Explorations in Economic History 87 (2023) 101469

Contents lists available at ScienceDirect

Explorations in Economic History

journal homepage: www.elsevier.com/locate/eeh

Research Paper

Digitization and data frames for card index records☆

Someswar Amujala[a], Angela Vossmeyer[a,b,*], Sanjiv R. Das[c]

[a] Claremont McKenna College, USA
[b] NBER, USA
[c] Santa Clara University, USA

ARTICLE INFO

JEL classification:
C8
N4

Keywords:
Machine learning
Natural-language processing
Archival records
Unstructured data

ABSTRACT

We develop a methodology for converting card index archival records into usable data frames for statistical and textual analyses. Leveraging machine learning and natural-language processing tools from Amazon Web Services (AWS), we overcome hurdles associated with character recognition, inconsistent data reporting, column misalignment, and irregular naming. In this article, we detail the step-by-step conversion process and discuss remedies for common problems and edge cases, using historical records from the Reconstruction Finance Corporation.

1. Introduction

Card indexes (example in Fig. 1) are a common form of archival records. They often summarize the content of more detailed files, aid in the location of records, or aggregate pertinent information from individual, micro-unit records. Economic historians have employed card index records for the study of patented inventions (Griffiths et al., 1992), manufacturing firms (Yates, 1989), banking and industrial history (Perlinge, 2004), and government recapitalization programs (Vossmeyer, 2016). While index cards are often an efficient way to present information, their automated use in academic research can be challenging due to imperfect character recognition, inconsistencies in data reporting, irregular naming, and column misalignment. These challenges render many tab or character delimitation and regular expression tools inapplicable, leaving researchers with the painstaking task of manually converting this information into data frames.

In this article, we develop a streamlined methodology for the extraction of data from card index records, and more broadly, unstructured records, addressing a data preparation problem facing many economic historians. Our methods leverage readily available machine learning and natural-language processing tools through the cloud infrastructure of Amazon Web Services (AWS) and customize new tools to efficiently convert PDF images of card index records to usable data frames for statistical and textual analyses. To demonstrate our approach, we apply our methodology to records from the National Archives on the Reconstruction Finance Corporation's (RFC) Card Index To Loans Made To Banks And Railroads, 1932-57. We approximate that our machine learning approach saved about 820 hours (34 days) of manual data entry time for these records. The methods highlighted in this paper can be easily employed for most card index records and adapted for other unstructured archival records.

We detail our RFC records and provide step-by-step instructions to our methodology in Section 2. Section 3 details performance metrics, discusses the various challenges that one can face in digitizing and categorizing these data, and offers remedies for common

☆ The Python coding files associated with this work can be found at https://github.com/somu-a/record-digitization.

* Corresponding author.
E-mail addresses: samujala23@cmc.edu (S. Amujala), angela.vossmeyer@cmc.edu (A. Vossmeyer), srdas@scu.edu (S.R. Das).

https://doi.org/10.1016/j.eeh.2022.101469
Received 23 March 2022; Received in revised form 29 June 2022; Accepted 8 July 2022
Available online 15 July 2022

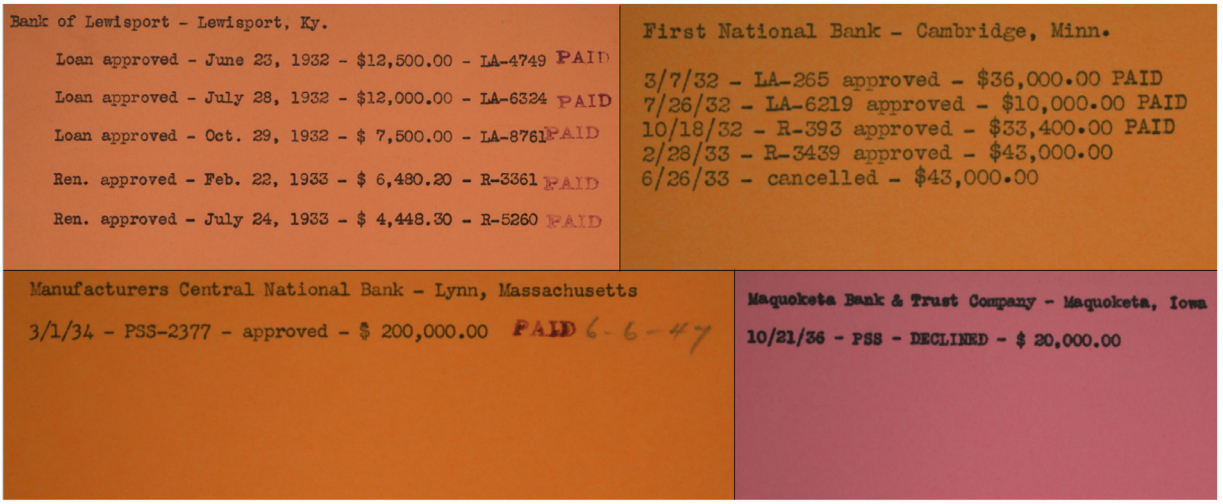


Fig. 1. Images of four example RFC card files.

problems. In Section 4, we review how to implement the methodology with our publicly-available code. Finally, Section 5 provides concluding remarks and offers additional applications in which economic historians can employ this methodology.

2. Methodology

2.1. Example application

Fig. 1 presents several images from the RFC Card Index. The RFC was a government-sponsored bank rescue program established in 1932 under President Hoover. Banks could apply to the RFC for a collateralized loan or recapitalization.[1] The RFC reviewed the submitted application, assessing the quality of the bank's assets and management and the importance of that bank to the local area. The RFC then approved or declined the bank's application and determined the appropriate loan amount. The National Archives stores all of the approved and declined loan files from the RFC, but an individual bank's file can be several hundred pages. The card index, on the other hand, summarizes the information in the application files for each bank and its loan requests. These cards detail the bank's name, location, loan dates, loan types, loan decisions, loan amounts, and whether or not the loans were paid. Generally, we observe one card per bank, with all of its loans listed, but occasionally, we observe several cards for a single bank.

Fig. 1 demonstrates the challenges that arise with these records. First, there are various fonts including typewriter text, "PAID" stamps, and even some handwriting; all of which needs to be correctly processed using optical character recognition. Second, the date is presented in many formats and in several different locations. The same is true for the loan type and loan decision. Loan type is characterized by various abbreviations. For example, "R" or "Ren" represent renewals, "PSS" represents preferred stock subscriptions, and "LA" represents collateralized loan approvals. Loan decision includes words, such as "approved," "cancelled," and "declined." Finally, the ordering of the information varies from card to card. Because these data are unstructured, we exploit natural-language processing tools based on machine learning to categorize the text into relevant entities.

Manually entering this information into data frames is possible, but horribly time consuming. The RFC alone has several card indexes, reaching over 200 linear feet of archival space. This card index to banks and railroads is 27 linear feet of archival space, equating to 22,436 unique index card records. Therefore, we develop a streamlined and efficient software pipeline for character recognition and categorization.

2.2. Pipeline overview

We employ two artificial intelligence (AI) services from AWS to convert PDF images of our index card records to data frames. The first is Textract. Textract is a machine learning service that automatically extracts text, handwriting, and data from scanned documents. Textract advances the usual optical character recognition (OCR) by implementing artificial intelligence to extract text, layouts, forms, and tables without configuration, training, or custom code.[2] The second service is Comprehend. Comprehend is a

[1] Hereafter, we use "loan" to characterize collateralized loans or recapitalization for brevity.
[2] https://aws.amazon.com/textract/

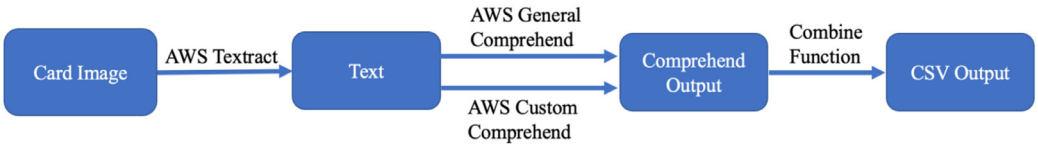


Fig. 2. Graphical depiction of the workflow.

Bank Name	Location	Date	Abbrev	Decision	Amount	Paid	Line
Bank of Lewisport	Lewisport, Ky	June 23, 1932	LA-4749	Loan approved	$12,500.00	PAID	Loan approved - June 23, 1932 - $12,500.00 - LA-4749 PAID
Bank of Lewisport	Lewisport, Ky	July 28, 1932	LA-6324	Loan approved	$12,000.00	PAID	Loan approved - July 28, 1932 - $12,000.00 - LA-6324 PAID
Bank of Lewisport	Lewisport, Ky	Oct. 29, 1932	LA-8761	Loan approved	$7,500.00	PAID	Loan approved - Oct. 29, 1932 - $7,500.00 - LA-8761PAID
Bank of Lewisport	Lewisport, Ky	Feb. 22, 1933	R-3361	Ren. approved	$6,480.20	PAID	Ren. approved - Feb. 22, 1933 - $6,480.20 - R-3361 PAID
Bank of Lewisport	Lewisport, Ky	July 24, 1933	R-5260	Ren. approved	$4,448.30	PAID	Ren. approved - July 24, 1933 - $ 4,448.30 - R-5260 PAID
First National Bank	Cambridge, Minn.	3/7/32	LA-265	approved	$36,000.00	PAID	3/7/32 - LA-265 approved - $36,000.00 PAID
First National Bank	Cambridge, Minn.	7/26/32	LA-6219	approved	$10,000.00	PAID	7/26/32 - LA-6219 approved - $10,000.00 PAID
First National Bank	Cambridge, Minn.	10/18/32	R-393	approved	$33,400.00	PAID	10/18/32 - R-393 approved - $33,400.00 PAID
First National Bank	Cambridge, Minn.	2/28/33	R-3439	approved	$43,000.00		2/28/33 - R-3439 approved - $43,000.00
First National Bank	Cambridge, Minn.	6/26/33		cancelled	$43,000.00		6/26/33 - cancelled - $43,000.00
Maquoketa Bank & Trust Company	Maquoketa, Iowa	10/21/36		DECLINED	$20,000.00		10/21/36 - PSS - DECLINED - $20,000.00
Manufacturers Central National Bank	Lynn, Massachusetts	3/1/34	PSS-2377	approved	$200,000.00	PAID	3/1/34 - PSS-2377 - approved - $200,000.00 PAID 6-6-47

Fig. 3. Output file of digitized and categorized data for our four example RFC cards.

natural-language processing (NLP) service that uses machine learning to uncover information in unstructured data. Comprehend can be trained such that the classification is tailored to particular archival records.[3]

These services have several advanced features that are appealing for an application such as the one in this paper. First, data extracted from Textract is returned with its bounding box (a polygon frame that surrounds the data) and coordinates, which identify where on the page or source document the extracted text is located. Layout can be an important aspect of document AI (Cui et al., 2021). Second, Textract is trained to work well with mixed type text and handwritten text. It also performs well on mixed digital and scanned PDFs (Gralinski et al., 2020). Third, common OCR utilities (e.g., Adobe Acrobat) cannot be used programmatically from Jupyter notebooks with languages like Python. We require programmatic processing because of the vast number of documents that we pipeline with Comprehend. Fourth, Comprehend offers a general model that has been trained on a multitude of text and business documents, making it well-suited for our application. Lastly, Comprehend offers custom entity recognition to identify terms that are specific to a domain. The custom Comprehend approach can be trained through a convenient and manageable user interface, requiring no knowledge of machine learning techniques. Because we wish to combine OCR with entity recognition, and do this programmatically, we are able to build a single pipeline on the AWS cloud using a common set of application programming interfaces (APIs) and extensible data storage. If needed, the pipeline can also be improved with Augmented AI, exploiting Textract's human-in-the-loop setup. The disadvantage of using a combined pipeline is that it locks one into AWS, especially since interoperability across cloud providers is often difficult to implement.

Fig. 2 graphically demonstrates our workflow. The workflow begins by inputting PDF images of the card index to AWS Textract for text extraction. Textract outputs flat text files, which serve as the input for Comprehend. Comprehend then categorizes the features within the text. We employ two models for Comprehend. The first is a prebuilt, general AWS model that is able to categorize certain entities on the cards. The second is a custom model that we train to capture specifics of the RFC program and edge cases. We then combine the outputs from the two models into a CSV (comma-separated values) file.

Fig. 3 presents the raw output file that is associated with the cards pictured in Fig. 1. In our application, we focus on extracting seven variables (i.e., features) from the cards, which are represented by the first seven columns of Fig. 3. Nearly the entire process—going from card images to our seven variables—is automated through AWS APIs, allowing asynchronous processing in the cloud and easing any computational burden on personal devices.

2.3. Step-by-step guide

Here, we review the step-by-step process of converting the RFC files into data frames and discuss when training and customization are necessary. We also generalize the approach in each step to illustrate how these methods can be applied to other records.

Step 1 – File upload and Textract:

The process begins by creating an AWS account, saving the user key, and uploading the PDF files of the card records to an S3 bucket in AWS. The user information and file location are preliminary inputs for this process.

[3] https://aws.amazon.com/comprehend/

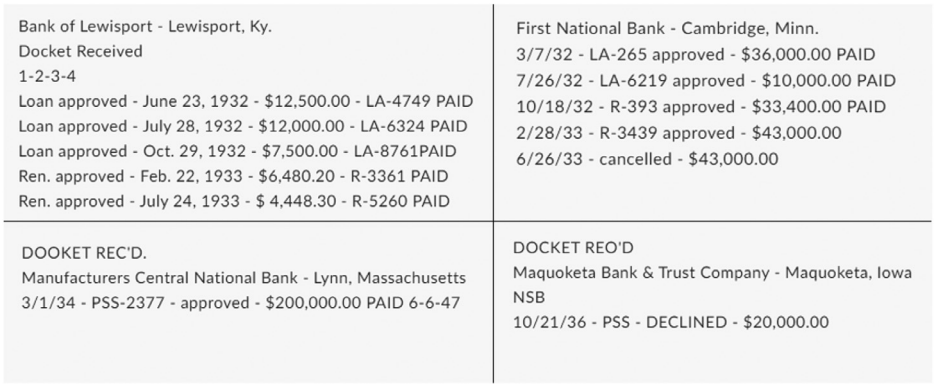


Fig. 4. Images of the Textract output files for our four example RFC cards.

```
In [8]: initial_comprehend('Output2.txt')

Out[8]: {'0': {'ORGANIZATION': 'Bank of Lewisport', 'LOCATION': 'Lewisport, Ky'},
         '1': {},
         '2': {},
         '3': {'DATE': 'June 23, 1932', 'QUANTITY': '$12,500.00', 'OTHER': '4749'},
         '4': {'DATE': 'July 28, 1932', 'QUANTITY': '$12,000.00', 'OTHER': '6324'},
         '5': {'DATE': 'Oct. 29, 1932', 'QUANTITY': '$7,500.00', 'OTHER': '8761PAID'},
         '6': {'PERSON': 'Ren',
          'DATE': 'Feb. 22, 1933',
          'QUANTITY': '$6,480.20',
          'OTHER': 'R-3361'},
         '7': {'PERSON': 'Ren',
          'DATE': 'July 24, 1933',
          'QUANTITY': '$ 4,448.30',
          'OTHER': '5260'}}

In [10]: initial_comprehend('Output17.txt')

Out[10]: {'0': {'ORGANIZATION': 'First National Bank', 'LOCATION': 'Cambridge, Minn.'},
          '1': {'DATE': '3/7/32', 'COMMERCIAL_ITEM': '-265', 'QUANTITY': '$36,000.00'},
          '2': {'DATE': '7/26/32', 'OTHER': '-6219', 'QUANTITY': '$10,000.00'},
          '3': {'DATE': '10/18/32', 'OTHER': 'R-393', 'QUANTITY': '$33,400.00'},
          '4': {'DATE': '2/28/33', 'OTHER': 'R-3439', 'QUANTITY': '$43,000.00'},
          '5': {'DATE': '6/26/33', 'QUANTITY': '$43,000.00'}}
```

Fig. 5. Images of the JSON output for Comprehend's general model.

As Fig. 2 demonstrates, the card images are initially processed by Textract. In our file, each page of the PDF is a new card. Textract does not require any training or customization. Once Textract processes the file, it outputs a text file (.txt) for each page of the PDF. Fig. 4 provides examples of the text file outputs that are associated with the four cards in Fig. 1. Notably, Textract recognizes text, stamps, and handwriting. Textract also extracts the layout of the card, which can be useful for records that contain tables or structured forms. The figure also displays some odd quantities, e.g., "Docket Received" and the like. These are faint stamps that appear in the corner of many cards, which were omitted from Fig. 1 to zoom in the cards. Our subsequent Comprehend-based process removes these unnecessary quantities.

Step 2 – Comprehend:

The plain text files that Textract produces are the inputs for Comprehend. Comprehend uses natural-language processing tools to designate words into certain entity categories. There is a general Comprehend model that can be used to classify the text or a custom model that can be built using a training dataset. For the RFC data, we employ both general and custom models.

(a) *General Model* – The general model that is built into Comprehend is able to categorize common expressions. For instance, in our RFC example, the built-in function often successfully recognizes the bank name ("organization"), bank location ("location"), loan amounts ("quantity"), and dates ("dates") without a training sample. However, the built-in function is often unsuccessful at recognizing the abbreviations and decisions, motivating us to create a custom model as well. Because of the general model's success in categorizing four of our seven variables, we proceed to process our 22,436 records. Comprehend outputs a single text file in JSON format for the whole run. The text file identifies and labels each recognized word, and it provides each word's location based on an index number for the card (begin offset and end offset). Fig. 5 provides an example of the Comprehend

Text	Type
Loan approved	Decision
Ren. approved	Decision
Originally approved	Decision
approved	Decision
Railroad Loan - DECLINED	Decision
rescinds	Decision
B & L Loan approved	Decision
INCREASED	Decision

(a) Training Decision

File	Line	Begin Offset	End Offset	Type
Output38.txt	1	0	22	BANK
Output38.txt	1	25	47	LOCATION
Output10.txt	0	0	32	BANK
Output10.txt	0	35	59	LOCATION
Output583.txt	0	0	16	BANK
Output583.txt	1	0	24	LOCATION
Output597.txt	0	0	32	BANK
Output597.txt	0	35	62	LOCATION

(b) Training Bank Name and Location

Fig. 6. Images of our labeled training sample CSV files.

output for the top two cards presented in Fig. 1. We've suppressed the indexing and performance metrics in the output to make it more visual.

As the figure shows, each item on the card is labeled with the category to which Comprehend believes it belongs. The categories of organization, date, location, and quantity appear to be consistent. However, the remaining items are often mislabeled (e.g., "Person" and "Commercial_item") or grouped into the "Other" category. Viewing the Comprehend output is a convenient way to determine how well the general model is categorizing the text. The Comprehend output file contains all of the data, labels, locations, and confidence scores for each cell. We save and store the output file, which is later combined with the custom model output.

(b) *Custom Model* – The variables that the general model has difficulty categorizing are often nuances that are specific to the RFC program. For instance, the abbreviations represent the various funding programs offered by the RFC (e.g., collateralized loans or preferred stock subscriptions). Similarly, the decision variable is specific to the RFC program's structure and processes. The general model also has difficulty categorizing when the text for an entity varies quite dramatically from document to document, such as the names of banks and locations. While the general model did a decent job categorizing these entities, we occasionally experienced errors with unusual names and locations. As a result, we develop custom models to identify and categorize these features.

Each custom model needs to be trained and AWS recommends a minimum of 200 examples for training. Therefore, we provide Comprehend with the Textract text files associated with at least 200 cards and a perfectly labeled CSV file. There are two ways in which a custom Comprehend model can be trained. The first is based on unique text. The decision variable is generally defined by a handful of different words, as well as some slight variations. Therefore, as shown in Panel (a) of Fig. 6, the simplest way to train the model for this variable is to provide a CSV file with a sample of text and the desired entity label. The second way to train a custom model is based on the location of the text in the document. Bank name and location consistently appear at the top of each card and remain in the same order (name then location). In this case, it is more accurate to train the custom model with an annotated CSV file that references a specific text file. For each reference, the researcher must provide the entity label, line number, beginning character offset, and ending character offset to locate each item, as shown in Panel (b) of Fig. 6. Creating this training sample file is the most laborious part of our process. However, we do not recommend using the method in Panel (a) for bank and location because there are complex variations in the names. Additionally, we do not recommend using the method in Panel (b) for the decision variable because that text can appear in many different locations on the document. Selecting the right training method leads to a more accurate model.

After submitting the training samples, Comprehend provides several performance metrics for each custom model. If the scores are lower than expected, researchers should double-check the labeled data in the training sample, increase the training sample size, or explore potential edge cases. Once the models are adequately trained, we process the whole sample of 22,436 text files. As with the output of the general model (Fig. 5), the custom Comprehend model outputs a single text file in JSON format with all of the data, labels, index values, and confidence scores for each cell. We review our performance metrics in Section 3.1.

Step 3 – Combine function:

Because we employ general and custom Comprehend models, we have multiple text files for our sample. Therefore, we write a Python function that combines the two output files together, creating one single entry line for each loan. In this step, we also input functions to categorize our paid and abbreviation variables, which are efficiently captured using regular expression tools.[4] Additional hard-coded functions can be inserted to catch common errors, inconsistent reporting, or other bugs to ensure a cleaner final output file. We discuss some of these functions and potential remedies in Section 3.2. Our final output is a single CSV file with all of the observations and columns of variables, as depicted in Fig. 3.

[4] https://en.wikipedia.org/wiki/Regular_expression

Table 1
Computation Information. The computational times are reported from the full run of 22,436 PDF images.

Function	Run Time	Scores
Textract	5 Hours	Average Confidence of 96.95%
Comprehend-General	7.5 Hours	Average Confidence of 90.83%
Comprehend-Custom	3 Hours	Average Confidence of 90.60%
Training	15 Minutes	Precision of 94.56% Recall of 98.86% F1 of 96.66%
Combine	4 Hours	.

Table 2
Word Replacements. Hard-coded corrections to the text. The percentage of times the intended word was incorrectly recognized by Textract is shown in the last column.

Intended word	Error word	Total occurrences	Incorrect occurrences	Percent
Bank	Banic	16,104	177	1.10%
Bank	Banle	16,104	218	1.35%
Paid	Pald	28,601	570	1.99%

3. Results

3.1. Performance metrics

For our RFC application, we process 22,436 index card records (over 20 GB of PDF files). We note that our implementation is mostly inferential, i.e., other than the custom training in Comprehend, all other functionality directly uses already trained models in Textract and Comprehend, without any additional supervised learning. Therefore, the usual training metrics from a supervised learning model are nonexistent. However, AWS provides some metrics at the inference level that we are able to report. Table 1 presents the approximate run time for each step of our process and performance metrics that AWS provides. These performance metrics are not realized accuracy scores, but instead metrics about how the machine learning tools performed.

Textract reports a confidence score for each line of text in the sample. Low confidence scores are useful to flag observations that would benefit from checking the Textract output against the original files. Similarly, Comprehend provides a confidence score for each entity that it classified. Lastly, when training the custom Comprehend model, researchers are provided scores for precision, recall, and F1 – the standard metrics for supervised learning. These represent the performance of the model during training; however, they are only an approximation of the API performance during entity discovery. We recommend using Comprehend's inference confidence scores as a means to flag observations that require another look, checking the out-of-sample performance of the custom Comprehend model. We are confident that the custom Comprehend model is not overfitted as the average confidence score is almost the same ($\sim$ 90%) as that of the general Comprehend model, even in the presence of very high training metrics (F1 of 96.66%).

The run times for each step are moderate. Nearly the entire process is automated and asynchronously processed in the cloud. Other than uploading, downloading, and running the Combine function on the Comprehend output, the computation time is not a function of personal computers. While Textract must run before Comprehend, the general and custom Comprehend models can run simultaneously to save computing time. We approximate that our machine learning approach saved about 820 hours (34 days) of manual data entry time. The cost of using these AWS services is low. Researchers pay per page processed. The total cost of our full run was about $300.

3.2. Special cases and assessment

In this section, we review common errors and edge cases that arose in our application and discuss how we flagged observations or remedied common problems. We also assess how well the OCR method represents the original files by presenting an accuracy rate accounting for erroneous insertions, deletions, and substitutions.

1. Due to smudged ink on the cards, Textract occasionally reads "Bank" as "Banic" or "Banle" and "Paid" as "Pald".
   Solution: We hard-coded functions to correct these issues and leveraged Python regular expressions to identify other potential patterns. Table 2 reports the number of times our hard-coded fix corrected these words. About 2.45% of instances of the intended word "Bank" were erroneously digitized as "Banic" or "Banle" and 1.99% of "Paid" instances were digitized as "Pald". We are confident that there weren't any erroneous changes, but caution the reader to only change nonsensical words.
2. If the text in the original file is slightly misaligned (e.g., the PAID stamp is slightly below the loan line), Textract reads it as a new line, creating two rows for the observation.

Table 3
Textract Performance. Errors in Textract's output from a random sample of 100 cards.

Error Type	All Cards	Bank Cards	Non-bank Cards
Number of Words	1834	1397	437
Avg. # Words Per Card	18.43	18.14	19.00
Deletions	22	20	2
Insertions	3	3	0
Substitutions	42	33	9
Error Rate	3.65%	4.01%	2.52%

Solution: We searched the output file for observations that had many empty cells and merged adjacent rows (assuming they had the appropriate empty cells). If there was misalignment with the relevant empty cells, we flagged the observation to be checked.

3. Comprehend's general model occasionally miscategorized complex bank names or locations. To be more confident in the labeling of these variables, we trained a custom Comprehend model on bank name and location.

   Solution: In the Combine function, where we merge the output of the two models, we created indicator variables that equal 1 if the general and custom models agree on bank name and location. Doing so increases our confidence in Comprehend's categorization of these variables.

4. The RFC cards also report loans granted to counties across the United States. In those cases, Comprehend often miscategorized the loan recipient as the location.

   Solution: We hard-coded a function that creates an indicator if the word "county" appears in the records. When we observe a one in that column, the observation is flagged so the location and organization classification can be checked for accuracy.

5. In reviewing the final CSV, it is often helpful to observe the Textract line that was used to classify each variable. Furthermore, one does not want to omit any unused text in the event that more variables need to be created later.

   Solution: The last column of our CSV file is the raw text that was used to create each variable, as shown in Fig. 3. The paid date is not always reported, so we currently do not create a variable for that. In the future, if we desire that information, it can be created from the raw text column.

We were quite liberal with the "flags" we put in the code to indicate observations that needed additional checking. In our assessment of the final CSV output file, we flagged about 20% of our observations. Of that 20%, 5% did not have any issues; the text recognition and categorization were accurate. For the other 15%, small mistakes were found, including misspellings of bank names or locations, failures to classify dates, names, or locations due to odd spacing or naming, or loan information spanned multiple rows due to misaligned text. Some of these failures, especially those due to misspellings, resulted from errors in Textract rather than Comprehend. These errors are more difficult to resolve as Textract is less customizable than Comprehend. In this process, we manually corrected a few thousand entries, whereas the remaining were quite accurate.

Solutions to classification issues in Comprehend are quite specific to the RFC application; however, character recognition issues with Textract are generalizable to most index card applications (particularly those with typewriter text and handwriting). Therefore, in order to provide a useful assessment, we randomly sampled 100 RFC index cards and compared the Textract output to the original card. In assessing Textract's performance, we report the number of erroneous deletions, insertions, and substitutions in Table 3. Table 3 also provides these statistics based on the subgroups of bank cards and non-bank cards. Non-bank cards, which include loans to counties, railroads, and building and loan associations, are rare and are generally simpler than bank cards. Across the 100 cards, there were 1834 words to be recognized. A total of 67 errors occurred, equating to a 96.35% accuracy rate (or 3.65% error rate). Most of the errors were substitutions, where Textract misread bank or location names. Substitution errors were almost never in numerical characters. The substitution errors reported in Table 3 do not include our "Bank" and "Paid" remedies presented in Table 2. Thus, substitution errors can be improved greatly if common issues are identified. Insertions were very rare and only occurred when there was smudged ink on the cards. Deletions primarily occurred when the "PAID" stamp overlapped typewriter text. Textract's overall performance is promising. Interestingly, the accuracy rate we report for these 100 cards aligns closely with the average confidence score reported in Table 1.

4. Implementation

Our Python coding files are publicly available at https://github.com/somu-a/record-digitization. This GitHub repository also includes detailed instructions, a sample of card files that can be processed with the code, the training sample files that are needed for the custom Comprehend models, and a final CSV output associated with the sample of cards. Because this is a smaller sample of cards, we omit some of the code that deals with the edge cases specific to the RFC application. We do this in order to keep the code more generalizable and easily transferable to other applications.

The main coding file is titled `AWS_Code.ipynb`.[5] Prior to running this file, users should set up an AWS account, upload the RFC application files to their S3 bucket, and authenticate use of AWS's Python SDK Boto3. Documentation for this step is linked in the `AWS_Code.ipynb` file. Once the AWS account steps are complete, the user should input the S3 bucket location and file details into the Jupyter notebook. These are clearly labeled with the word "INPUT." While the Jupyter notebook includes many functions, not all of them need to be called. Most of the functions are used as "helper" functions that a few "main" functions call. Thus, the user only has to run a few main functions, which are clearly labeled with "RUN" in the Jupyter notebook. While not all functions need to be run, they must all be "compiled" (i.e., read into memory) for the code to work properly.

To convert the card PDF file to text via Textract, run the `convert_card_image()` function. The text files for each card are then automatically uploaded to the user's S3 bucket. The next step is to train the custom Comprehend models, which requires training CSVs and training sample sets. The user should launch AWS Comprehend, go to custom entity recognition, and create a new model. Follow the subsequent on-screen steps and then input the resulting dARN and ARN from the models, as well as an S3 bucket location for your input and output, in the Jupyter notebook (these are labeled with the word "INPUT"). More details on how to complete these steps are located in the `Detailed_Instructions.pdf` in the GitHub repository.

After training, run the main function `start_all_comprehend()`, which executes the general and custom Comprehend models. Then, save the output files locally (in the same directory as the Jupyter notebook). These files are the inputs for the Combine step. Run the final step, the `get_csv_test()` function, which parses through the Comprehend output, executes the regular expression functions, and creates the desired CSV output. The final output file saves to the same directory as the Jupyter notebook.

Once the user successfully executes the code, their output file should look like the sample CSV output file provided in the GitHub repository. We note that AWS Textract and Comprehend are updated regularly, so there can be slight variation in how these files are processed. Further, supervised learning used for the custom Comprehend training may use stochastic gradient descent, leading to small differences in the final model as random seeds may differ, and updates to Python, numPy, sciPy, and other packages may generate variations based on differences in software versions.

5. Conclusion

In this paper, we offer a solution for converting analog card index records to digital data frames by employing AWS Textract and Comprehend services. These machine learning and natural-language processing tools not only digitize text, but also classify text into relevant entity categories. We tailor a workflow that is accurate, modular, and generalizable for the curation of datasets from card index records.

Many historical records are stored in the form of card indexes. In a search of the National Archives Records Administration, records from the Department of Defense, War Department, Department of the Navy, U.S. Bankruptcy Court, and many others include card indexes that summarize important information that can be employed in academic research. In the Appendix, we offer several examples of card images from these records, where we believe our approach would work well.

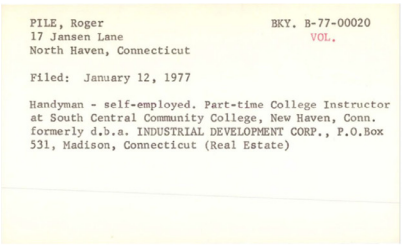
In addition to card index records, the methods highlighted in this paper can be easily adapted for other unstructured records, where difficulties include character recognition, inconsistent data reporting and formatting, and irregular naming. Our open-source code may be easily customized for other applications. Perusal of the source code in GitHub shows that very few changes are needed, barring edits to file names and the final block of code, in which the structure of the final output file is application mandated. Each different application may or may not need custom training in Comprehend. If it is needed, the user would have to prepare a small labeled dataset to undertake few-shot learning. While this would require data preparation, modifications to the provided code would be minimal.

Appendix A

In a search through the National Archives Records Administration, many record groups contain card indexes. Panels (a) and (b) of Fig. 7 display records from the U.S. Bankruptcy Courts. These cards contain information on individuals, locations, dates, and more for bankruptcy filings, which are presented in an unstructured manner. Data from these cards can be employed in time series or spatial studies of financial distress.

Panels (c) and (d) of Fig. 7 display records from the Department of Defense and War Department, respectively, representing promotions and qualifications of service members. Data from such records can be employed to study long-run effects of service or labor market outcomes. Researchers seeking to convert these records to data frames will find the AI methods outlined in this paper useful and efficient.

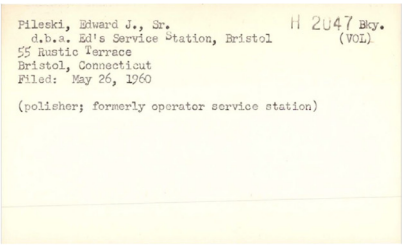
[5] Users should use the Jupyter Notebook IDE. Other IDEs, such as Google Colab, may also work but have not been fully tested.


PILE, Roger BKY. B-77-00020
17 Jansen Lane VOL.
North Haven, Connecticut

Filed: January 12, 1977

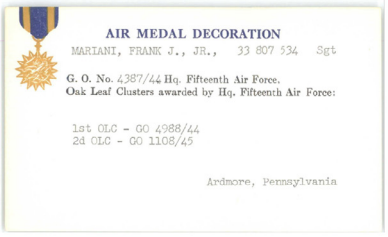
Handyman - self-employed. Part-time College Instructor at South Central Community College, New Haven, Conn. formerly d.b.a. INDUSTRIAL DEVELOPMENT CORP., P.O.Box 531, Madison, Connecticut (Real Estate)

(a) Record from the U.S. Bankruptcy Court.


Pileski, Edward J., Sr. H 2047 Bky.
d.b.a. Ed's Service Station, Bristol (VOL)
55 Rustic Terrace
Bristol, Connecticut
Filed: May 26, 1960

(polisher; formerly operator service station)

(b) Record from the U.S. Bankruptcy Court.

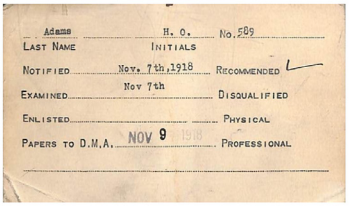

AIR MEDAL DECORATION
MARIANI, FRANK J., JR., 33 807 534 Sgt

G. O. No. 4387/44 Hq. Fifteenth Air Force.
Oak Leaf Clusters awarded by Hq. Fifteenth Air Force:

1st OLC - GO 4988/44
2d OLC - GO 1108/45

Ardmore, Pennsylvania

(c) Record from the Department of Defense: Air Force Award Cards.


Adams H. O. No. 509
LAST NAME INITIALS
NOTIFIED Nov. 7th, 1918 RECOMMENDED
Nov 7th
EXAMINED DISQUALIFIED
ENLISTED PHYSICAL
PAPERS TO D.M.A. NOV 9 1918 PROFESSIONAL

(d) Record from the War Department: Army Air Forces Qualifications.

Fig. 7. Example card index records from the National Archives that can employ the methods outlined in this paper.

References

Cui, L., Xu, Y., Lv, T., Wei, F., 2021. Document AI: benchmarks, models and applications. CoRR doi:10.48550/arXiv.2111.08609.

Gralinski, F., Stanislawek, T., Wróblewska, A., Lipinski, D., Kaliska, A., Rosalska, P., Topolski, B., Biecek, P., 2020. Kleister: a novel task for information extraction involving long documents with complex layout. CoRR doi:10.48550/arXiv.2003.02356.

Griffiths, T., Hunt, P.A., O'Brien, P.K., 1992. Inventive activity in the british textile industry, 1700–1800. J. Econ. Hist. 52 (4), 881–906.

Perlinge, A., 2004. The foundation for economic history research within banking and enterprise and the records of Stockholms Enskilda bank, 18561971. Financ. Hist. Rev. 11 (1), 105–120.

Vossmeyer, A., 2016. Sample selection and treatment effect estimation of lender of last resort policies. J. Bus. Econ. Stat. 34 (2), 197–212.

Yates, J., 1989. Information systems for handling manufacturing and marketing data in American firms, 1880–1920. Bus. Econ. Hist. 18 (2).