-

### 

UWM Production: Entity Extraction

Technical Design Document

Date:

Authors: Quantiphi

Prepared for: UWM

Document type: Technical Design Document

Table Of Content

[**1. Introduction**](#_kbkff0biatul) **5**

[**2. Requirements**](#_72lvqmjilzd) **5**

[2.1 2.1 Software requirements](#_25copoxlnfqb) 6

[9.1 Software Requirements  
 The programming language used is Python 3.7](#_t8aav3np5ax5) 6

[2.2 Google Services](#_kzrk20grfujo) 7

[2.2.1 Document AI](#_cevzf5dhogv) 7

[2.1 EE API](#_e104xevjx31a) 7

[Architecture Steps](#_vqnk5b326xc3) 9

[2.2 Data sources](#_r7fajh6kgkgb) 10

[**3. Input and Output**](#_tyjcwt) **10**

[3.1 Input PDF](#_v4rj1iyx94tv) 10

[3.1.1 Pre-Processing](#_t6vqo4gfyqri) 10

[3.1.2Extraction Using Form Parser](#_bfqdtayrsj) 10

[3.1.3 Post-Processing](#_tx0g2aphy3vb) 11

[3.1.3.1 Form 1120](#_2j71i8k0ms3f) 11

[3.1.3.2 Schedule K1- 1065](#_pvfn1c5ihgqp) 13

[3.1.3.3 Master insurance Policy](#_pvfn1c5ihgqp) 16

[3.1.3.4 Verification of Income](#_pvfn1c5ihgqp) 18

[3.1.3.5 IRS - W2](#_pvfn1c5ihgqp) 23

[3.1.3.6 Credit Supplements](#_pvfn1c5ihgqp) 24

[3.1.3.7 Gift Letter](#_pvfn1c5ihgqp) 26

[3.1.3.8 Form 8825](#_rmiv3ipptuxc) 27

[3.1.3.9 Property Insurance Invoice](#_tc4tb5xqwy1v) 32

[3.1.3.10 Payoff at Closing](#_qpvlynaioq93) 33

[3.1.3.11 Form 1120s](#_pvfn1c5ihgqp) 36

[3.1.3.12 Bank Statements](#_pvfn1c5ihgqp) 38

[3.1.3.13 Closing Protection Letter](#_pvfn1c5ihgqp) 39

[3.1.3.14 Paystub](#_chsday4kye9j) 41

[3.1.3.15 Form 1040](#_pvfn1c5ihgqp) 46

[3.1.3.16 Schedule E Form 1040](#_pvfn1c5ihgqp) 49

[3.1.3.17 Wire Instructions](#_pvfn1c5ihgqp) 55

[3.1.3.18 Mortgage Statements](#_pvfn1c5ihgqp) 56

[3.1.3.19 Form 1065](#_pvfn1c5ihgqp) 59

[3.1.3.20 Driver’s Licence](#_pvfn1c5ihgqp) 61

[3.1.3.21 IRS 1099-R](#_104js6prlih0) 62

[3.1.3.22 Form 4562](#_pvfn1c5ihgqp) 63

[3.1.3.23 Schedule G Form 1120](#_7zf7vuuipd8e) 64

[3.1.3.24 Flood Insurance Policy](#_pvfn1c5ihgqp) 66

[3.1.3.25 Property Insurance Policy(HO3/HO6)](#_pvfn1c5ihgqp) 69

[3.1.3.26 Schedule C Form 1040](#_pvfn1c5ihgqp) 72

[3.1.3.27 Verification Of Assets](#_pvfn1c5ihgqp) 75

[3.1.3.28 Social Security Award Letter](#_iyhmexmj942t) 77

[3.1.3.29 Tax Transcripts](#_pvfn1c5ihgqp) 78

[3.1.3.30 Verification Of Employment](#_pvfn1c5ihgqp) 82

[3.1.3.31 Residential Purchase Agreement](#_pvfn1c5ihgqp) 85

[3.1.3.32 Schedule K-1 Form 1120S:](#_pvfn1c5ihgqp) 89

3.1.3.33 Pension Award Letter 95

3.1.3.34 Schedule D Form 1040 96

[3.2 Pipeline Output](#_br2gqayvv4zx) 95

[3.2.1 Final JSON Format](#_dbq10vl6s5pw) 95

[**4. GitLab Repo**](#_atioxc6owvu9) **95**

[4.1 Code Structure](#_n4jl2xhp9tzr) 95

[4.1.1 Branching Strategies](#_u7skg77j7ds6) 95

[4.1.1.1 Master Branch](#_u7skg77j7ds6) 95

[4.1.1.2 Dev Branch](#_u7skg77j7ds6) 95

[4.1.1.3 QA Branch](#_u7skg77j7ds6) 95

[4.1.1.4 Feature Branch](#_u7skg77j7ds6) 95

[**5. Evaluation Scripts**](#_59bk72s8mxsx) **96**

[5.2 Overall Accuracy](#_3aly72fz7iwg) 98

[6.3 Accuracy per entity](#_xawlfqjfowde) 99

[6.4 Accuracy per document](#_c04nhm38yciq) 99

[**7. Latency Estimations**](#_6o5x51w476m5) **99**

[**9. Limitations**](#_cvnn0pug5mr0) **100**

[**10. Bug Fixes/Issues Handled**](#_9jrz1ou2hptg) **101**

[**11. Appendix**](#_x98sc786bf6u) **101**

[11.1 Reference documents](#_e6od5y37q8cd) 101

# 

# 

# 

# 

# 

# 

# 

# 

# 1. Introduction

United Wholesale Mortgage (UWM) requires an automated document processing environment to help speed up their income verification process. Quantiphi will build a GCP environment in which users can upload various income and asset verification document types via an API and receive a classification or entity extraction response for the specific document type.

This project aims at developing an end-to-end pipeline for document classification and entity extraction. This Technical Design Document defines the solutioning of entity extraction using the GCP ecosystem with respect to components. The goal of this document is to collaboratively provide development details of the entity extraction solution.

## 

# 2. Requirements

This section lists all relevant functional and non-functional requirements

## benefits2.1 Software requirements The programming language used is **Python 3.7**

* aiohttp==3.7.4.post0
* Werkzeug==0.15.4
* flask-restx==0.2.0
* gunicorn==19.9.0
* requests==2.21.0
* flask==1.0.2
* google-cloud-logging==1.11.0
* google-cloud-storage==1.36.0
* google-cloud-documentai==0.3.0
* google-cloud-logging==1.11.0
* google-cloud-firestore==2.0.2
* googleapis-common-protos==1.52.0
* google-cloud-bigquery==2.7.0
* google-cloud-logging-handler==1.3
* pandas==1.0.3
* zipcodes==1.1.2
* numpy==1.18.2
* pdf2image==1.6.0
* PyPDF2==1.26.0
* google-cloud==0.34.0
* img2pdf==0.4.0
* datefinder==0.7.1
* fuzzywuzzy==0.18.0
* Shapely==1.7.1
* Flask-UUID==0.2
* dateparser==1.0.0
* nltk==3.4.4
* spacy==3.0.0
* cryptography==3.4.6
* PyJWT==2.0.1
* pyOpenSSL==20.0.1
* Flask-Cors==3.0.10
* backoff==1.10.0
* pdf2image==1.14.0
* PDFNetPython3==9.0.0
* en\_core\_web\_md == 3.0.0
* en\_core\_web\_trf == 3.0.0

**Note**:

We send in the two models i.e en\_core\_web\_md and en\_core\_web\_trf in the same directory as of docker file and while creating the image we install these dependencies, and after that we remove it , so that it should not contribute in increasing the docker image size.

## 2.2 Google Services

### 2.2.1 Document AI

Document AI allows you to identify and extract text from documents in over 200 languages for printed text and 50 languages for handwritten text.

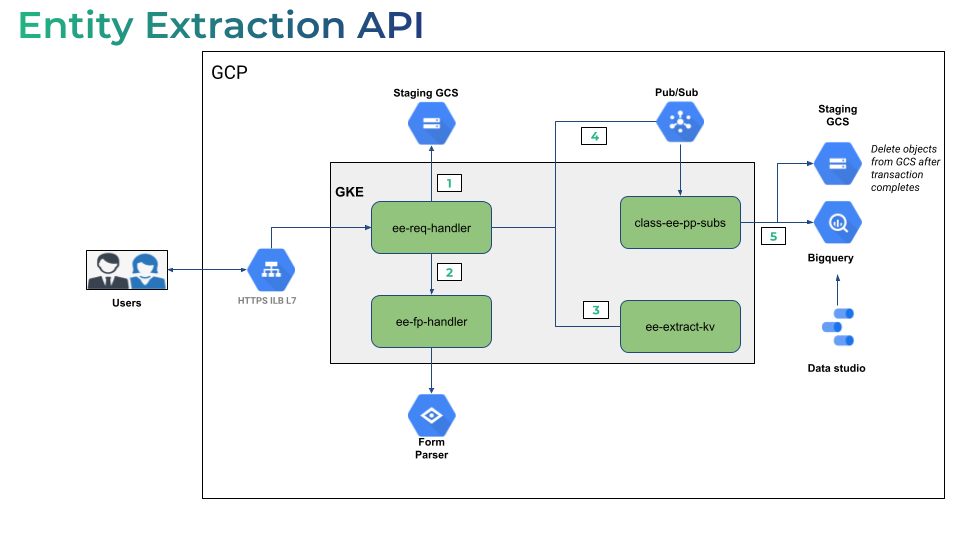
In Document AI API we are using Form Parser which is a general processor.

Document AI can detect and parse text from files, including text that contains unstructured data (fields, responses, dates, checkboxes, tables, etc.) in form documents.

We are using Asynchronous ("offline") requests to target longer documents. These types of requests start a long-running operation. When this operation finishes it stores output as a JSON file in a specified Cloud Storage bucket.

## 2.1 EE API

The Entity Extraction API takes a PDF document along with the document type as input and provides a JSON output with pre-identified entities specific to that document type extracted.



The above architecture defines four api’s to process requests for Entity Extraction.

Abbreviations used in API:

| **Abbreviation** | |
| --- | --- |
| ee | entity extraction |
| fp | Form Parser |
| kv | Key Value |
| pp | post processing |
| subs | subscriber |

API’s involved in the architecture:

| ee-req-handler | Batch documents in 5 pages for parallel processing of single request |
| --- | --- |
| ee-fp-handler | Invokes Form Parser API |
| ee-extract-kv | Extract key value pairs from Form Parser output |
| class-ee-pp-subs | Purge document and references from temporary storage locations such as Cloud Storage, and store details in Bigquery |

### **Architecture Steps**

* User will makes a request to the Entity Extraction API endpoint with a pdf file in payload and document type as part of the request
* The EE request handler API will split the document for every document greater than 5 pages(each batch containing 5 pages), and it will be stored in Cloud Storage bucket for temporary storage
* The request handler API will make an asynchronous request to the EE OCR API with GCS URI for every batch request
* The EE FP handler API will download the PDF object and will make a request to Form Parser API as soon as a batch is created, it is to extract the information from the document. This process will continue till EE API generates the last batch
* The EE FP handler API will store the Form Parser output in a GCS bucket and will respond to the EE request handler API for its status
* The EE request handler API will invokes EE Post processing API for further process such as cleanup and output storage to Bigquery
* The EE Post processing api will download the Form Parser output from GCS bucket, extract the Key-Values pairs, and it will respond to EE request handler API with the generated response
* The EE request handler API publishes the response to Post-processing topic and to the User
* The Post-processing Subscriber will pull the message from Pub/Sub topic and will perform a cleanup to temporary GCS bucket, and create a record in Bigquery for long term storage and Data Studio Dashboard

## 

## 2.2 Data sources

Files of 32 Document Types were shared by UWM with Quantiphi for the development of end-to-end entraction pipeline in GCP bucket:

**uwm-np-da-gcs-use4-copy-labelled-documents**

**uwm-np-da-gcs-use4-copy\_2-labelled-documents**

# 3. Input and Output

## 3.1 Input PDF

### 3.1.1 Pre-Processing

This module helps in improving the OCR results, and can be achieved using split pages,image rotation techniques.

The following steps are followed in Pre-Processing:

1. If pdf has more than 5 pages then split in batches of 5 pages
   1. Pass all batches to Form Parser asynchronously
   2. Check if any page is rotated in any batch, if yes, then use Form Parser’s output images to reconstruct the pdf batch
   3. Pass reconstructed pdf batch again to Form Parser
   4. Collect all asynchronous document objects returned by Form Parser
   5. Combine document objects of all batches, update page numbers and text boundaries for all elements on a page like Tokens, Blocks, KV Pairs etc
   6. Return final document object containing information of all pages and Rotation angle of each page
2. Else
   1. Pass the pdf to Form Parser
   2. Check if any page is rotated in the pdf, if yes, then use Form Parser’s output images to reconstruct the pdf
   3. Pass reconstructed pdf again to Form Parser
   4. Return final document object containing information of all pages and Rotation angle of each page

### 3.1.2 Extraction Using Form Parser

Google’s Form Parser API is used to extract all key-value pairs, tables, and entities from an input document/form. This is a pretrained model trained on variations of key-value pairs and tables across document types. The output also includes bounding box coordinates for all extracted information.

### 3.1.3 Post-Processing

This module helps to extract/clean the information obtained from Form Parser that is relevant to UWM and then store it in accordance with the output schema.This can be achieved by applying regex rules & mapping dictionaries.

### 3.1.3.1 Form 1120

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| Tax Year | - | * Extract from ROI method |
| Beginning Date | - | * Extract from ROI method |
| Ending date | - | * Extract from ROI method |
| Employer Identification Number | - | * Extract from Form Parser if Form parser rotates the doc then extract by ROI. |
| First Name | * In a few cases first name, last name , street address , city , state are combined. In those cases we use the Zipcode library to find zip code and identify the respective city and state. | * Extract from Form Parser with full name and than split it in first and last name |
| Last Name | * In a few cases first name, last name , street address , city , state are combined. In those cases we use the Zipcode library to find zip code and identify the respective city and state. | * Extract from Form Parser with full name and than split it in first and last name |
| Street Address | * In a few cases first name, last name , street address , city , state are combined. In those cases we use the Zipcode library to find zip code and identify the respective city and state. | * Extract from Form Parser if Form parser rotates the doc than extract by ROI. |
| City | * In a few cases first name, last name , street address , city , state are combined. In those cases we use the Zipcode library to find zip code and identify the respective city and state. | * Extract from Form Parser if Form parser rotates the doc than extract by ROI. |
| State | * In a few cases first name, last name , street address , city , state are combined. In those cases we use the Zipcode library to find zip code and identify the respective city and state. | * Extract from Form Parser if Form parser rotates the doc than extract by ROI. |
| Zip Code | * In a few cases first name, last name , street address , city , state are combined. In those cases we use the Zipcode library to find zip code and identify the respective city and state. | * Extract from Form Parser if Form parser rotates the doc than extract by ROI. |
| Date incorporated | - | * Extract from Form Parser if Form parser rotates the doc than extract by ROI. |
| Total Assets | - | * Extract from Form Parser if Form parser rotates the doc than extract by ROI. |
| Final Return Checkbox | - | * Extract from Form Parser if Form parser rotates the doc than extract by ROI. |
| Capital Gain Net Income | - | * Extract from ROI method |
| Net Gain (or Loss) |  | * Extract from ROI method |
| Other Income | - | * Extract from ROI method |
| Depreciation | - | * Extract from ROI method |
| Depletion | - | * Extract from ROI method |
| Net Operating Loss Deduction | - | * Extract from ROI method |
| Taxable Income | - | * Extract from ROI method |
| Total Tax | - | * Extract from ROI method |
| Schedule K Line 4a | - | * Extract from ROI method |
| Schedule K Line 4b | - | * Extract from ROI method |
| Mortgages, Notes, Bonds Payable in Less than 1 Year | - | * Extract from ROI method |
| Travel and Entertainment | - | * .Extract from ROI method |

#### 

#### 3.1.3.2 Schedule K1- 1065

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| Tax Year | - | * .Extract from ROI method |
| Beginning date | - | * Extract from ROI method |
| Ending date | - | * Extract from ROI method |
| Final K-1 checkbox | - | * Extract from Form Parser |
| Amended K-1 checkbox | - | * Extract from Form Parser |
| Employer Identification Number | - | Extract from Form Parser |
| Partnership’s name | * In a few cases first name, last name , street address , city , state are combined. In those cases we use the Zipcode library to find zip code and identify the respective city and state. | * Extract from Form Parser if not than extract from ROI |
| Partnership’s Street Address | * In a few cases first name, last name , street address , city , state are combined. In those cases we use the Zipcode library to find zip code and identify the respective city and state. | * Extract from Form Parser if not than extract from ROI |
| Partnership’s City | * In a few cases first name, last name , street address , city , state are combined. In those cases we use the Zipcode library to find zip code and identify the respective city and state. | * Extract from Form Parser if not than extract from ROI |
| Partnership’s State | * In a few cases first name, last name , street address , city , state are combined. In those cases we use the Zipcode library to find zip code and identify the respective city and state. | * Extract from Form Parser if not than extract from ROI |
| Partnership’s Zip Code | * In a few cases first name, last name , street address , city , state are combined. In those cases we use the Zipcode library to find zip code and identify the respective city and state. | * Extract from Form Parser if not than extract from ROI |
| Percentage of Ownership - Beginning Profit | - | * All 6 Percentage of Ownership keys are extracted from a table through ROI than split into individual 6 values |
| Percentage of Ownership - Ending Profit | - | * All 6 Percentage of Ownership keys are extracted from a table through ROI than split into individual 6 values |
| Percentage of Ownership - Beginning Loss | - | * All 6 Percentage of Ownership keys are extracted from a table through ROI than split into individual 6 values |
| Percentage of Ownership - Ending Loss | - | * All 6 Percentage of Ownership keys are extracted from a table through ROI than split into individual 6 values |
| Percentage of Ownership - Beginning Capital | - | * All 6 Percentage of Ownership keys are extracted from a table through ROI than split into individual 6 values |
| Percentage of Ownership - Ending Capital | - | * All 6 Percentage of Ownership keys are extracted from a table through ROI than split into individual 6 values |
| Ordinary Business Income (Loss) | - | * Extract from ROI method |
| Net Rental Real Estate Income (Loss) | - | * Extract from ROI method |
| Guaranteed payments for Service | - | * Extract from ROI method |
| Guaranteed payments for Capital | - | * Extract from ROI method |
| Total Guaranteed payments | - | * Extract from ROI method |
| Distributions\_1 | - | * Extract all distribution keys through ROI than split the numbers into Distributions\_1 and Distributions\_2 |
| Distributions\_2 | - | * Extract all distribution keys through ROI than split the numbers into Distributions\_1 and Distributions\_2 |

#### 

#### 3.1.3.3 Master insurance Policy

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| Insured Full Name | Mostly the name is combined with an address or multiple addresses | * Extract from Form Parser. * Name is combined with an address so filter all the address parts like city, state, zip code the remaining will be street address and name. * These are split by few rules like street number , P/0 or a new line |
| Insured Middle Initial | - | * Extract from Form Parser. * Name is combined with an address so filter all the address parts like city, state, zip code the remaining will be street address and name. * These are split by few rules like street number , P/0 or a new line |
| Insured Last Name | - | * Extract from Form Parser * . Name is combined with an address so filter all the address parts like city, state, zip code the remaining will be street address and name. * These are split by few rules like street number , P/0 or a new line |
| Property Street Address | * Free flowing * can also be present in extra pages. * combined among multiple addresses | * Use ROI to find the location where most probably we can find an address * Then use an Address regex to pull from that from the free flowing text. |
| Property City | * Free flowing * can also be present in extra pages. * combined among multiple addresses | * Use ROI to find the location where most probably we can find an address * Then use an Address regex to pull from that from the free flowing text. |
| Property State | * Free flowing * can also be present in extra pages. * combined among multiple addresses | * Use ROI to find the location where most probably we can find an address * Then use an Address regex to pull from that from the free flowing text. |
| Property Zipcode | * Free flowing * can also be present in extra pages. * combined among multiple addresses | * Use ROI to find the location where most probably we can find an address * Then use an Address regex to pull from that from the free flowing text. |
| Policy Effective Date | * Present in a table which can be anywhere. | * Write different ROI logics for different templates. * In a single template the key can be found in any row of the table. So written different ROI logics for each row for a single template |
| Policy Expiration Date | * Present in a table which can be anywhere. | * Write different ROI logics for different templates. * In a single template the key can be found in any row of the table. So written different ROI logics for each row for a single template |
| Dwelling Coverage | * Present in a table beside EA occurrence which can be present in any row or few times not present at all | * Write different ROI logics for different templates. * In a single template the key can be found in any row of the table. So written different ROI logics for each row for a single template |
| Mortgagee Clause | - | * Extract from different ROI logics for different templates. |
| Policy Type | * Free Flowing | * Use Regex to find predefined policy types in the Form. * Can’t extract if any new type policy type exists as the logic is written to find few predefined policy types. |
| Policy Number | - | * Write different ROI logics for different templates. * In a single template the key can be found in any row of the table. So written different ROI logics for each row for a single template |

#### 

#### 3.1.3.4 Verification of Income

**The required key-value pairs are extracted from the Form Parser json output using information related to Form Fields - Values, Blocks and Tokens.**

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| First Name, Middle Initial, Last Name | Mostly middle initial occurs attached to last name, like John AButler, but there are cases where it's written as ABUTLER (all caps), in these cases it cannot be separated from last name. | * Extract name from blocks, occurs mostly with SSN as <NAME XXX-XX-1234>. Using regex to identify correct block. Then iterate over Tokens to separate Name and SSN for bounding boxes. * Split Name into First Name, Middle Initial and Last Name. |
| Social Security Number | - | * Extract SSN from blocks, occurs mostly with SSN as <NAME XXX-XX-1234>. Using regex to identify correct blocks. Then iterate over Tokens to separate Name and SSN for bounding boxes. |
| Employer Name |  | * Extract from Form Parser Form Fields - Values. * If not found then identify the key on page using Tokens and gather tokens on the right side of the key as value. |
| Employer Street Address | - | * Extract from Form Parser Form Fields - Values. * If not found then identify the key on page using Tokens and gather tokens on the right side of the key as value. |
| Employer City | - | * Extract from Form Parser Form Fields - Values. |
| Employer State | - | * Extract from Form Parser Form Fields - Values. |
| Employer Zip Code | - | * Extract from Form Parser Form Fields - Values. |
| Information Current As Of | - | * Extract from Form Parser Form Fields - Values. |
| Most Recent Start Date | - | * Extract from Form Parser Form Fields - Values. * If not found then identify the key on page using Tokens and gather tokens on the right side of the key as value. |
| Original Hire Date | - | * Extract from Form Parser Form Fields - Values. * If not found then identify the key on page using Tokens and gather tokens on the right side of the key as value. |
| Total Time with Employer | - | * Extract from Form Parser Form Fields - Values. * If not found then identify the key on page using Tokens and gather tokens on the right side of the key as value. |
| Job Title | - | * Extract from Form Parser Form Fields - Values. * If not found then identify the key on page using Tokens and gather tokens on the right side of the key as value. |
| Rate of Pay | - | * Extract from Form Parser Form Fields - Values. |
| Average Hours Per Pay Period | - | * Extract from Form Parser Form Fields - Values. |
| Year 1 Base Pay | This table gets split in 2 pages in most cases, so Form Parser does not work | * Identify “Base Salary” Block on page and identify the other 5 columns on the right side using Blocks. * The values are below column blocks as individual blocks. Form ROI of each column block and get any value which falls inside column ROI. |
| Year 1 Overtime | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 1 Commission | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 1 Bonuses | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 1 Other Income | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 1 Total Pay | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 2 Base Pay | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 2 Overtime | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 2 Commission | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 2 Bonuses | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 2 Other Income | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 2 Total Pay | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 3 Base Pay | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 3 Overtime | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 3 Commission | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 3 Bonuses | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 3 Other Income | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |
| Year 3 Total Pay | - | * The value is below the column block as an individual block. Form ROI of column block and get any value which falls inside column ROI. |

#### 3.1.3.5 IRS - W2

**The required key-value pairs are extracted from the Form Parser json output using information related to Form Fields - Values, Blocks and Tokens.**

**Extracting Employee/Employer Address box:**

* Form bounding box of key using tokens which occur adjacent to each other, e.g. Employee’s name address and zip code, this is the key for employee address box
* To get ROI,
  1. Right bounding is double the width of key bbox
  2. Left bounding is 40 pixels less than the start of the key bbox, this is for the label which occurs in the form for a box like C., B.
  3. Top bounding is the key bbox itself
  4. Bottom bounding is a regex to match <State Zipcode>
* Gather all tokens which fall in the defined ROI
* Sort tokens on x,y coordinates as sometimes order is not correct in json
* All tokens on first line are Employer/Employee Name, all other tokens are part of address tokens
* Using zipcodes library, get city and state, use these to get correct city, state and zipcode tokens
* Leftover address tokens form the street address

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| First Name, Middle Initial, Last Name, Address, City, State, Zip Code | Sometimes Zip Code is written very far apart inside box, in those cases it goes beyond ROI so these fields are not captured correctly | * Extracted using the ROI method described above. * First line contains the name and rest all the tokens form the address. * City, State tokens are identified using zipcodes library. |
| Social Security Number | - | * Extract from Form Parser Form Fields - Values. |
| Employer Name, Employer Street Address, Employer City, Employer State, Employer Zip Code | Sometimes Zip Code is written very far apart inside box, in those cases it goes beyond ROI so these fields are not captured correctly | * Extracted using the ROI method described above. * First line contains the name and rest all the tokens form the address. * City, State tokens are identified using zipcodes library. |
| Tax Year | - | * Extract from Blocks, at the top of the page. |
| Employer Identification Number | - | * Extract from Form Parser Form Fields - Values. |
| Wages, Tips, Other Compensation | - | * Extract from Form Parser Form Fields - Values. * If not found, then iterate over Blocks and find the block containing this text. The value would be the first number after this key text. |
| Medical Wages and Tips | - | * Extract from Form Parser Form Fields - Values. * If not found, then iterate over Blocks and find the block containing this text. The value would be the first number after this key text. |

#### 

#### 3.1.3.6 Credit Supplements

**The required key-value pairs are extracted from the Form Parser json output using information related to Form Fields - Values, Blocks, Tokens and Lines.**

**Extraction Process for Tables:**

1. Identify the form type as the tables are not the same for every form and specific code has to be written to get table values. As per the shared files, 5 formats formed around 98% of the data.
2. Run the specific function as per the identified format to get the kv pairs. First identify the start boundary of the table and then find all adjacent columns. The values are present under each column.

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| Credit Agency | - | * Present at the footer on each page. Extracted using text matching to identify the correct Line and splitting text of the line. |
| File Number | - | * Extract from Form Parser Form Fields - Values. |
| Date Completed | - | * Extract from Form Parser Form Fields - Values. |
| Repositories | - | * Extract from Form Parser Form Fields - Values. |
| CoApplicant Social Security Number, Social Security Number | - | * Extracted by finding the key on the page (SSN). This occurs twice, for applicant and co-applicant. The value is adjacent to the key. * The first occurrence is of applicant and second is of co-applicant |
| Applicant Date of Birth, CoApplicant Date of Birth | - | * Extracted by finding the key on the page (DOB). This occurs twice, for applicant and co-applicant. The value is adjacent to the key. * The first occurrence is of applicant and second is of co-applicant |
| Current Address, Current City, Current State, Current Zipcode  Previous Address, Previous City, Previous State, Previous Zipcode | - | * Extract from Form Parser Form Fields - Values. * City, State and Street Address values are identified using zipcodes library. |
| Applicant First Name,  Applicant Middle Initial,  Applicant Last Name,  CoApplicant First Name,  CoApplicant Middle Initial,  CoApplicant Last Name | - | * Extract from Form Parser Form Fields - Values. * If not found, then extracted by finding the key on the page (Applicant/Co-Applicant). The value is adjacent to the key. |
| Creditor,  Date Reported,  Date Opened,  Date Last Active,  High Creditor Limit,  Account Type,  Account Balance,  Account Terms,  Account Payment Amount,  Past Due Amount,  Month Reviewed,  30 Day Lates,  60 Day Lates,  90 Day Lates,  Remarks | - | * For a specific format, table start boundary is identified, like **WHOSE** is written vertically or horizontally * Using start boundary identify all the columns and respective boundaries * Extract values present under each column, can be multiple |

#### 3.1.3.7 Gift Letter

**The required key-value pairs are extracted from the Form Parser json output using information related to Form Fields - Values, Blocks, Tokens and Lines.**

**Extraction Process:**

1. Identify the form type as the keys and ROI are not the same for every form and specific code has to be written to get values. As per the shared files, 3 formats were in majority, for others mainly dependent on FP and key synonyms.
2. Run the specific function as per the identified format to get the kv pairs.

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| Loan Number | - | * Extract from Form Parser Form Fields - Values. |
| Street Address,  City,  State,  Zip Code, | Sometimes present as a block with no key | * Extract from Form Parser Form Fields - Values. |
| Gift Amount | - | * Extract from Form Parser Form Fields - Values. * If not found then check Lines for keywords like “gift of”, “amount of” to get the correct line and extract amount using regex. |
| Source of Gift | - | * Extract from Form Parser Form Fields - Values. |
| Donor First Name,  Donor Middle Initial,  Donor Last Name | - | * Extract from Form Parser Form Fields - Values. * If not found then check Lines for keyword “I, ” to get the correct line and extract the name. |
| Date of Transfer | - | * Extract from Form Parser Form Fields - Values. |
| Donor Street Address,  Donor City,  Donor State,  Donor Zip Code | - | * Extract from Form Parser Form Fields - Values. |
| Donor Phone Number | - | * Extract from Form Parser Form Fields - Values. |
| Donor Relationship to Recipient | - | * Extract from Form Parser Form Fields - Values. |
| Recipient First Name,  Recipient Middle Initial,  Recipient Last Name | - | * Extract from Form Parser Form Fields - Values. |
| Recipient Street Address,  Recipient City,  Recipient State,  Recipient Zip Code | - | * Extract from Form Parser Form Fields - Values. |
| Donor Signature,  Donor Signature Date,  Borrower Signature,  Borrower Signature Date,  CoBorrower Signature,  CoBorrower Signature Date | - | * Extract from Form Parser Form Fields - Values. |

#### 3.1.3.8 Form 8825

**The required key-value pairs are extracted from the Form Parser json output using information related to Form Fields - Values and Tables.**

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| NAME | - | * Extract value from Form Parser Form Fields - Values. |
| EMPLOYER\_IDENTIFICATION\_NUMBER | - | * Extract value from Form Parser Form Fields - Values. |
| RENTAL\_PROPERTY\_ADDRESS\_A | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column * Split the Address into Street, City, State and Zip code |
| RENTAL\_PROPERTY\_CITY\_A | - |
| RENTAL\_PROPERTY\_STATE\_A | - |
| RENTAL\_PROPERTY\_ZIP\_CODE\_A | - |
| RENTAL\_PROPERTY\_ADDRESS\_B | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column * Split the Address into Street, City, State and Zip code |
| RENTAL\_PROPERTY\_CITY\_B | - |
| RENTAL\_PROPERTY\_STATE\_B | - |
| RENTAL\_PROPERTY\_ZIP\_CODE\_B | - |
| RENTAL\_PROPERTY\_ADDRESS\_C | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column * Split the Address into Street, City, State and Zip code |
| RENTAL\_PROPERTY\_CITY\_C | - |
| RENTAL\_PROPERTY\_STATE\_C | - |
| RENTAL\_PROPERTY\_ZIP\_CODE\_C | - |
| RENTAL\_PROPERTY\_ADDRESS\_D | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column * Split the Address into Street, City, State and Zip code |
| RENTAL\_PROPERTY\_CITY\_D | - |
| RENTAL\_PROPERTY\_STATE\_D | - |
| RENTAL\_PROPERTY\_ZIP\_CODE\_D | - |
| TYPE\_A | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| TYPE\_B | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| TYPE\_C | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| TYPE\_D | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| FAIR\_RENTAL\_DAYS\_A | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| FAIR\_RENTAL\_DAYS\_B | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| FAIR\_RENTAL\_DAYS\_C | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| FAIR\_RENTAL\_DAYS\_D | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| PERSONAL\_USE\_DAYS\_A | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| PERSONAL\_USE\_DAYS\_B | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| PERSONAL\_USE\_DAYS\_C | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| PERSONAL\_USE\_DAYS\_D | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| DEPRECIATION\_A | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| DEPRECIATION\_B | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| DEPRECIATION\_C | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| DEPRECIATION\_D | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_A1 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_A2 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_A3 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_A4 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_B1 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_B2 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_B3 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_B4 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_C1 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_C2 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_C3 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_C4 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_D1 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_D2 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_D3 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |
| OTHERS\_D4 | - | * Identify the correct table from the document * Find the row and column corresponding to the key * Extract the value based on the found row and column |

#### 3.1.3.9 Property Insurance Invoice

**The required key-value pairs are extracted from the Form Parser json output using information related to Form Fields - Values, Blocks and Tokens.**

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| INSURANCE\_COMPANY\_NAME | - | * Extracted from a pre-defined dictionary * If a value is not found, extracted from Form Parser Form Fields - Values. |
| POLICY\_NUMBER | - | * Extracted from Form Parser Form Fields - Values. |
| PROPERTY\_ADDRESS | - | * Extracted from Form Parser Form Fields - Values. * If value is incomplete as per regex pattern, use blocks to complete the address * If a value is not found, then find blocks with specific regex pattern * Split the address into Street, City, State and Zipcode |
| PROPERTY\_CITY | - |
| PROPERTY\_STATE | - |
| PROPERTY\_ZIPCODE | - |
| DOCUMENT\_DATE | - | * Extracted from Form Parser Form Fields - Values. * Use dateutil parser to return only dates |

#### 3.1.3.10 Payoff at Closing

**The required key-value pairs are extracted from the Form Parser json output using information related to Form Fields - Values, Tables, Blocks, Paragraphs, Lines and Tokens.**

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| CREDITOR | - | * Extracted from a pre-defined dictionary * If value not found, extracted from Form Parser Form Fields - Values. |
| BORROWER\_FIRST\_NAME | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then find the borrower name in blocks using mailing address and property address * Split the name into first name, middle initial and last name |
| BORROWER\_MIDDLE\_INITIAL | - |
| BORROWER\_LAST\_NAME | - |
| COBORROWER\_FIRST\_NAME | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then return the original text found for borrower name if it contains more than 3 words * Split the name into first name, middle initial and last name |
| COBORROWER\_MIDDLE\_INITIAL | - |
| COBORROWER\_LAST\_NAME | - |
| MAILING\_ADDRESS | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then find blocks containing specific regex pattern and not containing the creditor name. Try to avoid a block returning the same address as the property address. * Split the address into Street, City, State and Zipcode |
| MAILING\_CITY | - |
| MAILING\_STATE | - |
| MAILING\_ZIP\_CODE | - |
| STATEMENT\_DATE | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then find the date in blocks. Return the first occurrence. |
| PROPERTY\_ADDRESS | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then find blocks containing specific regex pattern * If value is incomplete as per regex pattern, use blocks to complete the address * Split the address into Street, City, State and Zipcode |
| PROPERTY\_CITY | - |
| PROPERTY\_STATE | - |
| PROPERTY\_ZIP\_CODE | - |
| ACCOUNT\_NUMBER | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then find the token having 10 digits. Return value. |
| PAY\_IN\_FULL\_DATE | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then find the line relevant to pay in full date. Return the date. |
| TOTAL\_PAYOFF\_AMOUNT | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then find the total payoff amount in blocks * If a value is not found, then find the total payoff amount in lines |
| PRINCIPAL\_BALANCE | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then find the principal balance in tables * If a value is not found, then find the principal balance in blocks * If a value is not found, then find the principal balance in lines |
| INTEREST\_GOOD\_THROUGH\_DATE | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then find the line relevant to interest good through date. Return the date. |
| INTEREST\_PER\_DIEM | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then find a block relevant to interest per diem, and search for an amount in it. Return the amount. |
| INTEREST\_PER\_MONTH | - | * Extracted from Form Parser Form Fields - Values. |
| LATE\_FEE | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then find a block related to late fee, and search for an amount in it. Return the amount. |
| NEXT\_DUE\_DATE | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then find the line relevant to the next due date. Return the date. |
| INTEREST\_PAID\_TO\_DATE | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then find the interest paid to date in tables * If a value is not found, then find the interest paid to date in blocks * If a value is not found, then find the interest paid to date in lines |

#### 3.1.3.11 Form 1120s

**The required key-value pairs are extracted from the Form Parser json output using information related to Form Fields - Values and Tokens.**

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| TAX\_YEAR |  |  |
| BEGINING\_DATE |  |  |
| ENDING\_DATE |  |  |
| FIRST\_NAME |  |  |
| MIDDLE\_INITIAL |  |  |
| LAST\_NAME |  |  |
| ADDRESS | - | * Extracted from Form Parser Form Fields - Values. |
| CITY | - | * Extracted from Form Parser Form Fields - Values. * If value not found, then find 5 consecutive tokens that match a specific regex pattern * Split the address into City, State and Zipcode |
| STATE | - |
| ZIP\_CODE | - |
| EMPLOYER\_IDENTIFICATION\_NUMBER | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then use regex on tokens to find 2 digits - 7 digits to extract value. |
| DATE\_INCORPORATED | - | * Extracted from Form Parser Form Fields - Values. * If a value is not found, then use regex on tokens to find {1 or 2} digits - {1 or 2} digits - {2 or 4} digits OR {1 or 2} digits / {1 or 2} digits / {2 or 4} digits to extract value. |
| OTHER\_INCOME\_LOSS |  | From custom ROI method using regex over it |
| DEPRECIATION |  | From custom ROI method using regex over it |
| DEPLETION |  | From custom ROI method using regex over it |
| TRAVEL\_AND\_ENTERTAINMENT |  | From custom ROI method using regex over it |
| CASH |  | From custom ROI method using regex over it |
| TRADE\_NOTES\_AND\_ACCOUNTS\_RECEIVABLE |  | From custom ROI method using regex over it |
| ALLOWANCE\_FOR\_BAD\_DEBTS |  | From custom ROI method using regex over it |
| US\_GOVERNMENT\_OBLIGATIONS |  | From custom ROI method using regex over it |
| TAX\_EXEMPT\_SECURITIES |  | From custom ROI method using regex over it |
| OTHER\_CURRENT\_ASSETS |  | From custom ROI method using regex over it |
| OTHER\_ASSETS |  | From custom ROI method using regex over it |
| TOTAL\_ASSETS |  | From custom ROI method using regex over it |
| MORTGAGES\_NOTES\_BONDS\_PAYABLE\_IN\_LESS\_THAN\_1\_YEAR |  | From custom ROI method using regex over it |
| OTHER\_CURRENT\_LIABILITIES |  | From custom ROI method using regex over it |
| ALL\_NONRECOURSE\_LOANS |  | From custom ROI method using regex over it |

#### 3.1.3.12 Bank Statements

**The required key-value pairs are extracted from the Form Parser json output using information related to Form Fields - Values, Blocks and Lines.**

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| BANK\_NAME | - | * Used a predefined dictionary to map certain keywords to the corresponding Bank\_Name |
| ADDRESS | - | * ADDRESS, CITY, STATE, ZIP\_CODE, ACCOUNT\_HOLDER and COACCOUNT\_HOLDER always occur together (if present). So, used Blocks attribute of form parser to extract this block and then split the details accordingly |
| CITY |
| STATE |
| ZIP\_CODE |
| ACCOUNT\_HOLDER\_FIRST\_NAME |
| ACCOUNT\_HOLDER\_MIDDLE\_INITIAL |
| ACCOUNT\_HOLDER\_LAST\_NAME |
| COACCOUNT\_HOLDER\_FIRST\_NAME |
| COACCOUNT\_HOLDER\_MIDDLE\_INITIAL |
| COACCOUNT\_HOLDER\_LAST\_NAME |
| TYPE\_OF\_ACCOUNT | - | * Used a predefined dictionary to map certain keywords to the corresponding ‘Type of Account’ |
| ENDING\_BALANCE\_OF\_ACCOUNT | - | * Extracted from Form Parser Form Fields - Values. |
| ACCOUNT\_NUMBER | - | * Extracted from Form Parser Form Fields - Values. |
| BEGINNING\_DATE\_OF\_STATEMENT | - | * BEGINNING\_DATE\_OF\_STATEMENT and ENDING\_DATE\_OF\_STATEMENT occur together. So, used Lines attribute of form parser to extract this line with the help of regex and then split the details accordingly |
| ENDING\_DATE\_OF\_STATEMENT |

#### 3.1.3.13 Closing Protection Letter

**The required key-value pairs are extracted from the Form Parser json output using information related to Form Fields - Values, Blocks.**

* CPL has 7 commonly occuring templates. We have to first identify if the document belongs to any of these templates, if yes, then we get the values by calling the respective functions
* Link to internal sheet - https://docs.google.com/spreadsheets/d/1jcrHJrszXysFAS66gP7ywdb8VJHnB4KWtDt3StyktH4/edit?usp=sharing

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| TITLE\_INSURANCE\_COMPANY |  | * Extract from Form Parser Form Fields - Values * If value is not found by FP, search for company names predefined in dictionary in blocks of text |
| CPL\_EO\_DATE |  | * Extract from Form Parser Form Fields - Values * For some templates, if value not found by FP, we run date regex in block of text to find date as only one date occurs in that template which is the CPL\_EO\_DATE |
| MORTGAGEE\_CLAUSE |  | * Extract from Form Parser Form Fields - Values * If value is not found by FP, based on template,a particular function is called which will search for particular keywords in blocks of text. Once the keyword is found, we take the lines below or next to the keyword |
| TITLE\_UNDERWRITER |  | * Extract from Form Parser Form Fields - Values * If value is not found by FP, based on template, a particular function is called which will search for particular keywords in blocks of text. Once the keyword is found, we take the lines below or next to the keyword |
| BORROWER\_FIRST\_NAME, BORROWER\_MIDDLE\_INITIAL  BORROWER\_LAST\_NAME CO\_BORROWER\_FIRST\_NAME,  CO\_BORROWER\_MIDDLE\_INITIAL,  CO\_BORROWER\_LAST\_NAME | * Mostly middle initial occurs attached to last name, like John AButler, but there are cases where it's written as ABUTLER (all caps), in these cases it cannot be separated from last name. * If we have text as “Jane A Smith, an unmarried woman”, we will not be able to distinguish the last 3 words as not part of name and will go as co-borrower name parts | * Split Name into Borrower First Name, Middle Initial and Last Name and Co-Borrower First Name, Middle Initial and Last Name * Use separators such as “, and & ” to separate out Borrower from Co-Borrower |
| ADDRESS, CITY, STATE,  ZIP\_CODE | - | * Extract from Form Parser Form Fields - Values * If value is not found, then find blocks containing specific address regex pattern * Using zipcodes library, get city and state, use these to get correct city, state and zipcode tokens * Leftover address tokens form the street address |
| TITLE\_COMMITMENT\_NUMBER | - | * Extract from Form Parser Form Fields - Values. |
| LOAN\_NUMBER | - | * Extract from Form Parser Form Fields - Values. |
| CPL\_SIGNATURE\_1 | - | * Extract from Form Parser Form Fields - Values. |
| CPL\_SIGNATURE\_2 | - | * Extract from Form Parser Form Fields - Values. |

#### 

#### 

#### 

#### 3.1.3.14 Paystub

**The required key-value pairs are extracted from the Form Parser json output using information related to Form Fields - Values, Blocks.**

* Paystub has several templates. One of the most commonly occurring one is ADP. We have to first identify if the document belongs to ADP, if yes, then we get the values by calling the respective functions. If template is not ADP, we have written a generalised code for the remaining templates.
* Link to internal sheet - https://docs.google.com/spreadsheets/d/1T56w4p2t7VmNaGI0iUJ8j-NEZU3WodPuZvV9iip3NLM/edit?usp=sharing

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| FIRST\_NAME, MIDDLE\_NAME,  LAST\_NAME | Mostly middle initial occurs attached to last name, like John AButler, but there are cases where it's written as ABUTLER (all caps), in these cases it cannot be separated from last name. | * Extract from Form Parser Form Fields - Values * If value is not found by FP, then iterate through the blocks till we find a text matching address regex. Remove the address from the block text, remaining value is the Name * To identify whether name is of employer or employee we run the name through NER model (spacy) * If NER returns label as person, the name is assigned to employee name * Split Name into First Name, Middle Name and Last Name |
| ADDRESS, CITY, STATE, ZIP\_CODE |  | * Extract from Form Parser Form Fields - Values * If value is not found, then find blocks containing specific address regex pattern * Name and address are part of same block, if name is returned with person label by NER model, the address of that block becomes employee address * Using zipcodes library, get city and state, use these to get correct city, state and zipcode tokens * Leftover address tokens form the street address |
| EMPLOYER\_NAME |  | * Extract from Form Parser Form Fields - Values * If value is not found by FP, then iterate through the blocks till we find a text matching address regex. Remove the address from the block text, remaining value is the Name * To identify whether name is of employer or employee we run the name through NER model (spacy) * If NER returns label that is not person, the name is assigned to employer name |
| EMPLOYER\_ADDRESS, EMPLOYER\_CITY, EMPLOYER\_STATE, EMPLOYER\_ZIP\_CODE | - | * Extract from Form Parser Form Fields - Values * If value is not found, then find blocks containing specific address regex pattern * Name and address are part of same block, if name is returned with not person label by NER model, the address of that block becomes employer address * Using zip codes library, get city and state, use these to get correct city, state and zipcode tokens * Leftover address tokens form the employer address |
| START\_DATE | - | * Extract from Form Parser Form Fields - Values |
| HIRE\_DATE | - | * Extract from Form Parser Form Fields - Values |
| SOCIAL\_SECURITY\_NUMBER | - | * Extract from Form Parser Form Fields - Values * Iterate through block and search for token matching SSN regex |
| PAY\_PERIOD\_START\_DATE | - | * Extract from Form Parser Form Fields - Values * Sometimes the start date value contains text along with both start and end date. We run a date regex to find the dates in the text, and then assign first date as pay period start date and second date as pay period end date |
| PAY\_PERIOD\_END\_DATE |  | * Extract from Form Parser Form Fields - Values * Sometimes the end date value contains text along with both start and end date. We run a date regex to find the dates in the text, and then assign first date as pay period start date and second date as pay period end date |
| PAYMENT\_DATE |  | * Extract from Form Parser Form Fields - Values |
| PAY\_RATE |  | * Extract from Form Parser Form Fields - Values * For ADP template - we take the pay rate as the rate given for ‘Regular’ key word in Earnings by using custom ROI |
| PAY\_FREQUENCY |  | * Extract from Form Parser Form Fields - Values |
| HOURS\_WORKED |  | * Extract from Form Parser Form Fields - Values * For ADP template - * We first check if ‘total hours worked’ token is present in the doc, if yes, we take the current value of that. * If not, we take the hours worked as the hours given for ‘Regular’ key word in Earnings using custom ROI |
| SALARY |  | * Extract from Form Parser Form Fields - Values |
| JOB\_TITLE |  | * Extract from Form Parser Form Fields - Values |
| GROSS\_CURRENT\_INCOME, GROSS\_CURRENT\_YTD\_INCOME |  | * Extract from Form Parser Form Fields - Values * If value is not found by FP, we iterate through tokens to find predefined keywords. Once keyword is found, we take value below it as gross current income, and value below that as gross current ytd income * For ADP template - we take the values of current and ytd given for the token ‘Gross Pay’ by using custom ROI |
| REGULAR\_OR\_BASE\_PAY\_CURRENT\_TOTAL |  | * Extract from Form Parser Form Fields - Values * For ADP template - we take the current value of ‘Regular’ keyword under Earnings using custom ROI |
| REGULAR\_OR\_BASE\_PAY\_YTD\_TOTAL |  | * Extract from Form Parser Form Fields - Values * For ADP template - we take the YTD value of ‘Regular’ keyword under Earnings using custom ROI |
| COMMISSIONS\_CURRENT\_TOTAL |  | * Extract from Form Parser Form Fields - Values * For ADP template - we take the current value of ‘Commission’ keyword under Earnings using custom ROI |
| COMMISSIONS\_YTD\_TOTAL |  | * Extract from Form Parser Form Fields - Values * For ADP template - we take the YTD value of ‘Commission’ keyword under Earnings using custom ROI |
| BONUS\_CURRENT\_TOTAL |  | * Extract from Form Parser Form Fields - Values * For ADP template - we take the current value of ‘Bonus’ keyword under Earnings using custom ROI |
| BONUS\_YTD\_TOTAL |  | * Extract from Form Parser Form Fields - Values * For ADP template - we take the YTD value of ‘Bonus’ keyword under Earnings using custom ROI |
| OVERTIME\_CURRENT\_TOTAL |  | * Extract from Form Parser Form Fields - Values * For ADP template - we take the current value of ‘Overtime’ or ’OT’ keyword under Earnings using custom ROI |
| OVERTIME\_YTD\_TOTAL |  | * Extract from Form Parser Form Fields - Values * For ADP template - we take the YTD value of ‘Overtime’ or ’OT’ keyword under Earnings using custom ROI |
| PAY\_TYPE\_OTHER\_1\_DESCRIPTION,  PAY\_TYPE\_OTHER\_2\_DESCRIPTION,  PAY\_TYPE\_OTHER\_3\_DESCRIPTION,  PAY\_TYPE\_OTHER\_4\_DESCRIPTION  PAY\_TYPE\_OTHER\_5\_DESCRIPTION |  | * Extracted using Form Parser line items and custom ROI * We ensure that the terms Regular, Bonus, Overtime and Commission do not come in paytype description |
| PAY\_TYPE\_OTHER\_1\_CURRENT\_TOTAL,  PAY\_TYPE\_OTHER\_2\_CURRENT\_TOTAL,  PAY\_TYPE\_OTHER\_3\_CURRENT\_TOTAL,  PAY\_TYPE\_OTHER\_4\_CURRENT\_TOTAL,  PAY\_TYPE\_OTHER\_5\_CURRENT\_TOTAL |  | * Extracted using custom ROI * We ensure that the current values of Regular, Bonus, Overtime and Commission do not come in paytype current |
| PAY\_TYPE\_OTHER\_1\_YTD\_TOTAL,  PAY\_TYPE\_OTHER\_2\_YTD\_TOTAL,  PAY\_TYPE\_OTHER\_3\_YTD\_TOTAL,  PAY\_TYPE\_OTHER\_4\_YTD\_TOTAL,  PAY\_TYPE\_OTHER\_5\_YTD\_TOTAL |  | * Extracted using custom ROI * We ensure that the YTD values of Regular, Bonus, Overtime and Commission do not come in paytype YTD |
| DEDUCTION\_OTHER\_1\_DESCRIPTION,  DEDUCTION\_OTHER\_2\_DESCRIPTION,  DEDUCTION\_OTHER\_3\_DESCRIPTION,  DEDUCTION\_OTHER\_4\_DESCRIPTION,  DEDUCTION\_OTHER\_5\_DESCRIPTION,  DEDUCTION\_OTHER\_6\_DESCRIPTION,  DEDUCTION\_OTHER\_7\_DESCRIPTION,  DEDUCTION\_OTHER\_8\_DESCRIPTION,  DEDUCTION\_OTHER\_9\_DESCRIPTION,  DEDUCTION\_OTHER\_10\_DESCRIPTION |  | * Extracted using Form Parser line items and custom ROI |
| DEDUCTION\_OTHER\_1\_CURRENT\_TOTAL,  DEDUCTION\_OTHER\_2\_CURRENT\_TOTAL,  DEDUCTION\_OTHER\_3\_CURRENT\_TOTAL,  DEDUCTION\_OTHER\_4\_CURRENT\_TOTAL,  DEDUCTION\_OTHER\_5\_CURRENT\_TOTAL,  DEDUCTION\_OTHER\_6\_CURRENT\_TOTAL,  DEDUCTION\_OTHER\_7\_CURRENT\_TOTAL,  DEDUCTION\_OTHER\_8\_CURRENT\_TOTAL,  DEDUCTION\_OTHER\_9\_CURRENT\_TOTAL,  DEDUCTION\_OTHER\_10\_CURRENT\_TOTAL |  | * Extracted using custom ROI |
| DEDUCTION\_OTHER\_1\_YTD\_TOTAL,  DEDUCTION\_OTHER\_2\_YTD\_TOTAL,  DEDUCTION\_OTHER\_3\_YTD\_TOTAL,  DEDUCTION\_OTHER\_4\_YTD\_TOTAL,  DEDUCTION\_OTHER\_5\_YTD\_TOTAL,  DEDUCTION\_OTHER\_6\_YTD\_TOTAL,  DEDUCTION\_OTHER\_7\_YTD\_TOTAL,  DEDUCTION\_OTHER\_8\_YTD\_TOTAL,  DEDUCTION\_OTHER\_9\_YTD\_TOTAL,  DEDUCTION\_OTHER\_10\_YTD\_TOTAL |  | * Extracted using custom ROI |

#### 3.1.3.15 Form 1040

#### 

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| TAX\_YEAR |  | * Extracted Using ROI (tokens) * Present In the Top 10 percent of the page in the right half. |
| FIRST\_NAME |  | * Form Parser Key value pairs. * If its not extracted via form parser, using ROI, search for the key and pick the text immediately below it |
| MIDDLE\_INITIAL |  | * Form Parser Key value pairs. * If its not extracted via form parser, using ROI, search for the key and pick the text immediately below it |
| LAST\_NAME |  | * Form Parser Key value pairs. * If its not extracted via form parser, using ROI, search for the key and pick the text immediately below it |
| SOCIAL\_SECURITY\_NUMBER |  | * Form Parser Key value pairs. |
| SPOUSE\_FIRST\_NAME |  | * Form Parser Key value pairs. * If its not extracted via form parser, using ROI, search for the key and pick the text immediately below it |
| SPOUSE\_MIDDLE\_INITIAL |  | * Form Parser Key value pairs. * If its not extracted via form parser, using ROI, search for the key and pick the text immediately below it |
| SPOUSE\_LAST\_NAME |  | * Form Parser Key value pairs. * If its not extracted via form parser, using ROI, search for the key and pick the text immediately below it |
| SPOUSE\_SOCIAL\_SECURITY\_NUMBER |  | * Form Parser Key value pairs. |
| ADDRESS |  | * Form Parser Key value pairs. * If its not extracted via form parser,using ROI, search for the Key name and pick up the text immediately below the Key.. |
| CITY |  | * Form Parser Key value pairs. * If its not extracted via form parser,using ROI, search for the Key name and pick up the text immediately below the Key. Its further split into City, State and ZipCode using the zipcodes library |
| STATE |  | * Form Parser Key value pairs. * If its not extracted via form parser,using ROI, search for the Key name and pick up the text immediately below the Key. Its further split into City, State and ZipCode using the zipcodes library |
| ZIP\_CODE |  | * Form Parser Key value pairs. * If its not extracted via form parser,using ROI, search for the Key name and pick up the text immediately below the Key. Its further split into City, State and ZipCode using the zipcodes library |
| WAGES\_SALARIES\_TIPS |  | * Find the Key name on the Page and then apply regex to extract first match of the Value to the right of the Key |
| IRA\_DISTRIBUTIONS |  | * Find the Key name on the Page and then apply regex to extract first match of the Value to the right of the Key |
| IRA\_DISTRIBUTIONS\_TAXABLE\_AMOUNT |  | * Find the Key name on the Page and then apply regex to extract the Value to the right of the Key. |
| PENSION\_AND\_ANNUITIES |  | * Find the Key name on the Page and then apply regex to extract first match of the Value to the right of the Key |
| PENSION\_AND\_ANNUITIES\_TAXABLE\_AMOUNT |  | * Find the Key name on the Page and then apply regex to extract the Value to the right of the Key. |
| SOCIAL\_SECURITY\_BENEFITS |  | * Find the Key name on the Page and then apply regex to extract first match of the Value to the right of the Key |
| SOCIAL\_SECURITY\_BENEFITS\_TAXABLE\_AMOUNT |  | * Find the Key name on the Page and then apply regex to extract the Value to the right of the Key. |
| CAPITAL\_GAINS\_OR\_LOSS |  | * Find the Key name on the Page and then apply regex to extract the Value to the right of the Key |

#### 

#### 3.1.3.16 Schedule E Form 1040

#### 

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| YEAR\_OF\_FORM |  | * Extracted Using ROI (tokens) * Present In the Top 10 percent of the page in the right half. |
| NAMES |  | * Form Parser Key value pairs. * If its not extracted via form parser, using ROI, search for the key and pick the text immediately below it |
| SSN |  | * Form Parser Key value pairs. |
| PHYSICAL\_ADDRESSES\_OF\_PROPERTIES\_1A\_A |  | * Using ROI, search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index |
| PHYSICAL\_ADDRESSES\_OF\_PROPERTIES\_1A\_B |  | * Using ROI, search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index |
| PHYSICAL\_ADDRESSES\_OF\_PROPERTIES\_1A\_C |  | * Using ROI, search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index |
| TYPE\_OF\_PROPERTY\_1B\_A |  | * Using ROI, search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index |
| TYPE\_OF\_PROPERTY\_1B\_B |  | * Search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index |
| TYPE\_OF\_PROPERTY\_1B\_C |  | * Search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index |
| FAIR\_RENTAL\_DAYS\_A |  | * Search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index |
| FAIR\_RENTAL\_DAYS\_B |  | * Search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index |
| FAIR\_RENTAL\_DAYS\_C |  | * Search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index |
| PERSONAL\_USE\_DAYS\_A |  | * Search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index |
| PERSONAL\_USE\_DAYS\_B |  | * Search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index |
| PERSONAL\_USE\_DAYS\_C |  | * Search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index |
| RENTS\_RECEIVED\_A |  | * Search for a marker which marks the start of the table. * Now all the values are present to the right of this marker and below it. Search for the columns ‘A”, ‘B’ and “C’ using tokens. ROI for values in Column A range till the midpoint of A and B, for Column B its midpoint of A and B to midpoint of B and C, for C its midpoint of B and C onwards. * Now search for the index number corresponding to the key .Mark it as the upper bound . Search for the next index and mark it as the lower Bound. Now search to the right between this space using regex. When the value is found, put it into column A,B or C depending on its region as discussed above |
| RENTS\_RECEIVED\_B |  |
| RENTS\_RECEIVED\_C |  |
| INSURANCE\_A |  |
| INSURANCE\_B |  |
| INSURANCE\_C |  |
| MORTGAGE\_INTEREST\_PAID\_TO\_BANKS\_ETC\_A |  |
| MORTGAGE\_INTEREST\_PAID\_TO\_BANKS\_ETC\_B |  |
| MORTGAGE\_INTEREST\_PAID\_TO\_BANKS\_ETC\_C |  |
| REPAIRS\_A |  |
| REPAIRS\_B |  |
| REPAIRS\_C |  |
| TAXES\_A |  |
| TAXES\_B |  |
| TAXES\_C |  |
| DEPRECIATE\_EXPENSE\_OR\_DEPLETION\_A |  |
| DEPRECIATE\_EXPENSE\_OR\_DEPLETION\_B |  |
| DEPRECIATE\_EXPENSE\_OR\_DEPLETION\_C |  |
| OTHER\_A |  |
| OTHER\_B |  |
| OTHER\_C |  |
| TOTAL\_EXPENSES\_A |  |
| TOTAL\_EXPENSES\_B |  |
| TOTAL\_EXPENSES\_C |  |
| NAME\_A |  | * Search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index. This continues till You find the next Key which is either a S or a P. That is the right boundary. If not found, we go till we find the EIN |
| NAME\_B |  | * Search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index. This continues till You find the next Key which is either a S or a P. That is the right boundary. If not found, we go till we find the EIN |
| NAME\_C |  | * Search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index. This continues till You find the next Key which is either a S or a P. That is the right boundary. If not found, we go till we find the EIN |
| NAME\_D |  | * Search for the key name, then search for the index. Note the upper and lower bound of the index. Now pick the tokens to the right of the index between the upper and lower bounds of the index. This continues till You find the next Key which is either a S or a P. That is the right boundary. If not found, we go till we find the EIN |
| PARTNERSHIP\_OR\_S\_CORP\_A |  | * Find the ROI and then search for S or P between the upper and lower bounds |
| PARTNERSHIP\_OR\_S\_CORP\_B |  | * Find the ROI and then search for S or P between the upper and lower bounds |
| PARTNERSHIP\_OR\_S\_CORP\_C |  | * Find the ROI and then search for S or P between the upper and lower bounds |
| PARTNERSHIP\_OR\_S\_CORP\_D |  | * Find the ROI and then search for S or P between the upper and lower bounds |
| FOREIGN\_PARTNERSHIP\_CHECKBOX\_A |  | * Find the ROI and then search if the tick is present or not |
| FOREIGN\_PARTNERSHIP\_CHECKBOX\_B |  | * Find the ROI and then search if the tick is present or not |
| FOREIGN\_PARTNERSHIP\_CHECKBOX\_C |  | * Find the ROI and then search if the tick is present or not |
| FOREIGN\_PARTNERSHIP\_CHECKBOX\_D |  | * Find the ROI and then search if the tick is present or not |
| EMPLOYER\_INDENTIFICATION\_NUMBER\_A |  | * Find the ROI and then using regex check if EIN is present or not |
| EMPLOYER\_INDENTIFICATION\_NUMBER\_B |  | * Find the ROI and then using regex check if EIN is present or not |
| EMPLOYER\_INDENTIFICATION\_NUMBER\_C |  | * Find the ROI and then using regex check if EIN is present or not |
| EMPLOYER\_INDENTIFICATION\_NUMBER\_D |  | * Find the ROI and then using regex check if EIN is present or not |

#### 

#### 3.1.3.17 Wire Instructions

#### 

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| WIRE\_BANK\_ABA |  | * Extracted Using ROI (tokens) * Its a 9 digit number present anywhere in the document. Extracted using Regex |
| BANK\_ABA\_NAME |  | * There is a csv which maps the extracted ABA with the bank names. That is used to extract the bank name |
| WIRE\_BANK\_ACCOUNT |  | * Form Parser Key value pairs. * If not, search for the key name and pick the value to the right of the key |
| TITLE\_COMPANY\_NAME |  | * Form Parser Key value pairs. * If not pick the very first line of the OCR text and depending on the number of characters pick one more line or return that value |
| TITLE\_COMPANY\_ADDRESS |  | * Iterate through various blocks and apply the regex for address. Take the first match and then split it into Street Address, City, State and ZipCode using Zipcodes library |
| TITLE\_COMPANY\_CITY |  | * Iterate through various blocks and apply the regex for address. Take the first match and then split it into Street Address, City, State and ZipCode using Zipcodes library |
| Title Company State |  | * Iterate through various blocks and apply the regex for address. Take the first match and then split it into Street Address, City, State and ZipCode using Zipcodes library |
| Title Company Zip Code |  | * Iterate through various blocks and apply the regex for address. Take the first match and then split it into Street Address, City, State and ZipCode using Zipcodes library |

#### 

#### 3.1.3.18 Mortgage Statements

#### 

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| BANK\_NAME\_MORTGAGE\_CO |  | * Used a predefined dictionary to map certain keywords to the corresponding Bank\_Name |
| BORROWER\_FIRST\_NAME |  | * Present in a block with the first 2 lines being the Name and Co-Borrowers Name and the last 2 being address * Identify the address block using regex and then split it into the Name Components |
| BORROWER\_MIDDLE\_INITIAL |  |
| BORROWER\_LAST\_NAME |  |
| COBORROWER\_FIRST\_NAME |  |
| COBORROWER\_MIDDLE\_INITIAL |  |
| COBORROWER\_LAST\_NAME |  |
| MAILING\_ADDRESS |  | * Using the Regex for address, iterate through various blocks and pick the right one * Then split into Street Address,city,state and Zipcode using the Zipcode Library |
| MAILING\_CITY |  |
| MAILING\_STATE |  |
| MAILING\_ZIP\_CODE |  |
| STATEMENT\_DATE |  | * Search for the key name, and then search the region to the right of the key first using the regex. If matched , return otherwise search the region below the key Name |
| ACCOUNT\_NUMBER |  | * Search for the key name, and then search the region to the right of the key first using the regex. If matched , return otherwise search the region below the key Name |
| PAYMENT\_DUE\_DATE |  | * Search for the key name, and then search the region to the right of the key first using the regex. If matched , return otherwise search the region below the key Name |
| AMOUNT\_DUE |  | * Search for the key name, and then search the region to the right of the key first using the regex. If matched , return otherwise search the region below the key Name |
| LATE\_FEE |  | * Search for the block containing the Key name * Apply regex to find the required Value in that block |
| PROPERTY\_ADDRESS |  | * Search for the key Name. Then apply regex to the area present to the right of the key as well as below it to pick the address. Then split it using Zipcodes Library |
| PROPERTY\_CITY |  |
| PROPERTY\_STATE |  |
| PROPERTY\_ZIP\_CODE |  |
| OUTSTANDING\_PRINCIPAL |  | * Search for the key Name.. Using Regex, check for the Value to the right of the key |
| ESCROW\_BALANCE |  | * Search for the key Name.. Using Regex, check for the Value to the right of the key |
| INTEREST\_RATE |  | * Search for the key Name.. Using Regex, check for the Value to the right of the key |
| PRINCIPAL\_AMOUNT\_BALANCE |  | * Search for the key Name.. Using Regex, check for the Value to the right of the key |
| INTEREST\_AMOUNT |  | * Search for the key Name.. Using Regex, check for the Value to the right of the key |
| ESCROW\_TAXES\_AND\_INSURANCE |  | * Search for the key Name.. Using Regex, check for the Value to the right of the key |
| LAST\_PAYMENT\_MADE |  | * Search for the key Name.. Using Regex, check for the Value to the right of the key |
| LAST\_PAYMENT\_MADE\_DATE |  | * Search for the key Name.. Using Regex, check for the Value to the right of the key or below it |
| PRINCIPAL |  | * Search for the key Name.. Using Regex, check for the Value to the right of the key |
| INTEREST |  | * Search for the key Name.. Using Regex, check for the Value to the right of the key |
|  |  |  |

#### 

#### 3.1.3.19 Form 1065

#### 

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| Tax Year |  | From custom ROI method |
| Beginning Date |  | From custom ROI method using regex over it |
| Ending Date |  | From custom ROI method using regex over it |
| Depreciate |  | From custom ROI method using regex over it |
| Less Depreciation Line 16b |  | From custom ROI method using regex over it |
| Less Depreciation Line 16c |  | From custom ROI method using regex over it |
| Depletion |  | From custom ROI method using regex over it |
| Ordinary Other income |  | From custom ROI method using regex over it |
| Other Income (Loss) |  | From custom ROI method using regex over it |
| Cash |  | From custom ROI method using regex over it |
| Trade Notes and Accounts Receivable |  | From custom ROI method using regex over it |
| Less Allowance for Bad Debts |  | From custom ROI method using regex over it |
| U.S. Government Obligations |  | From custom ROI method using regex over it |
| Tax-Exempt Securities |  | From custom ROI method using regex over it |
| Other Current Assets |  | From custom ROI method using regex over it |
| Total Assets |  | From custom ROI method using regex over it |
| Other Assets |  | From custom ROI method using regex over it |
| Accounts Payable |  | From custom ROI method using regex over it |
| Mortgages, Notes, Bonds Payable in Less than 1... |  | From custom ROI method using regex over it |
| Other Current Liabilities |  | From custom ROI method using regex over it |
| All Nonrecourse Loans |  | From custom ROI method using regex over it |
| Mortgage Notes, Bonds Payable Less than 1 Year |  | From custom ROI method using regex over it |
| Travel and Entertainment |  | From custom ROI method using regex over it |
| Name of Partnership |  | Form parser key value pairs |
| Business City |  | Using both form parser key value pairs and custom ROI |
| Business State |  | Using both form parser key value pairs and custom ROI |
| Business Street Address |  | Using both form parser key value pairs and custom ROI |
| Business Zip Code |  | Using both form parser key value pairs and custom ROI |
| Employer Identification Number |  | Form parser key value pairs |
| Date Business Started |  | Form parser key value pairs |

#### 

#### 3.1.3.20 Driver’s Licence

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| DATE\_OF\_BIRTH | - | From custom ROI method using regex over it |
| EXPIRATION\_DATE | - | From custom ROI method using regex over it |
| FIRST\_NAME | - | Custom logic over ocr text. |
| MIDDLE\_INITIAL | - | Custom logic over ocr text. |
| LAST\_NAME | - | Custom logic over ocr text. |
| ADDRESS | - | From custom ROI method using regex over it |
| CITY | - | From custom ROI method using regex over it |
| STATE | - | Using regex over tokens. |
| ZIP\_CODE | - | Using regex over tokens. |

#### 

#### 

#### 3.1.3.21 IRS 1099-R

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| YEAR\_OF\_FORM | - | Using regex over tokens. |
| PAYERS\_FIRST\_NAME | - | From custom ROI method using regex over it |
| PAYERS\_MIDDLE\_INITIAL | - | From custom ROI method using regex over it |
| PAYERS\_LAST\_NAME | - | From custom ROI method using regex over it |
| PAYERS\_ADDRESS | - | From custom ROI method using regex over it |
| PAYERS\_CITY | - | From custom ROI method using regex over it |
| PAYERS\_STATE | - | From custom ROI method using regex over it |
| PAYERS\_ZIP\_CODE | - | From custom ROI method using regex over it |
| RECIPIENTS\_FIRST\_NAME | - | From custom ROI method using regex over it |
| RECIPIENTS\_MIDDLE\_INITIAL | - | From custom ROI method using regex over it |
| RECIPIENTS\_LAST\_NAME | - | From custom ROI method using regex over it |
| RECIPIENTS\_ADDRESS | - | From custom ROI method using regex over it |
| RECIPIENTS\_CITY | - | From custom ROI method using regex over it |
| RECIPIENTS\_STATE | - | From custom ROI method using regex over it |
| RECIPIENTS\_ZIP\_CODE | - | From custom ROI method using regex over it |
| PAYERS\_TIN | - | Using Form Parser key value pairs /Custom ROI method using Regex. |
| RECIPIENTS\_TIN | - | Using Form Parser key value pairs /Custom ROI method using Regex. |
| GROSS\_DISTRIBUTION | - | Using Form Parser key value pairs /Custom ROI method using Regex. |
| TAXABLE\_AMOUNT | - | Using Form Parser key value pairs /Custom ROI method using Regex. |
| CAPITAL\_GAIN | - | Using Form Parser key value pairs /Custom ROI method using Regex. |

#### 3.1.3.22 Form 4562

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| TAX\_YEAR | - | Using regex over tokens. |
| NAMES\_SHOWN\_ON\_RETURN | - | Using Form Parser key value pairs. |
| IDENTIFYING\_NUMBER | - | Using Form Parser key value pairs. |
| TOTAL\_MILES\_VEHICLE\_1 | - | Using Form Parser Tables |
| TOTAL\_MILES\_VEHICLE\_2 | - | Using Form Parser Tables |
| TOTAL\_MILES\_VEHICLE\_3 | - | Using Form Parser Tables |
| TOTAL\_MILES\_VEHICLE\_4 | - | Using Form Parser Tables |
| TOTAL\_MILES\_VEHICLE\_5 | - | Using Form Parser Tables |
| TOTAL\_MILES\_VEHICLE\_6 | - | Using Form Parser Tables |
| TOTAL | - | From custom ROI method using regex over it |

#### 

#### 3.1.3.23 Schedule G Form 1120

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| TAX\_YEAR | - | Using regex over tokens. |
| NAME | - | Using Form Parser key value pairs. |
| EMPLOYER\_IDENTIFICATION\_NUMBER | - | Using Form Parser key value pairs. |
| NAME\_OF\_INDIVIDUAL\_OR\_ESTATE\_1 | - | Using Form Parser Tables |
| NAME\_OF\_INDIVIDUAL\_OR\_ESTATE\_2 | - | Using Form Parser Tables |
| NAME\_OF\_INDIVIDUAL\_OR\_ESTATE\_3 | - | Using Form Parser Tables |
| NAME\_OF\_INDIVIDUAL\_OR\_ESTATE\_4 | - | Using Form Parser Tables |
| NAME\_OF\_INDIVIDUAL\_OR\_ESTATE\_5 | - | Using Form Parser Tables |
| NAME\_OF\_INDIVIDUAL\_OR\_ESTATE\_6 | - | Using Form Parser Tables |
| NAME\_OF\_INDIVIDUAL\_OR\_ESTATE\_7 | - | Using Form Parser Tables |
| NAME\_OF\_INDIVIDUAL\_OR\_ESTATE\_8 | - | Using Form Parser Tables |
| NAME\_OF\_INDIVIDUAL\_OR\_ESTATE\_9 | - | Using Form Parser Tables |
| NAME\_OF\_INDIVIDUAL\_OR\_ESTATE\_10 | - | Using Form Parser Tables |
| IDENTIFYING\_NUMBER\_1 | - | Using Form Parser Tables |
| IDENTIFYING\_NUMBER\_2 | - | Using Form Parser Tables |
| IDENTIFYING\_NUMBER\_3 | - | Using Form Parser Tables |
| IDENTIFYING\_NUMBER\_4 | - | Using Form Parser Tables |
| IDENTIFYING\_NUMBER\_5 | - | Using Form Parser Tables |
| IDENTIFYING\_NUMBER\_6 | - | Using Form Parser Tables |
| IDENTIFYING\_NUMBER\_7 | - | Using Form Parser Tables |
| IDENTIFYING\_NUMBER\_8 | - | Using Form Parser Tables |
| IDENTIFYING\_NUMBER\_9 | - | Using Form Parser Tables |
| IDENTIFYING\_NUMBER\_10 | - | Using Form Parser Tables |
| PERCENTAGE\_OWNED\_IN\_VOTING\_STOCK\_1 | - | Using Form Parser Tables |
| PERCENTAGE\_OWNED\_IN\_VOTING\_STOCK\_2 | - | Using Form Parser Tables |
| PERCENTAGE\_OWNED\_IN\_VOTING\_STOCK\_3 | - | Using Form Parser Tables |
| PERCENTAGE\_OWNED\_IN\_VOTING\_STOCK\_4 | - | Using Form Parser Tables |
| PERCENTAGE\_OWNED\_IN\_VOTING\_STOCK\_5 | - | Using Form Parser Tables |
| PERCENTAGE\_OWNED\_IN\_VOTING\_STOCK\_6 | - | Using Form Parser Tables |
| PERCENTAGE\_OWNED\_IN\_VOTING\_STOCK\_7 | - | Using Form Parser Tables |
| PERCENTAGE\_OWNED\_IN\_VOTING\_STOCK\_8 | - | Using Form Parser Tables |
| PERCENTAGE\_OWNED\_IN\_VOTING\_STOCK\_9 | - | Using Form Parser Tables |
| PERCENTAGE\_OWNED\_IN\_VOTING\_STOCK\_10 | - | Using Form Parser Tables |

#### 

**Method to create lines of text using tokens:**

1. Pick the first token .
2. Draw a boundary (**boundary 1**) parallel to the top line of the bbox of the token and at a distance d above it.
3. Similarly draw a boundary (**boundary 2**) parallel to the bottom line of the bbox of the token and at a distance d below it.
4. Search for tokens that lie within the region bounded by boundary 1 and boundary 2.
5. Every time a token is added to the region, update the boundaries with the top right point and top left point of the tokens in the region as it helps countering the rotation or shear present in documents..
6. Keep repeating until all the tokens are assigned to some line.

**Method to group tokens nearby in a line of text :**

1. Pick the first token, calculate the average width per character using:

**#characters in token/width of token.**

1. Assign tokens within distance x times average per character of the first token to the same group.
2. Keep repeating until all the tokens are assigned to some group.

**Method to extract addresses :**

1. Create lines using the lines method.
2. Create nearby chunks of each line.
3. In chunks match for city , state and zip code regex.
4. If regex matches, collate chunks directly above the city state zip code chunk recursively and stop collating if you find a key in one of the lines nearby the relevant chunk.
5. All the collated chunks are the part of the address.

#### 3.1.3.24 Flood Insurance Policy

#### 

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| INSURED\_FIRST\_NAMES |  | * Form Parser Key value pairs. * Sometimes names are present on top of addresses. To extract those, extract addresses using the address extraction method described above and get chunks on top of address blocks to extract names. |
| INSURED\_MIDDLE\_INITIAL |  | 1. Form Parser Key value pairs. 2. Sometimes names are present on top of addresses. To extract those, extract addresses using the address extraction method described above and get chunks on top of address blocks to extract names. |
| INSURED\_LAST\_NAMES |  | * Form Parser Key value pairs. * Sometimes names are present on top of addresses. To extract those, extract addresses using the address extraction method described above and get chunks on top of address blocks to extract names. |
| COINSURED\_FIRST\_NAMES |  | * Form Parser Key value pairs. * Sometimes names are present on top of addresses. To extract those, extract addresses using the address extraction method described above and get chunks on top of address blocks to extract names. |
| COINSURED\_MIDDLE\_INITIAL |  | * Form Parser Key value pairs. * Sometimes names are present on top of addresses. To extract those, extract addresses using the address extraction method described above and get chunks on top of address blocks to extract names. |
| COINSURED\_LAST\_NAMES |  | * Form Parser Key value pairs. * Sometimes names are present on top of addresses. To extract those, extract addresses using the address extraction method described above and get chunks on top of address blocks to extract names. |
| INSURANCE\_COMPANY\_NAME |  | * Using tokens on top portion of the page and searching from a list of available options |
| PROPERTY\_ADDRESS |  | * Using method to extract addresses |
| PROPERTY\_CITY |  | * Using method to extract addresses |
| PROPERTY\_STATE |  | * Using method to extract addresses |
| PROPERTY\_ZIP\_CODE |  | * Using method to extract addresses |
| POLICY\_EFFECTIVE\_DATE |  | * Regex, * Form parser key value pairs * using dateparser around a found key. |
| POLICY\_EXPIRATION\_DATE |  | * Regex, * Form parser key value pairs * using dateparser around a found key. |
| DWELLING\_COVERAGE |  | * From custom lines using regex |
| MORTAGEE\_CLAUSE |  | * Using method to extract addresses |
| PREMIUM |  | * Using form parser key value pairs * custom lines using regex around a found key.. |
| POLICY\_TYPE |  | * Using form parser key value pairs * custom lines. |
| REPLACEMENT\_COST |  | * Using form parser key value pairs * custom lines with regex. * Tables |
| REPLACEMENT\_COST\_PERCENTAGE |  | * Using form parser key value pairs * custom lines with regex. |
| MAILING\_ADDRESS |  | * Using method to extract addresses |
| POLICY\_NUMBER |  | * Using form parser key value pairs * custom lines. |

#### 

#### 3.1.3.25 Property Insurance Policy(HO3/HO6)

#### 

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| INSURED\_FIRST\_NAMES |  | * Form Parser Key value pairs. * Sometimes names are present on top of addresses. To extract those, extract addresses using the address extraction method described above and get chunks on top of address blocks to extract names. |
| INSURED\_MIDDLE\_INITIAL |  | 1. Form Parser Key value pairs. 2. Sometimes names are present on top of addresses. To extract those, extract addresses using the address extraction method described above and get chunks on top of address blocks to extract names. |
| INSURED\_LAST\_NAMES |  | * Form Parser Key value pairs. * Sometimes names are present on top of addresses. To extract those, extract addresses using the address extraction method described above and get chunks on top of address blocks to extract names. |
| COINSURED\_FIRST\_NAMES |  | * Form Parser Key value pairs. * Sometimes names are present on top of addresses. To extract those, extract addresses using the address extraction method described above and get chunks on top of address blocks to extract names. |
| COINSURED\_MIDDLE\_INITIAL |  | * Form Parser Key value pairs. * Sometimes names are present on top of addresses. To extract those, extract addresses using the address extraction method described above and get chunks on top of address blocks to extract names. |
| COINSURED\_LAST\_NAMES |  | * Form Parser Key value pairs. * Sometimes names are present on top of addresses. To extract those, extract addresses using the address extraction method described above and get chunks on top of address blocks to extract names. |
| INSURANCE\_COMPANY\_NAME |  | * Using tokens on top portion of the page and searching from a list of available options |
| PROPERTY\_ADDRESS |  | * Using method to extract addresses |
| PROPERTY\_CITY |  | * Using method to extract addresses |
| PROPERTY\_STATE |  | * Using method to extract addresses |
| PROPERTY\_ZIP\_CODE |  | * Using method to extract addresses |
| POLICY\_EFFECTIVE\_DATE |  | * Regex, * Form parser key value pairs * using dateparser around a found key. |
| POLICY\_EXPIRATION\_DATE |  | * Regex, * Form parser key value pairs * using dateparser around a found key. |
| DWELLING\_COVERAGE |  | * From custom lines using regex |
| MORTAGEE\_CLAUSE |  | * Using method to extract addresses |
| PREMIUM |  | * Using form parser key value pairs * custom lines using regex around a found key.. |
| POLICY\_TYPE |  | * Using form parser key value pairs * custom lines. |
| REPLACEMENT\_COST |  | * Using form parser key value pairs * custom lines with regex. * Tables |
| REPLACEMENT\_COST\_PERCENTAGE |  | * Using form parser key value pairs * custom lines with regex. |
| MAILING\_ADDRESS |  | * Using method to extract addresses |
| POLICY\_NUMBER |  | * Using form parser key value pairs * custom lines. |

#### 

#### 3.1.3.26 Schedule C Form 1040

#### 

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| TAX\_YEAR |  | * Using ROI bounded by **OMB** from above and **Sequence** from below to get tokens in the region. * Match regex to extract year from token in the region. |
| NAME\_OF\_PROPRIETOR |  | * Create lines using custom lines logic and get nearby chunks. * Search for the chunk just below “Name of proprietor”. |
| BUSINESS\_NAME |  | * Create lines using custom lines logic and get nearby chunks. * Search for the chunk just below “business name”. |
| BUSINESS\_ADDRESS |  | * Create lines from custom logic. * Search for key “business address” in lines and get whatever is present in front of it as value |
| BUSINESS\_CITY |  | * Create lines from custom logic. * Search for key “city, town” in lines and get whatever is present in front of it as value * Use separator logic to separate city, state and zipcode. |
| BUSINESS\_STATE |  | * Create lines from custom logic. * Search for key “city, town” in lines and get whatever is present in front of it as value * Use separator logic to separate city, state and zipcode. |
| BUSINESS\_ZIP\_CODE |  | * Create lines from custom logic. * Search for key “city, town” in lines and get whatever is present in front of it as value * Use separator logic to separate city, state and zipcode. |
| EMPLOYER\_IDENTIFICATION\_NUMBER |  | * Extract from Form Parser key value pairs. * If not extracted , create custom lines and in each line group nearby chunks of text. * Search for all the chunks just below the “employer id number” chunk. |
| SOCIAL\_SECURITY\_NUMBER |  | * Extract from Form Parser key value pairs. * If not extracted , create custom lines and in each line group nearby chunks of text. * Search for all the chunks just below the “social security number” chunk. |
| OTHER\_INCOME |  | * Create lines using custom logic * Search for “6 Other Income” or “6\n” in line text. * Match value in front of “6\n” to a number regex to extract value. |
| CAR\_AND\_TRUCK\_EXPENSES |  | * Create lines using custom logic * Search for “9 Car and Track” or “9\n” in line text. * Match value in front of “9\n” to a number regex to extract value. |
| DEPLETION |  | * Create lines using custom logic * Search for “12 Depletion” or “12\n” in line text. * Match value in front of “12\n” to a number regex to extract value. |
| DEPRECIATION |  | * Create lines using custom logic * Search for “13 Depreciation” or “13\n” in line text. * Match value in front of “13\n” to a number regex to extract value. |
| DEDUCTIBLE\_MEALS |  | * Create lines using custom logic * Search for “24b\n” in line text. * Match value in front of “24b\n” to a number regex to extract value. |
| EXPENSES\_FOR\_BUSINESS\_USE |  | * Create lines using custom logic * Search for “30\n” in line text. * Match value in front of “30\n” to a number regex to extract value. |
| NET\_PROFIT\_OR\_LOSS |  | * Create lines using custom logic * Search for “31\n” in line text. * Match value in front of “31\n” to a number regex to extract value. |
| BUSINESS\_MILES |  | * Create lines using custom logic * Search for 'a\nBusiness' or 'a Business' or 'aBusiness' in line text. * Match value in front of “Business” to a number regex to extract value. |

#### 

#### 3.1.3.27 Verification Of Assets

#### 

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| BANK\_NAME |  | * Create custom line text. * Search for anything before “current balance”. |
| FIRST\_NAME |  | * Create custom line text. * Search for key “account name(s)”. * Extract everything in front of that key and clean unwanted strings by splitting with strings like “Deposits”. * Separate holder coholder names using split by “AND”, “/” etc. |
| MIDDLE\_INITIAL |  | * Create custom line text. * Search for key “account name(s)”. * Extract everything in front of that key and clean unwanted strings by splitting with strings like “Deposits”. * Separate holder coholder names using split by “AND”, “/” etc. |
| LAST\_NAME |  | * Create custom line text. * Search for key “account name(s)”. * Extract everything in front of that key and clean unwanted strings by splitting with strings like “Deposits”. * Separate holder coholder names using split by “AND”, “/” etc. |
| TYPE\_OF\_ACCOUNT |  | * Extract from form parser key value. * If that fails extract using custom line text. |
| ENDING\_BALANCE\_OF\_ACCOUNT |  | * Extract from Form Parser keys. * If the form parser fails , create custom line texts and search for the amount in front of the key. |
| ACCOUNT\_NUMBER |  | * Extract from form parser key value. * If that fails extract using custom line text. |
| BEGINNING\_DATE\_OF\_STATEMENT |  | * Create custom lines and search for key “Transaction History” in line text. * Take everything after the key in line text and match a regex to extract dates. |
| ENDING\_DATE\_OF\_STATEMENT |  | * Create custom lines and search for key “Transaction History” in line text. * Take everything after the key in line text and match a regex to extract dates. |
| AVAILABLE\_BALANCE |  | * Extract from Form Parser keys. * If the form parser fails , create custom line texts and search for the amount in front of the key. |
| COACCOUNT\_HOLDER\_FIRST\_NAME |  | * Create custom line text. * Search for key “account name(s)”. * Extract everything in front of that key and clean unwanted strings by splitting with strings like “Deposits”. * Separate holder coholder names using split by “AND”, “/” etc. |
| COACCOUNT\_HOLDER\_MIDDLE\_INITIAL |  | * Create custom line text. * Search for key “account name(s)”. * Extract everything in front of that key and clean unwanted strings by splitting with strings like “Deposits”. * Separate holder coholder names using split by “AND”, “/” etc. |
| COACCOUNT\_HOLDER\_LAST\_NAME |  | * Create custom line text. * Search for key “account name(s)”. * Extract everything in front of that key and clean unwanted strings by splitting with strings like “Deposits”. * Separate holder coholder names using split by “AND”, “/” etc. |

#### 

#### 3.1.3.28 Social Security Award Letter

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| FIRST\_NAME, MIDDLE\_INITIAL  LAST\_NAME |  | * Extract from form parser keys. * If the form parser misses, create lines using tokens and search for key in lines and extract from there. * Sometimes no key is present but names are present on top of addresses. To extract those, extract addresses using the address extraction method described above and get chunks on top of address blocks to extract names. |
| BEGINING\_DATE\_OF\_BENEFITS |  | 1. Create lines using tokens and search for segments like '**monthly amount**', '**on or about**', '**in your check on'**, '**as of'**. 2. Extract date from remaining line text using date parsers search date method |
| MONTHLY\_AWARDED\_AMOUNT |  | 1. Create lines using tokens and search for segments like '**beginning**', '**monthly social security benefit**', '**monthly benefit'**, '**monthly social security payment'**, **‘monthly social security benefit’**. 2. Extract numbers using regex to extract amounts and pick the biggest one in the remaining line text. |

#### 

#### 3.1.3.29 Tax Transcripts

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| REQUEST\_DATE |  | * From custom ROI method using regex over it |
| TAX\_PERIOD\_ENDING\_DATE |  | * From custom ROI method using regex over it |
| SOCIAL\_SECURITY\_NUMBER |  | * From custom ROI method using regex over it |
| SPOUSE\_SOCIAL\_SECURITY\_NUMBER |  | * From custom ROI method using regex over it |
| FIRST\_NAMES\_SHOWN\_ON\_RETURN |  | * From custom ROI method using regex over it |
| MIDDLE\_INITIALS\_SHOWN\_ON\_RETURN |  | * From custom ROI method using regex over it |
| LAST\_NAMES\_SHOWN\_ON\_RETURN |  | * From custom ROI method using regex over it |
| ADDRESS | - | * From custom ROI method using regex over it |
| CITY |  | * From custom ROI method using regex over it |
| STATE |  | * From custom ROI method using regex over it |
| ZIP\_CODE |  | * From custom ROI method using regex over it |
| RECEIVED\_DATE |  | * From custom ROI method using regex over it |
| WAGES\_SALARIES\_TIPS\_ETC |  | * From custom ROI method using regex over it |
| ORDINARY\_DIVIDEND\_INCOME\_SCH\_B |  | * From custom ROI method using regex over it |
| ALIMONY\_RECEIVED |  | * From custom ROI method using regex over it |
| BUSINESS\_INCOME\_OR\_LOSS\_SCHEDULE\_C |  | * From custom ROI method using regex over it |
| CAPITAL\_GAIN\_OR\_LOSS\_SCHEDULE\_D |  | * From custom ROI method using regex over it |
| TOTAL\_PENSIONS\_AND\_ANNUITIES |  | * From custom ROI method using regex over it |
| TAXABLE\_PENSION\_ANNUITY\_AMOUNT |  | * From custom ROI method using regex over it |
| ADDITIONAL\_INCOME |  | * From custom ROI method using regex over it |
| RENT\_ROYALTY\_PARTNERSHIP\_ESTATE\_SCHEDULE\_E |  | * From custom ROI method using regex over it |
| ESTATE\_TRUST\_INCOME\_LOSS\_PER\_COMPUTER |  | * From custom ROI method using regex over it |
| FARM\_INCOME\_OR\_LOSS\_SCHEDULE\_F |  | * From custom ROI method using regex over it |
| UNEMPLOYMENT\_COMPENSATION |  | * From custom ROI method using regex over it |
| TOTAL\_SOCIAL\_SECURITY\_  BENEFITS |  | * From custom ROI method using regex over it |
| OTHER\_INCOME |  | * From custom ROI method using regex over it |
| ALIMONY\_PAID |  | * From custom ROI method using regex over it |
| REFUND\_AMOUNT |  | * From custom ROI method using regex over it |
| GROSS\_SCHEDULE\_B\_INTEREST |  | * From custom ROI method using regex over it |
| GROSS\_SCHEDULE\_B\_DIVIDENDS |  | * From custom ROI method using regex over it |
| SCH\_C\_SOCIAL\_SECURITY\_NUMBER |  | * From custom ROI method using regex over it |
| SCH\_C\_EMPLOYER\_ID\_NUMBER |  | * From custom ROI method using regex over it |
| SCH\_C\_BUSINESS\_NAME |  | * From custom ROI method using regex over it |
| SCH\_C\_GROSS\_RECEIPTS\_OR\_SALES |  | * From custom ROI method using regex over it |
| SCH\_C\_SCHEDULE\_C\_FORM\_1099\_REQUIRED |  | * From custom ROI method using regex over it |
| SCH\_C\_CAR\_AND\_TRUCK\_EXPENSES |  | * From custom ROI method using regex over it |
| SCH\_C\_DEPRECIATION |  | * From custom ROI method using regex over it |
| SCH\_C\_TRAVEL |  | * From custom ROI method using regex over it |
| SCH\_C\_MEALS\_AND\_ENTERTAINMENT |  | * From custom ROI method using regex over it |
| SCH\_D\_NET\_SHORT\_TERM\_GAIN\_LOSS |  | * From custom ROI method using regex over it |
| SCH\_D\_NET\_LONG\_TERM\_GAIN\_LOSS |  | * From custom ROI method using regex over it |
| SCH\_E\_TOTAL\_RENTS\_RECEIVED |  | * From custom ROI method using regex over it |
| SCH\_E\_TOTAL\_ROYALTIES\_RECEIVED |  | * From custom ROI method using regex over it |
| PARTNERSHIP\_INCOME |  | * From custom ROI method using regex over it |
| PARTNERSHIP\_LOSS |  | * From custom ROI method using regex over it |
| ESTATE\_AND\_TRUST\_INCOME |  | * From custom ROI method using regex over it |
| ESTATE\_AND\_TRUST\_LOSS |  | * From custom ROI method using regex over it |

#### 3.1.3.30 Verification Of Employment

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| EMPLOYER\_NAME |  | * Using form parser key value pairs * From custom ROI method using regex over it |
| SIGNATURE\_OF\_LENDER |  | * Using form parser key value pairs * From custom ROI method |
| DATE |  | * From custom ROI method using regex over it |
| NAME\_AND\_ADDRESS\_OF\_APPLICANT |  | * Using form parser key value pairs and using regex over it * From custom ROI method using regex over it |
| APPLICANTS\_DATE\_OF\_EMPLOYMENT |  | * From custom ROI method using regex over it |
| PRESENT\_POSITION |  | * Using form parser key value pairs * From custom ROI method |
| PROBABILITY\_OF\_CONTINUED\_EMPLOYMENT |  | * Using form parser key value pairs * From custom ROI method |
| CURRENT\_GROSS\_BASE\_PAY\_CHECK\_BOX\_MONTHLY | - | * From custom ROI method |
| CURRENT\_GROSS\_BASE\_PAY\_CHECK\_BOX\_ANNUAL |  | * From custom ROI method |
| CURRENT\_GROSS\_BASE\_PAY\_CHECK\_BOX\_OTHER |  | * From custom ROI method |
| CURRENT\_GROSS\_BASE\_PAY\_CHECK\_BOX\_WEEKLY |  | * From custom ROI method |
| CURRENT\_GROSS\_BASE\_PAY\_CHECK\_BOX\_HOURLY |  | * From custom ROI method |
| CURRENT\_GROSS\_BASE\_PAY |  | * From custom ROI method |
| YTD\_THRU |  | * From custom ROI method |
| BASE\_PAY\_YTD |  | * From custom ROI method |
| OVERTIME\_YTD |  | * From custom ROI method |
| COMMISSIONS\_YTD |  | * From custom ROI method |
| BONUS\_YTD |  | * From custom ROI method |
| TOTAL\_YTD |  | * From custom ROI method |
| PAST\_YEAR\_1\_BASE\_PAY |  | * From custom ROI method |
| PAST\_YEAR\_1\_OVERTIME |  | * From custom ROI method |
| PAST\_YEAR\_1\_COMMISSIONS |  | * From custom ROI method |
| PAST\_YEAR\_1\_BONUS |  | * From custom ROI method |
| PAST\_YEAR\_1\_TOTAL |  | * From custom ROI method |
| PAST\_YEAR\_2\_BASE\_PAY |  | * From custom ROI method |
| PAST\_YEAR\_2\_OVERTIME |  | * From custom ROI method |
| PAST\_YEAR\_2\_COMMISSIONS |  | * From custom ROI method |
| PAST\_YEAR\_2\_BONUS |  | * From custom ROI method |
| PAST\_YEAR\_2\_TOTAL |  | * From custom ROI method |
| PAY\_GRADE |  | * From custom ROI method |
| BASE\_PAY |  | * Using form parser key value pairs * From custom ROI method |
| RATIONS |  | * Using form parser key value pairs * From custom ROI method |
| FIGHT\_OR\_HAZARD |  | * Using form parser key value pairs * From custom ROI method |
| CLOTHING |  | * Using form parser key value pairs * From custom ROI method |
| QUARTERS |  | * Using form parser key value pairs * From custom ROI method |
| PRO\_PAY |  | * Using form parser key value pairs * From custom ROI method |
| OVERSEAS\_OR\_COMBAT |  | * Using form parser key value pairs * From custom ROI method |
| VARIABLE\_HOUSING\_ALLOWANCE |  | * Using form parser key value pairs * From custom ROI method |
| OVERTIME\_BONUS\_CONTINUANCE\_LIKELY\_OVERTIME\_YES |  | * From custom ROI method |
| OVERTIME\_BONUS\_CONTINUANCE\_LIKELY\_BONUS\_YES |  | * From custom ROI method |
| OVERTIME\_BONUS\_CONTINUANCE\_LIKELY\_OVERTIME\_NO |  | * From custom ROI method |
| OVERTIME\_BONUS\_CONTINUANCE\_LIKELY\_BONUS\_NO |  | * From custom ROI method |
| AVERAGE\_HOURS\_PER\_WEEK |  | * Using form parser key value pairs * From custom ROI method |
| DATE\_OF\_NEXT\_PAY\_INCREASE |  | * From custom ROI method |
| PROJECTED\_AMOUNT\_OF\_NEXT\_PAY\_INCREASE |  | * Using form parser key value pairs * From custom ROI method |
| DATE\_OF\_LAST\_PAY\_INCREASE |  | * From custom ROI method |
| AMOUNT\_OF\_LAST\_PAY\_INCREASE |  | * Using form parser key value pairs * From custom ROI method using regex over it |
| DATE\_HIRED |  | * From custom ROI method using regex over it |
| DATE\_TERMINATED |  | * From custom ROI method using regex over it |
| BASE |  | * From custom ROI method using regex over it |
| OVERTIME |  | * From custom ROI method using regex over it |
| COMMISSIONS |  | * From custom ROI method using regex over it |
| BONUS |  | * From custom ROI method using regex over it |

#### 3.1.3.31 Residential Purchase Agreement

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| Contract Effective Date | - | * Extract from Form Parser Form Fields - Values. * Use regex to extract the date. |
| Purchase Price | - | * Used a pre-defined dictionary to identify lines containing purchase price. * Used regex on form parser detected values |
| Seller Concessions |  | * Used a pre-defined dictionary to identify lines containing purchase price. * Used regex on form parser detected values |
| Agent Credits |  | * Extract from Form Parser Form Fields - Values. |
| Street Address |  | * Extract from Form Parser Form Fields - Values. |
| City |  | * Extract address from Form Parser Form Fields - Values. * Use zip code to get the City name. |
| State |  | * Extract address from Form Parser Form Fields - Values. * Use zip code to get the State name. |
| Zip Code |  | * Extract address from Form Parser Form Fields - Values. * Identify zip code using RegEx |
| Buyer First Name 1 |  | * Extract full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Buyer Middle Initial 1 |  | * Extract full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Buyer Last Name 1 |  | * Extract full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Buyer First Name 2 |  | * Extract full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Buyer Middle Initial 2 |  | * Extract full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Buyer Last Name 2 |  | * Extract full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Signatures |  |  |
| Seller First Name 1 |  | * Extract full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Seller Middle Initial 1 |  | * Extract full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Seller Last Name 1 |  | * Extract full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Seller First Name 2 |  | * Extract full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Seller Middle Initial 2 |  | * Extract full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Seller Last Name 2 |  | * Extract full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Buyers Real Estate Broker Agency Name |  | * Extract from Form Parser Form Fields - Values. |
| Buyers Agents Full Name |  | * Extract agents full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Buyers Agents Middle Initial Name |  | * Extract agents full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Buyers Agents Last Name |  | * Extract agents full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Buyers Agency NMLS |  | * Extract agents full name from Form Parser Form Fields - Values. |
| Buyers Agency State License |  | * Extract agents full name from Form Parser Form Fields - Values. |
| Buyers Agency Street Address |  | * Extract from Form Parser Form Fields - Values. |
| Buyers Agency City |  | * Extract address from Form Parser Form Fields - Values. * Use zip code to get the City name. |
| Buyers Agency State |  | * Extract address from Form Parser Form Fields - Values.Use zip code to get the State name. |
| Buyers Agency Zip Code |  | * Extract address from Form Parser Form Fields - Values. * Identify zip code using RegEx |
| Sellers Real Estate Broker Agency Name |  | * Extract from Form Parser Form Fields - Values. |
| Sellers Agents Full Name |  | * Extract agents full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Sellers Agents Middle Initial Name |  | * Extract agents full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Sellers Agents Last Name |  | * Extract agents full name from Form Parser Form Fields - Values. * Split Name into First Name, Middle Initial and Last Name |
| Sellers Agency NMLS |  | * Extract from Form Parser Form Fields - Values. |
| Sellers Agency State License |  | * Extract from Form Parser Form Fields - Values. |
| Sellers Agency Street Address |  | * Extract from Form Parser Form Fields - Values. |
| Sellers Agency City |  | * Extract address from Form Parser Form Fields - Values. * Use zip code to get the City name. |
| Sellers Agency State |  | * Extract address from Form Parser Form Fields - Values.Use zip code to get the State name. |
| Sellers Agency Zip Code |  | * Extract address from Form Parser Form Fields - Values. * Identify zip code using RegEx |
| Earnest Money Deposit Amount |  | * Extract from Form Parser Form Fields - Values. |

#### 3.1.3.32 Schedule K-1 Form 1120S:

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| BEGINNING\_DATE |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| ENDING\_DATE |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| SHAREHOLDER\_ID |  | * Text and Bounding Box extracted from Form Parser |
| NET\_RENTAL\_REAL\_ESTATE\_INCOME\_LOSS |  | * Text and Bounding Box extracted from Form Parser |
| OTHER\_NET\_RENTAL\_INCOME\_LOSS |  | * Text and Bounding Box extracted from Form Parser |
| SHAREHOLDER\_PERCENTAGE\_OF\_OWNERSHIP |  | * Text and Bounding Box extracted from Form Parser |
| FINAL\_K1 |  | * Text and Bounding Box extracted from Form Parser |
| AMENDED\_K1 |  | * Text and Bounding Box extracted from Form Parser |
| ORDINARY\_BUSINESS\_INCOME\_LOSS |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| EMPLOYER\_IDENTIFICATION\_NUMBER |  | * Text and Bounding Box extracted from Form Parser |
| SHAREHOLDER\_NAME |  | * Text is extracted as a single block for Shareholder information, which is then split and part of Shareholder Name is assigned here * We first calculate the ROI and if we aren't able to pick up the text or bounding box from ROI then we use what Form Parser provides us with |
| SHAREHOLDER\_ADDRESS |  | * Text is extracted as a single block for Shareholder information, which is then split and part of Shareholder Address is assigned here * We first calculate the ROI and if we aren't able to pick up the text or bounding box from ROI then we use what Form Parser provides us with |
| SHAREHOLDER\_CITY |  | * Text is extracted as a single block for Shareholder information, which is then split and part of Shareholder City is assigned here * We first calculate the ROI and if we aren't able to pick up the text or bounding box from ROI then we use what Form Parser provides us with |
| SHAREHOLDER\_STATE |  | * Text is extracted as a single block for Shareholder information, which is then split and part of Shareholder State is assigned here * We first calculate the ROI and if we aren't able to pick up the text or bounding box from ROi then we use what Form Parser provides us with |
| SHAREHOLDER\_ZIP\_CODE |  | * Text is extracted as a single block for Shareholder information, which is then split and part of Shareholder Zip Code is assigned here * We first calculate the ROI and if we aren't able to pick up the text or bounding box from ROi then we use what Form Parser provides us with |
| CORPORATION\_NAME |  | * Text is extracted as a single block for Corporation information, which is then split and part of Corporation Name is assigned here * We first calculate the ROI and if we aren't able to pick up the text or bounding box from ROi then we use what Form Parser provides us with |
| CORPORATION\_ADDRESS |  | * Text is extracted as a single block for Corporation information, which is then split and part of Corporation Address is assigned here * We first calculate the ROI and if we aren't able to pick up the text or bounding box from ROi then we use what Form Parser provides us with |
| CORPORATION\_CITY |  | * Text is extracted as a single block for Corporation information, which is then split and part of Corporation City is assigned here * We first calculate the ROI and if we aren't able to pick up the text or bounding box from ROi then we use what Form Parser provides us with |
| CORPORATION\_STATE |  | * Text is extracted as a single block for Corporation information, which is then split and part of Corporation State is assigned here * We first calculate the ROI and if we aren't able to pick up the text or bounding box from ROi then we use what Form Parser provides us with |
| CORPORATION\_ZIP\_CODE |  | * Text is extracted as a single block for Corporation information, which is then split and part of Corporation Zip Code is assigned here * We first calculate the ROI and if we aren't able to pick up the text or bounding box from ROi then we use what Form Parser provides us with |
| TAX\_YEAR |  | * Tax year value is calculated from the OCR text of the document by regex * Bounding box is calculated by using ROI |
| ITEMS\_AFFECTING\_SHAREHOLDER\_BASIS\_1 |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| REFERENCE\_CODE\_1 |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| ITEMS\_AFFECTING\_SHAREHOLDER\_BASIS\_2 |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| REFERENCE\_CODE\_2 |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| REFERENCE\_CODE\_3 |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| ITEMS\_AFFECTING\_SHAREHOLDER\_BASIS\_3 |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| REFERENCE\_CODE\_4 |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| ITEMS\_AFFECTING\_SHAREHOLDER\_BASIS\_4 |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| REFERENCE\_CODE\_5 |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| ITEMS\_AFFECTING\_SHAREHOLDER\_BASIS\_5 |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| REFERENCE\_CODE\_6 |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |
| ITEMS\_AFFECTING\_SHAREHOLDER\_BASIS\_6 |  | * ROI is used to find the expected location in the page. * Text and BoundingBox extracted from ROI |

#### 3.1.3.33 Pension Award Letter

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| NAME |  |  |
| ADDRESS |  |  |
| DATE |  |  |
| GROSS\_AMOUNT |  |  |
| NET\_AMOUNT |  |  |
| EMPLOYER |  |  |

#### 3.1.3.34 Schedule D Form 1040

| **Fields (to be extracted from the document)** | **Issues (if any)** | **Extraction method** |
| --- | --- | --- |
| NAME |  |  |
| SOCIAL\_SECURITY\_NUMBER |  |  |
| NET\_SHORT\_TERM\_CAPITAL\_GAIN\_OR\_LOSS |  |  |
| NET\_LONG\_TERM\_CAPITAL\_GAIN\_OR\_LOSS |  |  |
| CAPITAL\_LOSS\_DEDUCTION |  |  |

## 3.2 Pipeline Output

The final solution output is a JSON file containing key entities from the respective document types.

### 3.2.1 Final JSON Format

As per the API documentation link -   
<https://docs.google.com/document/d/1NxTGUWSbvAIV81GAlMkIr6Afyqfar3z3naSssbF_ZAs/edit#>  
  
3.2.2 Confidence Value Reporting - Historical

# 4. Unit Tests

Unit test cases are only defined for functions prone to break the pipeline. In the following tables, test cases for each field to be extracted from the document **(keys)** are given with corresponding sample input data.

## 4.1 Pension Award Letter

| **Test Cases** | **Sample Input** |
| --- | --- |
| Test\_net\_amount | [  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data1.json", ["200.00"]),  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data2.json", ["190.00"]),  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data3.json", [None]),  ], |
| Test\_gross\_amount | [  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data1.json", ["200.00"]),  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data2.json", ["200.00"]),  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data3.json", ["3,313.63"]),  ], |
| Test\_employee | [  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data1.json", ["Sarah Smith (123) 456-7890 NO\_REPLY@EXAMPLE.COM","123 YOUR STREET YOUR CITY, ST 12345"]),  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data2.json", ["Sarah Smith (123) 456-7890 NO\_REPLY@EXAMPLE.COM","123 YOUR STREET YOUR CITY, ST 12345"]),  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data3.json", ["\u00b9Permanente Building 100N.E. 503-813-3000","123 Street, Suite XX Portland, OR 92332-2099"]),  ], |
| Test\_employer | [  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data1.json", ["GE HR SERVICES "]),  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data2.json", ["New England Health Care Employees Pension Fund"]),  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data3.json", ["Northwest Permanente PC, Physicians & Surgeons"]),  ], |
| Test\_effective\_date | [  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data1.json", [None]),  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data2.json", ["June 1 2019"]),  (f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data3.json", [None]),  ], |
| Test\_get\_pal\_result\_main | (  f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data1.json",  {  "ADDRESS": "123 YOUR STREET YOUR CITY, ST 12345",  "NAME": "Sarah Smith (123) 456-7890 NO\_REPLY@EXAMPLE.COM",  "GROSS\_AMOUNT": 200,  "NET\_AMOUNT": 200,  "EMPLOYER": "GE HR SERVICES ",  "DATE": None,  },  ), |
| Test\_start\_processing | (  f"{Path(\_\_file\_\_).parent}/fp\_json/pal\_data1.json",  {  "ADDRESS": "123 YOUR STREET YOUR CITY, ST 12345",  "NAME": "Sarah Smith (123) 456-7890 NO\_REPLY@EXAMPLE.COM",  "GROSS\_AMOUNT": 200,  "NET\_AMOUNT": 200,  "EMPLOYER": "GE HR SERVICES ",  "DATE": None,  },  ), |

## 4.2 Schedule D Form 1040

| **Test Cases** | **Sample Input** |
| --- | --- |
| Test\_get\_name\_ssn | [  ("d\_1.json", {"your\_social\_security\_number" : ["529-02-2934",1,0,[]], "name(s)\_shown\_on\_return" : ["Joseph A & Kristi B Ballstaedt",1,0,[]]}, ["529-02-2934","Joseph A & Kristi B Ballstaedt"]),  ("d\_2.json", {"your\_social\_security\_number" : ["230-91-3306",1,0,[]], "name(s)\_shown\_on\_return" : ["RAED Y ABUALZALAF",1,0,[]]}, ["230-91-3306","RAED Y ABUALZALAF"]),  ("d\_1.json", {}, ["",""],),  ], |
| Test\_get\_name | [  ("d\_1.json", ["Joseph A & Kristi B Ballstaedt"]),  ("d\_2.json", ["RAED Y ABUALZALAF"]),  ], |
| Test\_get\_tabular\_values | [  ("d\_1.json","7","carryover",0 , ["-86"],"NET\_SHORT\_TERM\_CAPITAL\_GAIN\_OR\_LOSS",),  ("d\_1.json","15","carryover" ,0 , ["724,885"],"NET\_LONG\_TERM\_CAPITAL\_GAIN\_OR\_LOSS",),  ("d\_1.json","21" ,"unrecaptured",1 , [""],"CAPITAL\_LOSS\_DEDUCTION",),  ("d\_2.json","7","carryover",0 , ["-27,175"],"NET\_SHORT\_TERM\_CAPITAL\_GAIN\_OR\_LOSS",),  ("d\_2.json","15","carryover" ,0 , ["0"],"NET\_LONG\_TERM\_CAPITAL\_GAIN\_OR\_LOSS",),  ("d\_2.json","21" ,"unrecaptured",1 , ["( 3,000 )"],"CAPITAL\_LOSS\_DEDUCTION",),    ], |
| Test\_D\_main\_1040 | (  "d\_1.json",  {  "NAME": "Joseph A & Kristi B Ballstaedt",  "SOCIAL\_SECURITY\_NUMBER": "529-02-2934",  "NET\_SHORT\_TERM\_CAPITAL\_GAIN\_OR\_LOSS": "-86",  "NET\_LONG\_TERM\_CAPITAL\_GAIN\_OR\_LOSS": "724,885",  "CAPITAL\_LOSS\_DEDUCTION": "",  },  ), |
| Test\_start\_processing | (  "d\_1.json",  {  "NAME": "Joseph A & Kristi B Ballstaedt",  "SOCIAL\_SECURITY\_NUMBER": "529022934",  "NET\_SHORT\_TERM\_CAPITAL\_GAIN\_OR\_LOSS": -86,  "NET\_LONG\_TERM\_CAPITAL\_GAIN\_OR\_LOSS": 724885,  "CAPITAL\_LOSS\_DEDUCTION": "",  },  ), |

# 5. Bitbucket Repository

## 5.1 Entity Extraction Pipeline

| 5.1.1 service-py-docuai-ee-req-handler Request handler does the preprocessing (split if pages > 5) of each request(pdf), makes asynchronous calls to fp-handler microservice, collects the response from ee-extract microservice and gives the response back to the user. 5.1.2 service-py-docuai-ee-fp-handler Fp-handler makes asynchronous calls to Form Parser (compress file if size > 20MB) and returns the Form Parser json back to req-handler. 5.1.3 service-py-docuai-ee-extract-kv Extract-kv receives Form Parser results (multiple jsons if number of pages > 5 in original pdf) from req-handler. If there are multiple jsons, then it combines the results of all splits to form a final document object (containing Form Parser result of all the pages of the original pdf). This final document object is passed to the respective python file (depending on the document type) and the formatted json with KV pair info is returned to req-handler. | |
| --- | --- |

# 

# 6. Evaluation Scripts

The performance of the solution was measured by calculating the accuracies of the entities extracted from the extraction pipeline & comparing those with the ground truth(Tagged Dataset ). This evaluation was carried out on a sample set of 50 documents per document type. It is the arithmetic mean of accuracies over 50 sample documents.

Accuracy of a document is the arithmetic mean of accuracies of each key which is calculated by comparing the key wise Model's output with ground truth. Herein, if the values are string for keys like address, name etc, **Levenshtein Similarity** is used to calculate the accuracy, and if values are numeric for keys like policy number, amount, date, etc, accuracy is calculated using **Business accuracy** which is equal to 100 if it has exact match otherwise 0.

**Levenshtein Similarity:** The Levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other.

**Business accuracy:** Returns 100 if string has exact match otherwise returns 0.

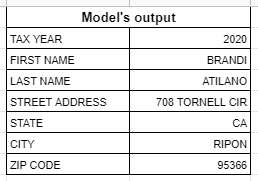
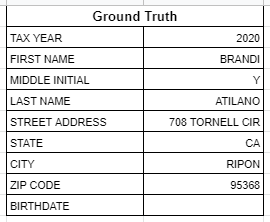
**Evaluation Process:**.

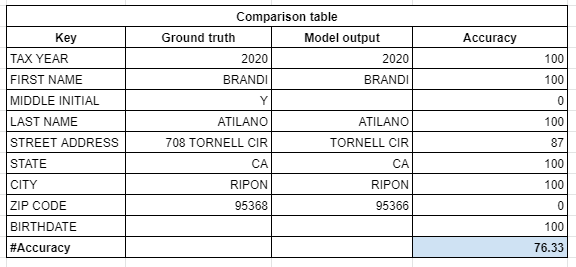
* Compare key-values of Model output with ground truth key-values and calculate

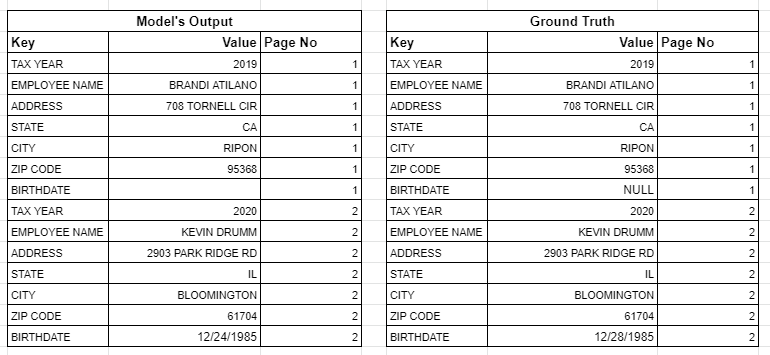
accuracy using levenshtein score or business accuracy.

* **SCENARIO 1:** If a document contains unique keys-values(no keys repeating across a document), accuracy of a document is the average of key accuracies.

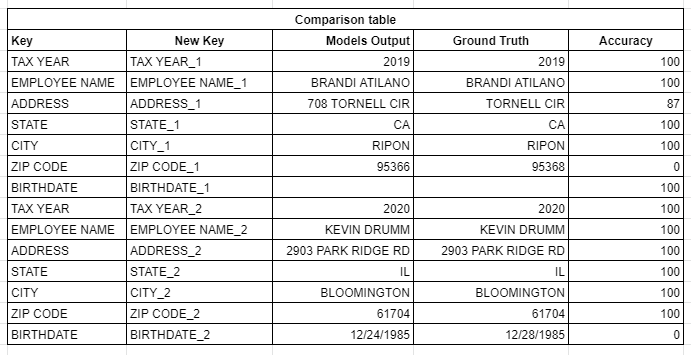
Below example demonstrates the comparison of Model’s output and Tagged data output.

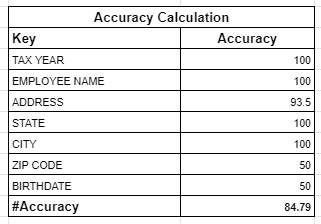




* **SCENARIO 2:** If a document contains repeating keys, all key-values are considered during accuracy calculations.

For repeating keys-values, key name and page no. is used as a key during comparison and then calculated key wise average of accuracy.





Assumptions:

* NULL/BLANK values are considered here as Blank as string where distinguishing between the NULL/BLANKS is not possible.
* If a key is missing, it’s value is Blank calculated using Business accuracy rules of 100 & 0.
* General preprocessing steps are applied on values before going into comparison.

Example: Remove spaces, converting into lower cases, remove punctuations(/,-$), strip blank space or new lines, strip decimal if available at the end of string/numbers

## 6.2 Overall Accuracy

The final accuracy was reported in two- folds as mentioned below.

* **Accuracy per entity**
* **Accuracy per document type**

Detail accuracies per document type & per entity are mentioned in the MODEL KPIs Sheet Final - APR 9  
(link- https://docs.google.com/spreadsheets/d/1PXXNa6oMMoSDGsPXWh7SZn2CnY2uFZX7vYKoZavwgz4/edit#gid=0)

## 6.3 Accuracy per entity

The reported accuracy per entity is the arithmetic mean of each entity across a batch of n documents( n = 50)

## 6.4 Accuracy per document

The reported accuracy per document is the arithmetic mean of the entity overall accuracy across each document.

# 7. Latency Estimations

**End-to-End Pipeline Services**

The Overall End-to-End pipeline includes the calls to these services per document:

* 6 queries are written in cloud SQL.
* 1 message is written in each of the 6 Pub/Sub topics(total 5).
* 1 Document AI API call is executed.
* 3 write in Firestore.
* 6 Cloud function gets triggered.
* 2 writes to GCS

The below table summarizes the results of latencies calculated by taking the average response time of 5 tests using 5 samples of similar page count and size.(For details please refer -

link (<https://docs.google.com/spreadsheets/d/1JbKxDTtheBP1pC64OU-Z0rzZawU7QIGkVcLWc0tlziM/edit#gid=1033163186>)

|  | File Size | | Latency (In secs) | | |
| --- | --- | --- | --- | --- | --- |
| Number of pages | Min | Max | Average | Min | Max |
| 1-2 | 923 KB | 2.2 MB | 6.534 | 4.65 | 9.1 |
| 3-6 | 422.8 KB | 1.7 MB | 7.71 | 6.52 | 8.6 |
| 7-19 | 265.5 KB | 2.5 MB | 12.67 | 5.52 | 26.09 |
| 20-94 | 1.8 MB | 10.7 MB | 36.11 | 19.3 | 70.09 |
| 95-300 | 2.8 MB | 41.3 MB | 54.546 | 43.53 | 67.98 |

# 

# 8. Limitations

**Context:**

Analysis has been carried out on the document types which have comparatively lower extraction accuracy. Here in this section the analysis results for those document types/entities are documented.

**Numeric Data type: Preceding & Trailing zeroes (Identified in Mortgage Statements)**

The trailing zeroes case is handled upto 2 decimals places the values will be present in the modified key output but it is not possible in the case of preceding ones as python on its own removes the preceding zeros.

**Bounding Boxes for rotated documents (Identified in Form 1120s)**

**https://docs.google.com/document/d/1BqTtyKCmM71cPKLSqHjj443d1i8fyg4uxj5TGTua3kM/edit**

**Signature Key - Boolean values extraction (Identified in Purchase Agreements)**

For single signature key present/not present extraction, it was defined to check all the pages including both Buyers & Sellers signature on all pages & then mark them as present/not present which is a larger problem to solve. That is the reason for a low extraction accuracy

**Multiple DL on a page**

# 9. Bug Fixes/Issues Handled

**Context:**

The reported bugs & fixes were handled & documented on the JIRA board(UWM3295), respective deployments were tracked on the deployment tracker sheet

https://docs.google.com/spreadsheets/d/116yWkDmJZNCQmiyhfORTdiiaJlryKLTo2K3JeNzk-1w/edit?usp=drive\_web&ouid=111251272381743907824

9.

Appendix

## 10.1 Reference documents

| **Document** | **Description** | **Link** |
| --- | --- | --- |
| EE Dictionary Sheet | This Sheet contains the information regarding the confirmed data points as of April 1. | <https://docs.google.com/spreadsheets/d/1tXUIS2elSAYkPScDKWRiOd1AGEQSmVnC/edit?ts=60521da2#gid=694198749> |
| Test case document | This sheet contains information regarding error handling with error codes for all possible errors. | <https://docs.google.com/spreadsheets/d/1ApWMy1pbENV7BejBm2yw4BHs3RZhm9u5k-AKVj_Fa_s/edit?ts=602cbc8d#gid=1906858942> |
| Evaluation sheet comparison | This Sheet contains information regarding the method of evaluation used and has data of run-wise accuracy for all doc types. | <https://docs.google.com/spreadsheets/d/1n-vYHgSu71BiXLarZnN_RQz2y6E0kb-DfDvyhuUMzTw/edit?ts=6034f23b#gid=0> |
| Model KPI | These sheets were the ones that were shared regularly with UWM and contain accuracy figures for all the doc types. | [Model KPIs Mar 5](https://docs.google.com/spreadsheets/d/1nsx61d_gL1NupY4bRiGeZA8y4EGEZ-d_hMhnjuxZH3U/edit#gid=0)  [Model KPIs Mar 12](https://docs.google.com/spreadsheets/d/1IZfyrq1cBQH54cMlaPwGMIFwY8RyoyLS6zC03xgmq78/edit#gid=0)  [UWM: EE Comparison Table for String Matching Techniques](https://docs.google.com/spreadsheets/u/0/d/1n-vYHgSu71BiXLarZnN_RQz2y6E0kb-DfDvyhuUMzTw/edit)  [Model KPIs Final - Mar 31](https://docs.google.com/spreadsheets/u/0/d/1PXXNa6oMMoSDGsPXWh7SZn2CnY2uFZX7vYKoZavwgz4/edit)  [Model KPIs Final - APR 9](https://docs.google.com/spreadsheets/u/0/d/18VQ4fyt2Ul3EnXFHehGkeWyI_LGV34H2v_NBQ3qELng/edit)  [Book1(Latest Number|Q updates| APR 21 & MAY 4)](https://docs.google.com/spreadsheets/u/0/d/1s-1cBTShakAQwq26bVrRD-S9akzDz4Uc7-QJh8ZuIgo/edit)  [Book1(May5)](https://docs.google.com/spreadsheets/d/14ziEMXUtOMlk0_8JEP7giLrTrOVJUkBxy0XJwez17uA/edit?pli=1#gid=475230064) |
| API Naming Conventions | This Sheet has information on Naming Conventions and Rules for some keys that are common across all doc types. | <https://docs.google.com/spreadsheets/d/1RaBUoS1VePd0Df6HwCF99LXFWQsecZcEGKtNgbzR3Pw/edit#gid=1831690820> |
| Tagging Rule Book | This Sheet contains Rules of the UWM Tagging Tool. | <https://docs.google.com/spreadsheets/d/1_XcHvJ4GXRBoCo2EKbxfYV8blWgVsWM9oBnLerpvDvs/edit#gid=0> |
| Tagging Tasks Details | This sheet has information on taggers and SPOCs for the concerned doc type. | <https://docs.google.com/spreadsheets/d/1KwMfF97PAsYpEvgkh3N1DCOlV-sCT23tH2l4TBDyhS8/edit?ts=60054f6f#gid=598111631> |
| Tagging User Manual | This doc gives a brief about how to use the tagging tool. | <https://docs.google.com/document/d/1jDwlcp0a640X8bJIoGOhrPp3zvLtVxaakMuCFkB-gXo/edit#heading=h.js1wpa6855s2> |
| Tagging onboarding Doc | This doc has information regarding taggers onboarding and training. | <https://docs.google.com/document/d/1WnorxFbGcCmhxvYvZbdjlD-B2oD7MDnUD6SedG80xZo/edit> |
| Confirmed Data Types and Key Names | This Sheet contains the most recent information regarding the confirmed data points and key names. | [UWM\_Confirmed\_Shared EE Data Points \_ Q Clarifications.xlsx](https://docs.google.com/spreadsheets/u/0/d/1duMJA5lRBl_sEHGtFPLfLd6yUp3HPhMe/edit) |
| Peer Review Document |  |  |
| Unit Testing Document |  |  |

**END OF DOCUMENT**