

# PRE - POST HITL PARSER AND OCR ISSUE IDENTIFIER (External)

## Table of Contents

<b>Disclaimer</b>	<b>1</b>
<b>Objective</b>	<b>2</b>
<b>Prerequisites</b>	<b>2</b>
<b>Step by Step procedure</b>	<b>2</b>
1. Config file Creation	2
a. Config file Creation	2
2. Input Details	3
3. Run the Code	3
4. Output	3
Notebook Script	5

## Disclaimer

This tool is not supported by Google Engineering or Product. It is provided and supported on a best-effort basis by the DocAI Incubator Team. No guarantees of performance are implied.

# Objective

Pre and POST HITL comparison tool which will detect two issues - Parser issue and OCR issue. And the result output contains a summary json file which shows basic stats, count of the OCR and Parser issues for entities present in each document and corresponding analysis csv files.

- **Parser issue** : This issue is identified with the parser when the bounding box is not covering the text region completely and hence the required text was not captured completely. The user will access HITL worker UI and adjust the bounding box to include the text region and save. The script will highlight such cases
- **OCR issue** : This issue is identified with the parser when the bounding box covers the whole text region and as result the expected text was not captured completely. The script will highlight such cases.

## Prerequisites

- Vertex AI Notebook
- Google Cloud Storage bucket
- Pre HITL and Post HITL Json files (filename should be same) in GCS Folders
- DocumentAI and HITL

## Step by Step procedure

### 1. Config file Creation

#### a. Config file Creation

Run the below code and create a config.ini file for providing input.

```
Python
import configparser
config = configparser.ConfigParser()
# Add the structure to the file we will create
config.add_section('Parameters')
config.set('Parameters', 'project_id', 'xxx-xxxxxx-xxxxxx')
config.set('Parameters', 'Pre_HITL_Output_URI', 'gs://')
config.set('Parameters', 'Post_HITL_Output_URI', 'gs://')
# Write the new structure to the new file
with open(r"configfile.ini", 'w') as configfile:
    config.write(configfile)
```

## 2. Input Details

- a. Once **config.ini** file is created with the above step , enter the input in the config file with necessary details as below
  - i. **project\_id**: provide the project id
  - ii. **Pre\_HITL\_Output\_URI**: provide the gcs path of pre HITL jsons (processed jsons)
  - iii. **Post\_HITL\_Output\_URI**: provide the gcs path of post HITL jsons (Jsons processed through HITL)

```
1 [Parameters]
2
3 project_id = xxxx-xxxxxx-xxxxxxx
4 pre_hitl_output_uri = gs://xxxxx_bucket/xxxxxx/xxxxxx/xxxxxxxxxx/
5 post_hitl_output_uri = gs://xxxxx_bucket/xxxxxxx/
6
```

**NOTE: The Name of Post-HITL Json will not be the same as the original file name by default. This has to be updated manually before using this tool.**

## 3. Run the Code

- a. Copy the code provided in this document, Enter the path of the Config file and Run without any edits. The complete notebook script is found in the last section of this document. The output is the summary of entities updated through HITL which has the comparison of pre and post HITL jsons and count of Parser or OCR issue per document.

Python

```
import configparser
#input
Path= "configfile.ini" #Enter the path of config file
config = configparser.ConfigParser()
config.read(Path)

project_id=config.get('Parameters','project_id')
pre_HITL_output_URI = config.get('Parameters','pre_hitl_output_uri')
post_HITL_output_URI = config.get('Parameters','post_hitl_output_uri')
```

## 4. Output

- a. Result summary table is obtained which will highlight the count of parser and ocr issues for each file. The result table will contain details related to pre and post HITL entity changes, whether there were bounding box coordinates mismatched

upon post HITL processing. The below screenshots showcases the parser or ocr issue.

[22]:

	File Name	Entity Type	Pre_HITL_Output	Pre_HITL_bbox	Post_HITL_Output	hitl_update	Post_HITL_bbox	Match	Fuzzy Ratio	bbox_mismatch	OCR Issue	Parser Issue
0	sample-1.json	currency	\$	[0.85089236, 0.43763581, 0.85549796, 0.44936985]	\$	NO	[0.85089236, 0.43763581, 0.85549796, 0.44936985]	TP	1.0	False	No	No
1	sample-1.json	payment_terms	Due upon receipt	[0.7489925, 0.20817035, 0.86931491, 0.22207735]	Due upon receipt	NO	[0.7489925, 0.20817035, 0.86931491, 0.22207735]	TP	1.0	False	No	No
2	sample-1.json	receiver_name	Roger Johnson	[0.74784112, 0.14558887, 0.83304548, 0.15688831]	Roger	YES	[0.74784112, 0.14558887, 0.78776598, 0.15609087]	FP	0.555556	True	No	Yes
3	sample-1.json	supplier_address	1300 Post Road, Suite #2\nFairfield, CT 06824	[0.15947035, 0.098218165, 0.33333334, 0.12038244]	1300 Post Road, Suite #2\nFairfield, CT 06824	NO	[0.15947035, 0.098218165, 0.33333334, 0.12038244]	TP	1.0	False	No	No
4	sample-1.json	total_amount	632.66	[0.8583765, 0.43720123, 0.91134137, 0.44980443]	632.66	NO	[0.8583765, 0.43720123, 0.91134137, 0.44980443]	TP	1.0	False	No	No
8	sample-1.json	due_date	08/04/20	[0.64363843, 0.15688831, 0.70293611, 0.16688396]	08/04/20	NO	[0.64363843, 0.15688831, 0.70293611, 0.16688396]	TP	1.0	False	No	No
9	sample-1.json	ship_to_name	Miller and Sons	[0.15947035, 0.8548457, 0.2642487, 0.86527598]	Miller & Sons	YES	[0.15947035, 0.8548457, 0.2642487, 0.86527598]	FP	0.857143	False	Yes	No
10	sample-1.json	invoice_date	08/18/2020	[0.74841678, 0.18730986, 0.82556129, 0.1990439]	08/18/2020	NO	[0.74841678, 0.18730986, 0.82556129, 0.1990439]	TP	1.0	False	No	No
11	sample-1.json	ship_from_name	Roger Johnson	[0.15947035, 0.6936115, 0.24525043, 0.70578009]	Roger Johnson	NO	[0.15947035, 0.6936115, 0.24525043, 0.70578009]	TP	1.0	False	No	No

Summary json file is generated which will highlight count of bounding box mismatches, OCR and Parser errors and analysis path to result table for each of the processed files.

malj\_ecolab\_hitl\_error /

Name	Last Modified
analysis_07_06_23-092533	an hour ago
configfile.ini	3 hours ago
pre_post_HITL_detect.Parser_OCR_i...	an hour ago
Process_Document_Batch_Single.ipynb	a day ago
summary_07_06_23-092533.json	an hour ago

```

root:
  sample-1.json:
    bbox_mismatch: 1
    OCR_issue: 1
    Parser_issue: 1
    output_file: "analysis_07_06_23-092533/sample-1.json.csv"
  sample-2.json:
    bbox_mismatch: 2
    OCR_issue: 2
    Parser_issue: 2
    output_file: "analysis_07_06_23-092533/sample-2.json.csv"

```

processed filenames

summary file

Entity wise analysis for each file can be observed in the following csv files under analysis/ folder.

sample-1.json.csv

sample-2.json.csv

analysis files

### Table columns:

The result output table has following columns and its details are as follows:

- File Name : name of the file
- Entity Type : type of the entity
- Pre\_HITL\_Output : entity text before HITL
- Pre\_HITL\_bbox : entity bounding box coordinates before HITL
- Post\_HITL\_Output : entity text after HITL
- Hitl\_update : if there was HITL update for that particular entity
- Post\_HITL\_bbox : entity bounding box coordinates after HITL
- Fuzzy Ratio : text match %
- Bbox\_mismatch : if the bounding box coordinates are mismatched
- OCR issue : represents if its classified as OCR Issue
- Parser issue : represents if its classified as Parser Issue

## Notebook Script

Python

```
# installing libraries
import pandas as pd
import operator
import difflib
import json
import os
import pandas as pd
import time
import gcsfs
import numpy as np
from google.cloud import storage
from google.cloud import documentai_v1beta3
from PIL import Image
from typing import Container, Iterable, Iterator, List, Mapping, Optional,
Sequence, Tuple, Union
from PyPDF2 import PdfFileReader
import configparser
import ast
import io
import re
pd.options.mode.chained_assignment = None # default='warn'
```

```

from datetime import datetime
import json,os

#input
Path= "configfile.ini" #Enter the path of config file
config = configparser.ConfigParser()
config.read(Path)

project_id=config.get('Parameters','project_id')
pre_HITL_output_URI = config.get('Parameters','pre_hitl_output_uri')
post_HITL_output_URI = config.get('Parameters','post_hitl_output_uri')

#checking whether bucket exists else create temporary bucket
def check_create_bucket(bucket_name):

    """This Function is to create a temporary bucket
    for storing the processed files
    args: name of bucket"""

    storage_client = storage.Client()
    try:
        bucket = storage_client.get_bucket(bucket_name)
        print(f"Bucket {bucket_name} already exists.")
    except:
        bucket = storage_client.create_bucket(bucket_name)
        print(f"Bucket {bucket_name} created.")
    return bucket

def bucket_delete(bucket_name):
    print("Deleting bucket : ", bucket_name)
    """This function deletes the bucket and used for deleting the temporary
    bucket
    args: bucket name"""
    storage_client = storage.Client()
    try:
        bucket = storage_client.get_bucket(bucket_name)
        bucket.delete(force=True)
    except:
        pass

def file_names(file_path):
    """This Function will load the bucket and get the list of files
    in the gs path given
    args: gs path

```

```

    output: file names as list and dictionary with file names as keys and file
    path as values"""
    bucket=file_path.split("/")[2]
    file_names_list=[]
    file_dict={}
    storage_client = storage.Client()
    source_bucket = storage_client.get_bucket(bucket)
    filenames = [filename.name for filename in
list(source_bucket.list_blobs(prefix=((('/')).join(file_path.split('/'))[3:])))]
    for i in range(len(filenames)):
        x=filenames[i].split('/')[1]
        if x is not "":
            file_names_list.append(x)
            file_dict[x]=filenames[i]
    return file_names_list,file_dict
#list
def list_blobs(bucket_name):
    """This function will give the list of files in a bucket
    args: gcs bucket name
    output: list of files"""
    blob_list = []
    storage_client = storage.Client()
    blobs = storage_client.list_blobs(bucket_name)
    for blob in blobs:
        blob_list.append(blob.name)
    return blob_list

#Bucket operations
def relation_dict_generator(pre_hitl_output_bucket, post_hitl_output_bucket):
    """This Function will check the files from pre_hitl_output_bucket and
    post_hitl_output_bucket
    and finds the json with same names(relation)"""
    pre_hitl_bucket_blobs = list_blobs(pre_hitl_output_bucket)
    post_hitl_bucket_blobs = list_blobs(post_hitl_output_bucket)

    relation_dict = {}
    non_relation_dict={}
    for i in pre_hitl_bucket_blobs:
        for j in post_hitl_bucket_blobs:
            matched_score = difflib.SequenceMatcher(None, i, j).ratio()
            print('matched_score : ', matched_score)
            if matched_score == 1: #0.9 This is for file name. pre and post
hitl json files are to be same
                relation_dict[i] = j

```

```

        else:
            non_relation_dict[i] = "NO POST HITL OUTPUT AVAILABLE"
            #print(i)
    for i in relation_dict:
        if i in non_relation_dict.keys():
            del non_relation_dict[i]
    print('relation_dict = ', relation_dict)
    print('non_relation_dict = ', non_relation_dict)
    return relation_dict, non_relation_dict
def blob_downloader(bucket_name, blob_name):
    """This Function is used to download the files from gcs bucket"""
    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(blob_name)
    contents = blob.download_as_string()
    return json.loads(contents.decode())

def copy_blob(
    bucket_name, blob_name, destination_bucket_name, destination_blob_name):
    """This Method will copy files from one bucket(or folder) to another"""
    storage_client = storage.Client()
    source_bucket = storage_client.bucket(bucket_name)
    source_blob = source_bucket.blob(blob_name)
    destination_bucket = storage_client.bucket(destination_bucket_name)
    blob_copy = source_bucket.copy_blob(
        source_blob, destination_bucket, destination_blob_name
    )
    # print(
    #     "Blob {} in bucket {} copied to blob {} in bucket {}".format(
    #         source_blob.name,
    #         source_bucket.name,
    #         blob_copy.name,
    #         destination_bucket.name,
    #     )
    # )

def bbox_maker(boundingPoly):
    x_list = []
    y_list = []
    for i in boundingPoly:
        x_list.append(i['x'])
        y_list.append(i['y'])
    bbox = [min(x_list), min(y_list), max(x_list), max(y_list)]
    return bbox

```



```

def JsonToDataframe(data):
    ''' Returns entities in dataframe format '''
    df = pd.DataFrame(columns=['type', 'mentionText', 'bbox'])

    if 'entities' not in data.keys():
        return df

    for entity in data['entities']:
        if 'properties' in entity and len(entity['properties'])>0:
            for sub_entity in entity['properties']:
                if 'type' in sub_entity:
                    try:
                        boundingPoly =
sub_entity['pageAnchor']['pageRefs'][0]['boundingPoly']['normalizedVertices']
                        bbox = bbox_maker(boundingPoly)
                        # bbox = [boundingPoly[0]['x'], boundingPoly[0]['y'],
boundingPoly[2]['x'], boundingPoly[2]['y']]
                        df.loc[len(df.index)] = [sub_entity['type'],
sub_entity['mentionText'], bbox]
                    except KeyError:
                        if 'mentionText' in sub_entity:
                            df.loc[len(df.index)] = [sub_entity['type'],
sub_entity['mentionText'], []]
                        else:
                            df.loc[len(df.index)] = [sub_entity['type'],
'Entity not found.', []]
                    elif 'type' in entity:
                        try:
                            boundingPoly =
entity['pageAnchor']['pageRefs'][0]['boundingPoly']['normalizedVertices']
                            bbox = bbox_maker(boundingPoly)
                            # bbox = [boundingPoly[0]['x'], boundingPoly[0]['y'],
boundingPoly[2]['x'], boundingPoly[2]['y']]
                            df.loc[len(df.index)] = [entity['type'], entity['mentionText'],
bbox]
                        except KeyError:
                            if 'mentionText' in entity:
                                df.loc[len(df.index)] = [entity['type'],
entity['mentionText'], []]
                            else:
                                df.loc[len(df.index)] = [entity['type'], 'Entity not
found.', []]
    return df

```

```

def RemoveRow(df, entity):
    ''' Drops the entity passed from the dataframe'''
    return df[df['type'] != entity]

def FindMatch(entity_file1, df_file2):
    ''' Finds the matching entity from the dataframe using
    the area of IOU between bboxes reference
    '''
    bbox_file1 = entity_file1[2]
    # Entity not present in json file
    if not bbox_file1:
        return None

    # filtering entities with the same name
    df_file2 = df_file2[df_file2['type'] == entity_file1[0]]

    # calculating IOU values for the entities
    index_iou_pairs = []
    for index, entity_file2 in enumerate(df_file2.values):
        if entity_file2[2]:
            iou = BBIntersectionOverUnion(bbox_file1, entity_file2[2])
            index_iou_pairs.append((index, iou))

    # choose entity with highest IOU, IOU should be atleast > 0.5
    matched_index = None
    for index_iou in sorted(index_iou_pairs, key=operator.itemgetter(1),
reverse=True):
        if index_iou[1] > 0.2: #0.5
            matched_index = df_file2.index[index_iou[0]]
            break
    return matched_index

def BBIntersectionOverUnion(box1, box2):
    ''' Calculates the area of IOU between two bounding boxes '''
    print("++ BBIntersectionOverUnion ++")
    x1 = max(box1[0], box2[0])
    y1 = max(box1[1], box2[1])
    x2 = min(box1[2], box2[2])
    y2 = min(box1[3], box2[3])

    inter_area = abs(max((x2 - x1, 0)) * max((y2 - y1), 0))

```

```

    if inter_area == 0:
        return 0
    box1_area = abs((box1[2] - box1[0]) * (box1[3] - box1[1]))
    box2_area = abs((box2[2] - box2[0]) * (box2[3] - box2[1]))
    iou = inter_area / float(box1_area + box2_area - inter_area)

    return iou

def GetMatchRatio(values):
    file1_value = values[1]
    file2_value = values[3]
    if file1_value == 'Entity not found.' or file2_value == 'Entity not
found.':
        return 0
    else:
        return difflib.SequenceMatcher(a=file1_value, b=file2_value).ratio()

def compare_pre_hitl_and_post_hitl_output(file1, file2):
    ''' Compares the entities between two files and returns
    the results in a dataframe
    '''
    print("== compare_pre_hitl_and_post_hitl_output ==")
    df_file1 = JsonToDataframe(file1)
    df_file2 = JsonToDataframe(file2)
    file1_entities = [entity[0] for entity in df_file1.values]
    print(file1_entities, '\n')
    file2_entities = [entity[0] for entity in df_file2.values]
    print(file2_entities)

    # find entities which are present only once in both files
    # these entities will be matched directly
    common_entities = set(file1_entities).intersection(set(file2_entities))
    exclude_entities = []
    for entity in common_entities:
        print('entity -- : ', entity)
        if file1_entities.count(entity) > 1 or file2_entities.count(entity) >
1:
            exclude_entities.append(entity)

    print('exclude_entities : ', exclude_entities)
    for entity in exclude_entities:
        common_entities.remove(entity)
    df_compare = pd.DataFrame(columns=['Entity Type',
'Pre_HITL_Output', 'Pre_HITL_bbox' , 'Post_HITL_Output', 'Post_HITL_bbox'])

```

```

print('df_compare:--- \n', df_compare)
for entity in common_entities:
    value1 = df_file1[df_file1['type'] == entity].iloc[0]['mentionText']
    value2 = df_file2[df_file2['type'] == entity].iloc[0]['mentionText']
    bbox1= df_file1[df_file1['type'] == entity].iloc[0]['bbox']
    bbox2= df_file2[df_file2['type'] == entity].iloc[0]['bbox']
    df_compare.loc[len(df_compare.index)] = [entity, value1,
bbox1, value2, bbox2]

    # common entities are removed from df_file1 and df_file2
    df_file1 = RemoveRow(df_file1, entity)
    df_file2 = RemoveRow(df_file2, entity)

# remaining entities are matched comparing the area of IOU across them
mentionText2 = pd.Series(dtype=str)
bbox2 = pd.Series(dtype=str)
for index, row in enumerate(df_file1.values):
    matched_index = FindMatch(row, df_file2)
    if matched_index != None:
        mentionText2.loc[index] = df_file2.loc[matched_index][1]
        bbox2.loc[index] = df_file2.loc[matched_index][2]
        df_file2 = df_file2.drop(matched_index)
    else:
        mentionText2.loc[index] = 'Entity not found.'
        bbox2.loc[index] = 'bbox not found'

df_file1['mentionText2'] = mentionText2.values
df_file1['bbox2'] = bbox2.values
#df_file1 = df_file1.drop(['bbox'], axis=1)
df_file1.rename(columns={'type':'Entity Type',
'mentionText':'Pre_HITL_Output', 'bbox':'Pre_HITL_bbox'
,'mentionText2':'Post_HITL_Output', 'bbox2':'Post_HITL_bbox'}, inplace=True)
df_compare = df_compare.append(df_file1, ignore_index=True)

# adding entities which are present in file2 but not in file1
for row in df_file2.values:
    df_compare.loc[len(df_compare.index)] = [row[0], 'Entity not
found.', 'bbox not present', row[1], row[2]]

# df_compare['Match'] = df_compare['Ground Truth Text'] ==
df_compare['Output Text']
match_array = []
for i in range(0, len(df_compare)):

```

```

        match_string = ''
        if df_compare.iloc[i]['Pre_HITL_Output'] == 'Entity not found.' and
df_compare.iloc[i]['Post_HITL_Output'] == 'Entity not found.':
            match_string = 'TN'
        elif df_compare.iloc[i]['Pre_HITL_Output'] != 'Entity not found.' and
df_compare.iloc[i]['Post_HITL_Output'] == 'Entity not found.':
            match_string = 'FN'
        elif df_compare.iloc[i]['Pre_HITL_Output'] == 'Entity not found.' and
df_compare.iloc[i]['Post_HITL_Output'] != 'Entity not found.':
            match_string = 'FP'
        elif df_compare.iloc[i]['Pre_HITL_Output'] != 'Entity not found.' and
df_compare.iloc[i]['Post_HITL_Output'] != 'Entity not found.':
            if df_compare.iloc[i]['Pre_HITL_Output'] ==
df_compare.iloc[i]['Post_HITL_Output']:
                match_string = 'TP'
            else:
                match_string = 'FP'
        else:
            match_string = 'Something went Wrong.'

    match_array.append(match_string)

df_compare['Match'] = match_array

df_compare['Fuzzy Ratio'] = df_compare.apply(GetMatchRatio, axis=1)
if list(df_compare.index):
    score = df_compare['Fuzzy Ratio'].sum()/len(df_compare.index)
else:
    score = 0

print('match_array')
print(match_array)
return df_compare, score

```

#Execute the below code

```

pre_HITL_output_URI = config.get('Parameters','pre_hitl_output_uri')
post_HITL_output_URI = config.get('Parameters','post_hitl_output_uri')
#print(pre_HITL_output_URI)
#print(post_HITL_output_URI)

#creating temporary buckets
import datetime
now = str(datetime.datetime.now())

```

```

now = re.sub('\W+', '', now)

print("Creating temporary buckets")
pre_HITL_bucket_name_temp = 'pre_hitl_output'+"_"+now
post_HITL_bucket_name_temp = 'post_hitl_output_temp'+"_"+now
#bucket name and prefix
pre_HITL_bucket=pre_HITL_output_URI.split("/")[2]
post_HITL_bucket=post_HITL_output_URI.split("/")[2]
#getting all files and copying to temporary folder

try:
    check_create_bucket(pre_HITL_bucket_name_temp)
    check_create_bucket(post_HITL_bucket_name_temp)
except Exception as e:
    print("unable to create bucket because of exception : ",e)

try:
    pre_HITL_output_files,pre_HITL_output_dict=file_names(pre_HITL_output_URI)
    #print(pre_HITL_output_files,pre_HITL_output_dict)

post_HITL_output_files,post_HITL_output_dict=file_names(post_HITL_output_URI)
    #print(post_HITL_output_files,post_HITL_output_dict)
    print("copying files to temporary bucket")
    for i in pre_HITL_output_files:

copy_blob(pre_HITL_bucket,pre_HITL_output_dict[i],pre_HITL_bucket_name_temp,i)
        for i in post_HITL_output_files:

copy_blob(post_HITL_bucket,post_HITL_output_dict[i],post_HITL_bucket_name_temp,
i)
        pre_HITL_files_list=list_blobs(pre_HITL_bucket_name_temp)
        post_HITL_files_list=list_blobs(post_HITL_bucket_name_temp)
except Exception as e:
    print("unable to get list of files in buckets because : ",e)
#processing the files and saving the files in temporary GCP bucket
fs=gcsfs.GCSFileSystem(project_id)
relation_dict ,non_relation_dict=
relation_dict_generator(pre_HITL_bucket_name_temp, post_HITL_bucket_name_temp)

time_stamp = datetime.now().strftime('%d_%m_%y-%H%M%S')
filename_error_count_dict = {}

```

```

compare_merged = pd.DataFrame()
accuracy_docs=[]
print("comparing the PRE-HITL Jsons and POST-HITL jsns ....Wait for Summary ")
for i in relation_dict:
    #print("***** i : ", i)
    pre_HITL_json = blob_downloader(pre_HITL_bucket_name_temp, i)
    post_HITL_json = blob_downloader(post_HITL_bucket_name_temp,
relation_dict[i])
    # print('pre_HITL_json : ', pre_HITL_json)
    # print('post_HITL_json : ', post_HITL_json)
    compare_output = compare_pre_hitl_and_post_hitl_output(pre_HITL_json,
post_HITL_json)[0]
    # print('compare_output :',compare_output)
    column = [relation_dict[i]] * compare_output.shape[0]
    #print("++++column++++")
    #print(column)
    compare_output.insert(loc = 0,
        column = 'File Name',
        value = column)

    compare_output.insert(loc=5,column = 'hitl_update',value = " ")
    for j in range(len(compare_output)):
        if compare_output['Fuzzy Ratio'][j]!=1.0: #strict
            if compare_output['Pre_HITL_Output'][j]=='Entity not found.' and
compare_output['Post_HITL_Output'][j]=='Entity not found.':
                compare_output['hitl_update'][j]='NO'
            else:
                compare_output['hitl_update'][j]='YES'
        else:
            compare_output['hitl_update'][j]='NO'
    for k in range(len(compare_output)):
        if compare_output['Fuzzy Ratio'][k]!=1.0: #strict
            hitl_update="HITL UPDATED"
            break
        else:
            compare_output['hitl_update'][k]='NO'

    ##
    compare_output['bbox_mismatch'] = compare_output['Pre_HITL_bbox'] !=
compare_output['Post_HITL_bbox']

```

```

#OCR Issue
compare_output['OCR Issue'] = 'No'
#compare_output.loc[(compare_output['Pre_HITL_Output'] !=
compare_output['Post_HITL_Output']), 'OCR Issue'] = 'Yes' # & cordinates are
same
compare_output.loc[
    ((compare_output['Pre_HITL_Output'] !=
compare_output['Post_HITL_Output']))
    &
    (compare_output['Pre_HITL_bbox'] ==
compare_output['Post_HITL_bbox']), 'OCR Issue'
] = 'Yes'

#Parser Issue
compare_output['Parser Issue'] = 'No'
compare_output.loc[ (compare_output['hitl_update'] == 'YES') &
(compare_output['bbox_mismatch'] == True), 'Parser Issue' ] = 'Yes' # &
cordinates are different
#compare_output.loc[
#    ((compare_output['hitl_update'] == 'YES') &
(compare_output['bbox_mismatch'] == True))
#    &
#    (compare_output['Pre_HITL_bbox'] !=
compare_output['Post_HITL_bbox']), 'Parser Issue'
#    ] = 'Yes'

#Parser Issue - entity not found cases | skip if both are 'Entity not
found'
try:
    compare_merged.loc[(compare_merged['Post_HITL_Output'] == 'Entity not
found.') | (compare_merged['Pre_HITL_Output'] == 'Entity not found.'), 'Parser
Issue'] = 'Yes'
except:pass

## global dict : no of parser error / file
temp = {}
temp['bbox_mismatch'] =
len(compare_output[compare_output['bbox_mismatch']==True])

temp['OCR_issue'] = len(compare_output.loc[
    ((compare_output['Pre_HITL_Output'] !=
compare_output['Post_HITL_Output']))

```



```

        &
        (compare_output['Pre_HITL_bbox'] ==
compare_output['Post_HITL_bbox']]))
    temp['Parser_issue'] = len(compare_output.loc[
(compare_output['hitl_update'] == 'YES') & (compare_output['bbox_mismatch'] ==
True)])
    temp['output_file'] = 'analysis_'+time_stamp+'/'+i.replace('json','csv')

    filename_error_count_dict[i] = temp

    new_row=pd.Series([i,"Entities","are updated","by
HITL",":",np.nan,hitl_update,',','',',',',','], index=compare_output.columns)
    compare_output=compare_output.append(new_row,ignore_index= True)
    frames = [compare_merged, compare_output]
    compare_merged = pd.concat(frames)

with open('summary_'+time_stamp+'.json', 'w') as ofile:
    ofile.write(json.dumps(filename_error_count_dict))

for x in relation_dict:
    #print(x)
    file_out = compare_merged[compare_merged['File Name'] == x]
    try:
        os.mkdir('analysis_'+time_stamp)
    except:pass
    file_out.to_csv('analysis_'+time_stamp+'/'+x.replace('json','csv'))

bucket_delete(pre_HITL_bucket_name_temp)
bucket_delete(post_HITL_bucket_name_temp)

```