to identify the pages that IDP had reorganized
* Current Runtime: 5-6 minutes(comparing two files)

Pages Advanced.py(Titling not perfected yet)

* This program gets the believed title for each page of a pdf and finds the matching page in the second pdf that corresponds to the first pdf
* It has a method get\_page that takes in two inputs. It then iterates through the two documents and finds the page number and believed title for each page and which pages are equivalent for pdf1.
  + Inputs: f1 and f2
  + Output: df(A data frame with the columns: “File1: Title”, “File1: Page”, “File2: Title”, “File2: Page”, and “File1 Matching Page”)
* The method utilizes multiple concepts to obtain a title. As it iterates throught the text of the document, it utilizes blacklisting, whitelisting and translation. If no matching title is found, it will return the first line(that does not contain punctuation or numbers, this is in an effort to avoid displaying loan numbers or any sensitive data contained within the document) of the pdf if available
* The Datframe is then exported as a CSV(['Pages Advanced.csv'](https://docs.google.com/spreadsheets/d/1uw_6X3qfyofk6DzUM3oIj_f--uLGeS_Ee1VfbD0Kvpw/edit?usp=sharing))
* IDP use case: This program was created to identify the pages that had been reorganized
* Current Runtime: 6-7 minutes(comparing two files)

[Bookmark Page.py](https://drive.google.com/file/d/1o3LQKAkCB68iwXZ6z29EGuhIo5wvccOx/view?usp=sharing)

* This program loops through a directory to find the bookmark and corresponding page number for each pdf
* It has a method show\_tree which takes in a list of bookmarks and adds each bookmark with its corresponding page number into a dictionary
  + Inputs: bookmark list
  + Output: None
* The output of show\_tree is then placed into a dictionary with the key being the file name and the value being the outputted dictionary from show\_tree
* The nested dictionary is then converted to a data frame outputted to a CSV[(“bookmarks.csv”)](https://docs.google.com/spreadsheets/d/1JEMQDh7x_9dt_Mkq7MUHK7uTCpwjKG_uSpiyS085YA0/edit?usp=sharing)
* Current Runtime: 5-6 minutes (148 files)

[Search Algorithm.ipynb](https://drive.google.com/file/d/1Dd1nJJkrbu307TYxpPOSqzMVx8JR73BC/view?usp=drive_link)

* This is a Jupyter Notebook used to find which PDFs have a specified bookmark and the corresponding page number of the bookmark
* It has a method show\_tree which takes in a list of bookmarks and adds each bookmark with its corresponding page number into a dictionary
  + Inputs: bookmark list
  + Output: None
* The program then loops through the nested dictionary to find every document with the desired bookmark and returns the file name and page
* Current Runtime: 5-7(148 FIles)