**UFE 302 Portfolio**

Sam Robson

*“The University Fellows Experience” 1*

*“Fuck Your Club” 2*

*“Marianne Williamson” 5*

*“Fuck CJ Sneckenberger” 8*

*“To Ethan” 10*

*“Ethan Sneckenberger” 12*

*“Untitled (short poem)” 16*

*alac.py 17*

*\_\_main\_\_.py 101*

*\_\_init\_\_.py 113*

*jalapeno.py 113*

*pyproject.toml 123*

*README.md 124*

*LICENSE 132*

(See completed project on [*GitHub*](https://github.com/sbrobson959/alacorder) or [*PyPI*](https://pypi.org/project/alacorder/77.6.8/))

# “The University Fellows Experience”

If you’re a member of the University Fellows Experience and your name isn’t Ethan Sneckenberger (who is himself wrapped up in this mess), I want you to know that I legitimately think you are a bad person. And I want you to know that if the childish, humiliating treatment with which you have greeted me every day for the past two years comes to be worth it when I get the love of my life back, I hope you will know that it wasn’t because of you. Y’all got off watching me fall off the edge. You thought I was being dramatic but you lived for the drama. You thought it was okay because you placed some silly bet a year ago and I stand to gain financially. You’re a bad person. You’re telling yourself excuses to feel good while watching another’s pain. You’re relishing the notion that shit doesn’t matter, or doesn’t have to matter. You’re fucking wrong. Even if you think you love me, I want you to know that you’ve never met me. I bet The Truman Show is a really good watch. It’s a shame y’all thought it was the real me the whole time. How many months after I quit my addiction did it take for y’all to realize you actually put me in danger? And how many minutes after that did it take for y’all to reframe the narrative so that again you are the Messiah, handing rotten bread (no snapper) to the one appointed Not Good Enough. You’re not good enough for Heaven. In Heaven you have to share. Did you know that it’s a privilege to know whether your best friends like you? Did you know that it’s a privilege to wake up in the morning and know you’ll have a conversation with another human being that day? I bet you think you know. I bet you’re coming up with all kinds of fantasies right now of ways y’all could “finish the deal.” Make the lesson sink in. The two year lesson being that I fucking suck. Guess what? You’re right. But not as much as y’all. I used to think I wanted to kill myself. I thought the greatest sights to see would always be out of reach. Fellows taught me that they fucking suck to look at anyway. Maybe you’re telling yourselves excuses now. Oh, Cuba will make it all better. He just doesn’t know what’s going on. Well, we’re just trying to see this thing through. Y’all don’t have a fucking clue how real the harm you have caused in my life is. Y’all don’t have a fucking clue how I’ll respond. You think you know the big secret that’ll save your asses from the guilt trip you’re about to endure. You have no fucking clue. Y’all were lucky I didn’t kill myself last year. This year, you’ll be lucky if you don’t end up dead. I hope I’ve been a good case study for you to make yourself feel good about the world. If it makes you feel any better, I sure don’t. I don’t feel good about the world. I don’t feel optimistic. And it isn’t my fault. I’ve been in the mud for too long. I’m dragging y’all in on my way out. No fucking jokes. I love you Ethan. Don’t let these fugly self-absorbed mansplaining ass hoes convince you that shit has to be like this. Shit can be easy and good. Some of your tbh I don’t actually believe you honestly think these bitches understand might just have to be brutally murdered first. Luckily, they’re doing everything in their power to expedite that fate. Are y’all thinking about how ungrateful I am to push against your shitty efforts at recompense? Good. I know y’all bitches are mad you couldn’t break me. You know y’all are adults right? Like, responsible for your own decisions? This is your decision? Fuck you. How much apple cider vinegar does it take to send a Twink over the edge? How much fucking ass kissing will it take you to feel like you didn’t ruin my life?

# “Fuck Your Club”

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB

FUCK YOUR CLUB !! GIVE ME ETHAN !! GIVE ETHAN ME !! STOP BEING CUNTS !!

FUCK YOUR CLUB

FUCK YOUR SHITTY FUCKING CLUB

FUCK YOUR SHITTY FUCKING CLUB

FUCK YOUR SHITTY FUCKING CLUB

FUCK YOUR SHITTY FUCKING CLUB

FUCK YOUR SHITTY FUCKING CLUB

FUCK YOUR SHITTY FUCKING CLUB

FUCK YOUR SHITTY FUCKING CLUB

GIVE ME WHO I FUCKING LOVE

GIVE ME WHO I FUCKING LOVE

GIVE ME WHO I FUCKING LOVE

GIVE ME WHO I FUCKING LOVE

GIVE ME WHO I FUCKING LOVE

GIVE ME WHO I FUCKING LOVE

GIVE ME WHO I FUCKING LOVE

GIVE ME WHO I FUCKING LOVE

GIVE ME WHO I FUCKING LOVE

GIVE ME WHO I FUCKING LOVE

FUCK WEST CUBA

FUCK FELLOWS

FUCK A.S.S.

FUCK THE COCKBLOCK

FUCK THE COCKBLOCK SO MUCH

FUCK THE COCKBLOCK MORE THAN ANYTHING ELSE ON THIS EARTH

I HOPE YOU ALL DROWN IN THE GULF OF MEXICO

I DIDNT KNOW I HAD SUCH A VIVID CAPACITY TO FANTASIZE VIOLENCE UNTIL I DISCOVERED THIS "JOB INTERVIEW"

FUCKING TORTUROUS SCUM. I THOUGHT MS AYCOCK AND MS SHINN WERE AS LOW AS IT GOT BUT HERE I AM DROWNING IN A PIT OF DESPAIR CONCOCTED BY MY OWN PROFESSORS. FUCK YOU DAVID BOLUS. FUCK YOU DANA PATTON. FUCK YOU JACQUELINE MORGAN. FUCK YOU EVERY FELLOW WHO HAS HELPED KEEP ME AWAY FROM THE ONE I LOVE AND LIED IN MY FUCKING FACE FOR YEARS I HOPE YOU ALL GO TO HELL AND CAN'T WAIT TO SEND YOU THERE. IF I HAD INTERNET I WOULD BE INVESTIGATING CUBAN MURDER LAWS TO SEE IF I COULD GET AWAY WITH IT. I LOOK FORWARD TO LEARNING MORE ABOUT ALABAMA'S LAWS UPON MY RETURN. IF YOU WANT TO COME THEN STOP FUCKING GASLIGHTING ME AND GIVE ME HIM. FUCK YOU. FUCK YOUR MONEY. FUCK YOUR ARROGANT MIDSOMMAR ''SPIRITUAL'' BULLSHIT. YOU RUINED MY FUCKING LIFE AND LIED TO MY FACE SAYING YOU SAVED IT DAY AFTER FUCKING DAY. AND I KNOW Y'ALL MADE THE BUS ANNOUNCER DUDE TALK ALL QUIET AND SHIT SO I WOULDN'T MAKE IT ON TIME.

# “Marianne Williamson”

to anyone who doesn't understand what the punishment is, i'll spell it out for you.

it has nothing to do with python.

the punishment is to subject myself at all times to a brain that works against me not for me. to live my life like im bill murray in groundhog day, but with a key added element: im not allowed to think. if i think i might do something different. i might wait a second too long before the 500th git commit. i might leave a comment i took out the draft before. or convert to spaces instead of tabs. im not allowed to grow. i already know pandas but i have to make a for loop from the start anyway. i have to remanufacture being stuck on a problem, so intently that i myself cannot tell the difference. at least in white bear the girl didn't have to take away her own memories. she didn't have to bear the shame or confusion of having done it to herself. she didn't have to sit with the terror of knowing you'll again be made to forget and rediscover. have you ever taken a deep breath and told yourself that by the end of your exhale you won't remember the reason why you get out of bed every morning? that if you don't, he'll actually go away? eurydice forgot the warmth of summer until orpheus reminds her with his song. orpheus still turned around to look. who knows if he will have the chance to walk forward again? if he can afford to give all of himself to something powerful yet indifferent and pray she will return until he can pray no longer. i havent attended a class besides fellows in a month and im stuck in a loop for a reason ive never been stuck before. theres been opportunities for growth. moments where my mind started to come to, to notice that theres something other than the computer screen in front of me as i sink further into credit card debt and academic disrepair, let my friends and family gaslight me, wonder if ive had a conversation two or three or four times before, wonder if i'll ever be able to trust the one i love, wonder if i'll ever be able to trust myself again. Every morning I decide again that this is worth the rest of my life's effort. I'll keep deciding that. i really will. but look: i have nothing. because of you. the one who could give me everything. at least give me everything he took away. the only friends ive seen in two years have had to have scripted conversations with me and lie to me. I know they thought I'd remember its "just a Fellows thing" or whatever. if y'all knew the truth of what its like to have everyone you've ever loved turned away from you overnight, to convince you that it's your fault, or leave you to convince yourself. and here i am living out my bad place truman show of horror wondering whether he'll actually come back this time or whether the months of submission and penitence are yet again an elaborate scheme to break your heart. A scheme which your parents endorsed. And your best friends. And your professors. And your therapist. I don't even know what it would mean to sign yourself up for the torture I have just gone through. Even though I did. The only reason I didn't kill myself in high school is that I still had enough to get me out of bed, even if I was miserable both in bed and out. i had a taste of what it felt like to be loved. i had a sense of what it felt like to love another. but all i could feel was the lack. how i wasn't good enough. how no matter how funny i was at school or hardworking i was in my studies i would never be treated like someone capable of giving a fuck. when i came to ua it was supposed to be a second chance at defining life on my terms. i knew how to check boxes but i had no clue how to make the boxes worth checking. i really hope none of you ever have to discover what its like to run out of energy for the ones you love. you might walk 500 miles and 500 more for him or her, but would you face the heartbreak of taking that last step when you can't take another? the heartbreak of having to find that out like john mulaneys dog petunia had mad beef with your sorry ass? i know a lot of you have felt a lot of shame for making a half-nighter of what was supposed to be an all-nighter. of getting a B on a test you really could've gotten an A on if you just studied a little more. what if that B meant you would never cry the same way you used to cry? that life itself was just a big scam? that we're playing ourselves by continuing to try? and how many hours spent studying would it take for you to feel like you gave it your best shot as you fail? they really do take everything away from you. everyone. even yourself. and they'll laugh at you about it. they'll take the thing that bugs you each morning and make a theme month out of it. you won't be invited to the party. you're the show. even now i fear sometimes that im just an emotional prostitute. here to spread my marianne williamson bullshit to help you hide from the fact that you never really know when your next moment of true safety is. To know that I'm ok. That my loved ones are ok. That the good stuff will come back in time. that the good ones haven't abandoned you. sometimes the good ones really do abandon you folks. there will probably be a moment in your life where you feel so unloved you forget what love feels like. where you don't know that at least at thanksgiving you'll get a good hug or a real conversation. it's all just a matter of how much effort we put in for each other. the difference between the wheat and the chaff is this: the chaff gives up when life isn't worth it. And sometimes, folks, life really isn't worth it. A lot of people out there will live a life of suffering before an early and untimely death. The sad shit really is sad. The sad shit really does happen. (The good stuff does too.) The hardest thing I've ever had to admit to myself is that sometimes life isn't worth it. Have you ever spent months in isolation writing a Python script to nanosecond specifications, only to finish moments before the one you most love in the entire world laughs in your face and deletes it. What would you do if that happened to you, if he were gone as quickly as he came, and you're left to sit with the humiliation of knowing you were the dumb fucking fool who poured his mind heart and soul into what was supposed to be a love letter to him. to yourself. to your family. to your community. to life giving you second chances. to giving life a second chance. and a third. and a fourth. to be told that it was all a delusion. that you're fucked up for what you said. and what would you do if no one called you afterwards to check in? or answered your texts? guess ill just go back to making these fruit fly traps and drowning in misery in peace. Too weak to remember. Too tired to cry. Too broken to recover. only mere feet from the greatest thing that ever happened to anyone or anything. from the one you were promised. from the one you promised yourself to. from what gave life promise. to know that before it wasn't your fault it was your fault. to know you've caused the pain you've felt. not just in you but in others. to know you're part of The Problem. i dont have a photographic memory. im just a photograph. stuck in time. but like, in a bad way. forced to represent someone i no longer stand for. i did it for him. i did it for me. i did it for both of us. i didnt do it for cj fuckin sneckenberger radiologist ass bitch who thinks he runs tuscaloosa ass bitch. i did it because i had nothing else to do that was worth it. i had no clue whether it would be. i knew i was in love but i thought i knew i was too much of a bitch to ever be loved by him. i saved my life when i decided id rather act on the former than the latter. would you rather hide in your room and feel nothing for another year, or let the world be cruel to take a chance on yourself? think if i were ethan: he put his faith in me by giving me a second chance (and a third blah blah you've read the poem). He put his faith in me long before I had the guts to have faith in myself. He put his faith in me knowing that. Maybe because of that. So I have decided to do the same for him. CJ doesn't stand a chance. Even as I sit here, day after day, facing the doubt and shame, I will keep walking forward for this man. It might be to Cuba or it might be to CJ's house with a rusty shovel and a debt to collect. I pray that everyone else out there does the same. Don't be like CJ. Don't be a fucking dick to prove you're strong to people. Don't teach the world's ills by inflicting them on your loved ones. They say you should stay with the devil you know. They're fucking wrong bro. Run away from that creature of satan. I don't have any money and my freezer holds but one amys frozen margherita to share. but ive got good friends, some shit to talk, two truly terribly designed couches, a shit ton of wax, and a passion for you. It aches to know that you had to grow up with that. To know that you had to wrestle with whether that's ok. To think that loving people means hurting people or being hurt by people. I guess they never managed to convince you of that. thank. god. I pray for the day I convince you the opposite. I pray to repeat that day everyday. If you'll let me. <3 Sam

# “Fuck CJ Sneckenberger”

fuck jupyter i hate it and i hate everything right now except for ethan sneckenberger

i knew how to make this in 2015

...

like...

what?

are?

yall?

on?

i know you want to torture me for what i did to your son freshman year. but hell, yall just wanna torture me in general. and y'all would rather him and i burn together while you sit back smugly sipping martinis priding yourselves on the important lessons you spend your time teaching the youth. i will smile the day cj sneckenberger dies. and even then i know it wouldn't feel but a fraction of the hate with which his family has conducted themselves toward me in his stead. ive been in a fucking black mirror episode for two years now. i dont know which way is up. i feel like sandra fucking bullock in gravity. y'all took my loved ones. my parents. my friends. my old friends. my old enemies. my fucking doctors. my therapist. my professors. my privacy. my free will. my sanity. my most important memories. you didn't just throw em away. you, as frank reynolds would say, smeared feces all over the walls. have you ever been forced to forget you fell in love? to have the love of your life say to your face that they hope you do more drugs so your life will be worse without them? that they hope you get worse? that they hope you stay sick? that they hope you get sicker? i was struggling with what is essentially a benzo addiction during the most difficult part of my life, not two months after my cousin died of a heroin overdose and you spent the time you could've used helping me get support cutting me off from my friends and family. Telling the people who I can ask for help that I'm just being dramatic. Just being a little baby. A little bitch who couldn't do what it takes. Come face me fuckign CJ and i'll show you that i have what it takes to put this to an end. if it means putting you to an end the biggest loss for me will be the blood on my fucking shirt.

you're right - i dont get to know what its like to know right from wrong or left from right. i don't know which version of my life story is true. i didn't ask for a fucking life of pi torture chamber and i won't stay in it any longer. So I have decided i'm the fucking tiger and you're going to fucking die. On Thursday we leave for Cuba. If Ethan and I aren't on the plane together, we'll be getting ready for your funeral. I hope you're fine with him bringing a plus one. things'll even out in the end.

you thought you pissed off someone powerless and weak. you were wrong. now you're hiding in a spy van or whatever trying to convince yourself, your son, and the folks around you that you're doing the right thing. the best thing for a just world.

you sound like joaquin phoenix in the fucking joker.

suck my dick.

choke on it.

relive your shitty life.

lie and tell yourself you hated out of love.

lie and tell yourself it was worth it.

or don't. i don't care.

i'm not here for you. i'm here for ethan. it's time you decided whether you value owning your son over being your son.

# “To Ethan”

I hear I’ve worked up quite an audience with this Python script. How many boogaloos does it take to fix a relationship? How many boogaloos does it take to fix a Selenium scraper, fighting against the slowest website of all time? I’m frankly disappointed I previously failed to answer all these sorts of questions. I’m more than disappointed. I’m heartbroken. Honestly I couldn’t give a fuck about my career when the love of my life is right in front of me. I can’t describe what it’s like to know I broke a perfect heart. I can’t describe what it’s like to have a second chance and to blow it because of fucking drugs and shitty nights at Red Shed. But I’m doing it differently this time. When I said in the poem that I would never again forget the one I most love, I meant it. Like, really meant it. I thought I was done for when I came into college. I thought the world was done for, that we were left to strive for faraway goals until our eyes fall closed. That I was happy but happiness was a lie, an image, a facade, a flex. That the only salve for the isolation of adolescence is giving up your will to pop culture and becoming a piece of plastic because you can’t bear to feel human. That all changed overnight when I fell in love with you. And I took it away when I forgot. And I will never forgive myself. But know this, Ethan: I will never forget our love. I will never forget how you told my story. I’ll never forget how you leaped at the chance to help write it. I’ll never forget how I hurt you. I’ll never forget how you loved me anyway. I feel like I know who I am when I’m with you. It’s a fucking shame I lied to myself and everyone I knew to convince them the high life was ours. Mine. What a dumb fucking lie. I’ve sat here for months writing line by line by line: only rarely do I see a face. But still I wake up everyday grateful that I can just be with myself. And if others do see me, it will be me they see. I hope they see you by my side. I hope I have the chance to make things better. To watch through a window and see my life playing out as I had hoped while I pull up the pandas cheat sheet for the 4th time on GitHub. Not knowing which of my friends are my friends. Not knowing which of my decisions will lead me to you and which will lead me away. I’ve been through some shit and I know you have too, I’m sorry to put you through more. I’m sorry for blaming you for the consequences of my actions. I’m sorry for blaming you for pain you had to feel because of things I said to you. Lies. And for what? A day on Ashlyn’s balcony? A half decent smoothie from heritage house? Thank you for calling me on my bullshit. No one else had the guts. Thank you for believing I can be better. Thank you for loving me flawed as I am. I love you more than life, and it is my greatest hope that I can spend my life together with you.

With love,

Sam

P.S. I saw you today in class. I was trying to fix the party search fields for the scraper and caught you in the corner of my eye. It broke me. Please come back. I will set whatever boundaries or whatever your parents want from me. I will write this script 1,000,000,000,000,000,000,000+ times. But it’s time I took my life back. You took incredible care of it, and I really hope you’ll share it with me. The one who wouldn’t give up on me. Please never give up. I know I have before, but let me tell you right now: I will not give up. And I will do what it takes to make you feel like the happiest man in the world. You deserve to be. I really fucking miss you, dude. I know your parents are in this as well and you don’t have control. But I pray that you use the control that you do have to take an nth chance on me. I love you, Ethan Sneckenberger, and I know I don’t deserve you, but goddamnit, I just can’t live without you. Every thought in every minute. Begging for you to come back. Even though I know I’m the one who left. I’m the one who iced things over. I’m the one who broke your trust. Who ruined what should have been great. Well, I thought that was true, but that was before I knew that you work miracles. You work fucking miracles, Ethan Sneckenberger. And even if it doesn’t work out, I hope you know that I will love you until the day I die. You are loved Ethan <3 I know you know that. I’m gonna keep telling you anyway. I wish I could do it in person. I wish I could hear your response. All day every day I play back our memories and think of where I’d be without you. You’ve put up with hate notes and gaslighting, forced acting to please your parents, lies and some cruel tricks of memory. I’m taking care of myself now so I can be there for you, and I understand it will take time to prove it. I hope you keep watching over me, even if things don’t work out. If it helps, I will work every day of my life to protect the one who protects me. To protect our love and the vulnerability you gave me. A gift I didn’t deserve. When you have put up with so much shit you didn’t deserve. But still you went forward. Still you didn’t give up on me when I had given up on myself. You saved my life Ethan. I really mean that. You saved my life.

I’m sorry for the shitty hate notes. And forgetting. These past few years have been a lot and it pains me to know that you have faced collateral damage from the worst decisions I have made in my life. Now I am making the best. All I want is to be in your arms. I will stay away if that brings you more pain. I do hope it doesn’t, and I feel confident when I say to you that I have put in the work to become me. In one single second you pierced the shame which hid my heart from my head. A curse I thought was unbreakable, broken by the one who I broke. In one single second I felt like myself, I felt like I know what that means. Please give me the chance to grow into me, to grow into me by your side. You’re the sun, Ethan. You’re the fucking sun. Forever will I revolve. I really hope you’ll share your warmth with me tonight.

Sam (again) (damn long postscript I guess)

# “Ethan Sneckenberger”

Ethan Sneckenberger

oh quash this beef dear family of mine

ive sat here for weeks, in python, i pine

its truly been a shitful year

and even still i'm not in the clear

so free me from my prison of shame

though surely i deserve the blame

ive sat in penitent filth for thee

ive published not one draft but 783

(to prove i will fight for my place in heaven,

i just published 787)

ive pepped and ive pooped and ive smoked so much tree

(like a lot a lot)

ive fixed all the indents and parsed all the fees

tallying charges all night and all day

here on this dumb east edge couch here i'll stay

so plunge into me as i plunge into you

oh alacorder you make me so blue

but one thing i know 'fore my heart can amend

i must tend to you 'fore my dick i will tend

my dick cries its hunger, i weep for its thirst

but do let me take care of tutwiler first

the snake in my pants puts my head in a trance

i give not a look, not a stare, not a glance

but still in my heart i know one thing is true

there's truly no end to how much i'll do

for the one that i love, i'll forego the dove

layer by layer i'll unpeel the onion

i'll fight through the rumors

the gossip the haters

ill fight through my doubt and

ill fight through my shame

ill toil and soil

submit to the coil

i won't lose myself

(but sure would everything else)

heck maybe i already have

but ill come up with more

(for i have more in store)

and i'll do what it takes to pull through

oh 'corder of 'corders i've filled in your borders

your seams and your wide open fields

if its not much trouble (though surely i'd double

my effort and time put in thee)

til i find more grub, i pray you sit stable

but first bring the one i most love

not alac but ethan the one who raised thee

i ate tenders chicken

i failed to build pygion

sublime text i trust has enabled your thrust

but the one true sublime is his faith in mine

i can no longer bear to be blind

i promise you now

that i'll never forget

of you, of your love, of ours.

i hope that you know

that our love is eternal

for cj i'll write every for loop and line (fuck him tho fr)

but for you i would lay down my life.

the one who gave me a second

and third

and fourth

and fifth

and sixth

and seventh

chance

i know there were more, for

brevity i'm sure, you'll

understand my will to abridge

but in case you do

doubt my love for you

i want you to know that i do

you called my worst bluff

brought up my worst stuff

you did what i thought was impossible

you understood my fears my sins and my heart

you charged my world to do the same

not only me but my whole family

will forever be changed by your name

i pray every day

you'll return and I'll stay

more innocent than 'fore we first met

i know i've atoned

e'en though i've been stoned

and trust i've no greater regret:

im sorry i broke your heart and i will not leave this earth

without putting it back together. never again could i look in your eyes and lie knowing you'd cry or you'd worry. i would break knowing i could break you. i will break to keep you unbroken.

and i broke knowing i broke you.

it is my greatest regret.

not only the lie but the shade and the sighs

of indifference i slandered your name with,

that i couldn't face the most beautiful face

is a burden i always must bear.

but i hope to grow from this burden, this shame

and to you share the fruits of my labor

from my greatest regret my mind has been set

on what and who matters to me

your trust is the one thing i'll live for and die for.

ill never forget i hurt the greatest one.

ill never forget i forgot.

the greatest thing that ever happened to me.

my guardian angel.

my rock.

the one.

the actual one.

my love. <3

ethan sneckenberger

# “Untitled (short poem)”

alacorder has been done. for like weeks. y'all are just too dumb to use a command line interface (try python3 -m alacorder --help). would you like the next draft to be dressed in lace? hand etched with the story of my youth? installable via a floppy disk on gold leaf?

the jupyter notebook isn't the alacorder it's the fucking reading glasses i have to make so that your 60 year old arrogant ass can claim i wasted my time.

burn in hell i hope you lose your voting rights.

fuck off everyone except ethan sneckenberger.

fuck right off.

love you ethan.

come back.

<3

sam

# alac.py

"""

alac 77

"""

import glob

import inspect

import math

import os

import re

import sys

import datetime

import time

import warnings

import PyPDF2

import click

import numpy as np

import pandas as pd

import selenium

from tqdm.autonotebook import tqdm

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import Select

from selenium.webdriver.chrome.options import Options

pd.set\_option("mode.chained\_assignment", None)

pd.set\_option("display.notebook\_repr\_html", True)

pd.set\_option('display.expand\_frame\_repr', False)

pd.set\_option('display.max\_rows', 100)

pd.set\_option('compute.use\_bottleneck', True)

pd.set\_option('compute.use\_numexpr', True)

pd.set\_option('display.max\_categories', 16)

pd.set\_option('display.precision',2)

tqdm.pandas()

warnings.filterwarnings('ignore')

## CONFIG

def setinputs(path, debug=False, fetch=False):

"""Verify and configure input path. Must use set() to finish configuration even if NO\_WRITE mode. Call setoutputs() with no arguments.

Args:

path (str): Path to PDF directory, compressed archive, or query template sheet,

debug (bool, optional): Print detailed logs,

fetch (bool, optional): Configure template sheet for web PDF retrieval

Returns:

inp = pd.Series({

INPUT\_PATH: (path-like obj) path to input file or directory,

IS\_FULL\_TEXT: (bool) origin is case text (False = PDF directory or query template),

QUEUE: (list) paths or case texts (dep. on IS\_FULL\_TEXT) for export function,

FOUND: (int) total cases found in input path,

GOOD: (bool) configuration succeeded,

PICKLE: (pd.DataFrame) original archive file (if appl.),

ECHO: (IFrame(str)) log data for console

})

"""

if fetch == True or (os.path.splitext(path)[1] in [".xlsx",".xls",".csv",".json"]):

queue = readPartySearchQuery(path)

out = pd.Series({

'INPUT\_PATH': path,

'IS\_FULL\_TEXT': False,

'QUEUE': queue,

'FOUND': 0,

'GOOD': True,

'PICKLE': '',

'ECHO': ''

})

return out

else:

found = 0

is\_full\_text = False

good = False

pickle = None

if not debug:

warnings.filterwarnings('ignore')

if isinstance(path, pd.core.frame.DataFrame) or isinstance(path, pd.core.series.Series):

if "AllPagesText" in path.columns and path.shape[0] > 0:

queue = path['AllPagesText']

is\_full\_text = True

found = len(queue)

good = True

pickle = path

path = "NONE"

elif isinstance(path, str) and path != "NONE":

queue = pd.Series()

if os.path.isdir(path): # if PDF directory -> good

queue = pd.Series(glob.glob(path + '\*\*/\*.pdf', recursive=True))

if queue.shape[0] > 0:

found = len(queue)

good = True

elif os.path.isfile(path) and os.path.splitext(path)[1] == ".xz":

good = True

pickle = pd.read\_pickle(path, compression="xz")

queue = pickle['AllPagesText']

is\_full\_text = True

found = len(queue)

elif os.path.isfile(path) and (os.path.splitext(path)[1] == ".zip"):

nzpath = path.replace(".zip","")

nozipext = os.path.splitext(nzpath)[1]

if debug:

click.echo(f"NZPATH: {nozipext}, NOZIPEXT: {nozipext}, PATH: {path}")

if nozipext == ".json":

pickle = pd.read\_json(path, orient='table',compression="zip")

queue = pickle['AllPagesText']

is\_full\_text = True

found = len(queue)

good = True

if nozipext == ".csv":

pickle = pd.read\_csv(path, escapechar='\\',compression="zip")

queue = pickle['AllPagesText']

is\_full\_text = True

good = True

found = len(queue)

if nozipext == ".parquet":

pickle = pd.read\_parquet(path,compression="zip")

queue = pickle['AllPagesText']

is\_full\_text = True

found = len(queue)

good = True

if nozipext == ".pkl":

pickle = pd.read\_pickle(path,compression="zip")

queue = pickle['AllPagesText']

is\_full\_text = True

found = len(queue)

good = True

elif os.path.isfile(path) and os.path.splitext(path)[1] == ".json":

try:

pickle = pd.read\_json(path, orient='table')

except:

pickle = pd.read\_json(path, orient='table',compression="zip")

queue = pickle['AllPagesText']

is\_full\_text = True

found = len(queue)

good = True

elif os.path.isfile(path) and os.path.splitext(path)[1] == ".csv":

pickle = pd.read\_csv(path, escapechar='\\')

queue = pickle['AllPagesText']

is\_full\_text = True

found = len(queue)

good = True

elif os.path.isfile(path) and os.path.splitext(path)[1] == ".pkl":

pickle = pd.read\_pickle(path)

queue = pickle['AllPagesText']

is\_full\_text = True

found = len(queue)

good = True

elif os.path.isfile(path) and os.path.splitext(path)[1] == ".parquet":

pickle = pd.read\_parquet(path)

queue = pickle['AllPagesText']

is\_full\_text = True

found = len(queue)

good = True

else:

good = False

else:

good = False

if good:

echo = f"Found {found} cases in input."

else:

echo = "Alacorder failed to configure input! Try again with a valid PDF directory or full text archive path, or run 'python3 -m alacorder --help' in command line for more details."

out = pd.Series({

'INPUT\_PATH': path,

'IS\_FULL\_TEXT': is\_full\_text,

'QUEUE': pd.Series(queue),

'FOUND': found,

'GOOD': good,

'PICKLE': pickle,

'ECHO': echo

})

return out

def setoutputs(path="", debug=False, archive=False, table="", fetch=False):

"""Verify and configure output path. Must use set(inconf, outconf) to finish configuration.

Args:

path (str): Path to PDF directory, compressed archive, or query template sheet,

debug (bool, optional): Print detailed logs,

archive (bool, optional): Create full text archive,

table (str, optional): Create table ('cases', 'fees', 'charges', 'filing', 'disposition')

Returns:

out = pd.Series({

'OUTPUT\_PATH': (path-like obj) path to output file or directory,

'ZIP\_OUTPUT\_PATH': (path-like obj) path to output file or directory with .zip if zip compression enabled,

'OUTPUT\_EXT': (str) output file extension,

'MAKE': (str) table, archive, or directory to be made at init(),

'GOOD': (bool) configuration succeeded,

'EXISTING\_FILE': (bool) existing file at output path,

'ECHO': (IFrame(str)) log data

)}

"""

good = False

make = ""

compress = False

exists = False

ext = ""

if not debug:

warnings.filterwarnings('ignore')

if fetch:

if os.path.isdir(path): # if PDF directory -> good

make = "pdf\_directory"

good = True

else:

good = False

else:

if ".zip" in path or ".xz" in path:

compress=True

nzpath = path.replace(".zip","")

# if no output -> set default

if path == "" and archive == False:

path = "NONE"

ext = "NONE"

make == "multiexport" if table != "cases" and table != "charges" and table != "fees" and table != "disposition" and table != "filing" else "singletable"

good = True

exists = False

if path == "" and archive == True:

path = "NONE"

ext = "NONE"

make == "archive"

good = True

exists = False

# if path

if isinstance(path, str) and path != "NONE" and make != "pdf\_directory":

exists = os.path.isfile(path)

ext = os.path.splitext(path)[1]

if ext == ".zip": # if vague due to compression, assume archive

ext = os.path.splitext(os.path.splitext(path)[0])[1]

compress = True

good = True

if ext == ".xz" or ext == ".parquet" or ext == ".pkl": # if output is existing archive

make = "archive"

compress = True

good = True

elif ext == ".xlsx" or ext == ".xls": # if output is multiexport

make = "multiexport"

good = True

elif archive == False and (ext == ".csv" or ext == ".dta" or ext == ".json" or ext == ".txt"):

make = "singletable"

good = True

elif archive == True and (ext == ".csv" or ext == ".dta" or ext == ".json" or ext == ".txt"):

make = "archive"

good = True

if good and not debug:

echo = "Successfully configured output."

if good and debug:

echo = "Output path is valid. Call set() to finish configuration."

out = pd.Series({

'OUTPUT\_PATH': nzpath,

'ZIP\_OUTPUT\_PATH': path,

'OUTPUT\_EXT': ext,

'MAKE': make,

'GOOD': good,

'EXISTING\_FILE': exists,

'COMPRESS': compress,

'ECHO': echo

})

return out

def set(inputs, outputs=None, count=0, table='', overwrite=False, log=True, dedupe=False, no\_write=False, no\_prompt=False, debug=False, no\_batch=True, compress=False, fetch=False, fetch\_cID="", fetch\_uID="", fetch\_pwd="",fetch\_qmax=0, fetch\_qskip=0, fetch\_speed=1, archive=False):

"""Verify and configure task from setinputs() and setoutputs() configuration objects and \*\*kwargs. Must call init() or export function to begin task.

DO NOT USE TO CALL ALAC.FETCH() OR OTHER BROWSER-DEPENDENT METHODS.

Args:

inputs (obj): configuration object from setinputs(),

outputs (None, optional): configuration object from setoutputs(),

count (int, optional): (int) total cases in queue,

table (str, optional): table export setting,

overwrite (bool, optional): overwrite without prompting,

log (bool, optional): print logs to console,

dedupe (bool, optional): remove duplicates from archive export,

no\_write (bool, optional): don't write to output file,

no\_prompt (bool, optional): don't prompt user for input,

debug (bool, optional): print detailed logs,

no\_batch (bool, optional): don't split task into batches,

compress (bool, optional): compress output file (Excel files not supported)

Returns:

out = pd.Series({

'GOOD': (bool) configuration succeeded,

'ECHO': (IFrame(str)) log data,

'TIME': timestamp at configuration,

'QUEUE': (list) paths or case texts to process,

'COUNT': (int) total in queue,

'IS\_FULL\_TEXT': (bool) origin is case text (False = PDF directory or query template),

'MAKE': (str) table, archive, or directory to be made at init(),

'TABLE': Table export selection if appl. ('cases', 'fees', 'charges', 'filing', 'disposition')

'INPUT\_PATH': (path-like obj) path to input file or directory,

'OUTPUT\_PATH': (path-like obj) path to output file or directory,

'OUTPUT\_EXT': (str) output file extension,

'OVERWRITE': (bool) existing file at output path will be overwritten,

'FOUND': (int) cases found in inputs,

'DEDUPE': (bool) remove duplicate cases from exported archives

'LOG': (bool) print logs to console,

'DEBUG': (bool) print detailed logs,

'NO\_PROMPT': (bool) don't prompt user for input,

'NO\_WRITE': (bool) don't write file to output path,

'NO\_BATCH': (bool) don't split task into batches,

'COMPRESS': (bool) compress output if supported

'FETCH': fetch,

'ALA\_CUSTOMER\_ID': fetch\_cID,

'ALA\_USER\_ID': fetch\_uID,

'ALA\_PASSWORD': fetch\_pwd

})

"""

echo = ""

will\_overwrite = False

good = True

if not debug:

sys.tracebacklimit = 0

warnings.filterwarnings('ignore')

else:

sys.tracebacklimit = 10

# DEDUPE

content\_len = len(inputs.QUEUE)

if dedupe and not fetch:

queue = inputs.QUEUE.drop\_duplicates()

dif = content\_len - queue.shape[0]

if (log or debug) and dif > 0:

click.secho(f"Removed {dif} duplicate cases from queue.", fg='bright\_yellow', bold=True)

else:

queue = inputs.QUEUE

# COUNT

content\_len = inputs['FOUND']

if content\_len > count and count != 0:

ind = count - 1

queue = inputs.QUEUE[0:ind]

elif count > content\_len and content\_len > 0:

count = inputs.QUEUE.shape[0]

elif count < content\_len and count == 0:

count = content\_len

else:

queue = inputs.QUEUE

count += 1

echo = echo\_conf(inputs.INPUT\_PATH, outputs.MAKE, outputs.OUTPUT\_PATH, overwrite, no\_write, dedupe, no\_prompt, compress)

if outputs.COMPRESS == True:

compress = True

cftime = time.time()

if archive:

make = "archive"

else:

make = outputs.MAKE

out = pd.Series({

'GOOD': good,

'ECHO': echo,

'TIME': cftime,

'QUEUE': queue,

'COUNT': count,

'IS\_FULL\_TEXT': bool(inputs.IS\_FULL\_TEXT),

'MAKE': make,

'TABLE': table,

'INPUT\_PATH': inputs.INPUT\_PATH,

'OUTPUT\_PATH': outputs.OUTPUT\_PATH,

'OUTPUT\_EXT': outputs.OUTPUT\_EXT,

'OVERWRITE': will\_overwrite,

'FOUND': inputs.FOUND,

'DEDUPE': dedupe,

'LOG': log,

'DEBUG': debug,

'NO\_PROMPT': no\_prompt,

'NO\_WRITE': no\_write,

'NO\_BATCH': no\_batch,

'COMPRESS': compress,

'FETCH': fetch,

'ALA\_CUSTOMER\_ID': fetch\_cID,

'ALA\_USER\_ID': fetch\_uID,

'ALA\_PASSWORD': fetch\_pwd

})

return out

def setpaths(input\_path, output\_path=None, count=0, table='', overwrite=False, log=True, dedupe=False, no\_write=False, no\_prompt=False, debug=False, no\_batch=True, compress=False, fetch=False, fetch\_cID="", fetch\_uID="", fetch\_pwd="", fetch\_qmax="", fetch\_qskip="", fetch\_speed=1, archive=False): # DOC

"""Substitute paths for setinputs(), setoutputs() configuration objects for most tasks. Must call init() or export function to begin task.

DO NOT USE TO CALL ALAC.FETCH() OR OTHER BROWSER-DEPENDENT METHODS.

Args:

input\_path (str): (path-like obj) path to input file or directory,

output\_path (None, optional): (path-like obj) path to output file or directory,

count (int, optional): (int) total cases in queue,

table (str, optional): table export setting,

overwrite (bool, optional): overwrite without prompting,

log (bool, optional): print logs to console,

dedupe (bool, optional): remove duplicates from archive export,

no\_write (bool, optional): don't write to output file or directory,

no\_prompt (bool, optional): don't prompt user for input,

debug (bool, optional): print detailed logs,

no\_batch (bool, optional): don't split task into batches,

compress (bool, optional): compress output file (Excel files not supported)

Returns:

out = pd.Series({

'GOOD': (bool) configuration succeeded,

'ECHO': (IFrame(str)) log data,

'TIME': timestamp at configuration,

'QUEUE': (list) paths or case texts to process,

'COUNT': (int) total in queue,

'IS\_FULL\_TEXT': (bool) origin is case text (False = PDF directory or query template),

'MAKE': (str) table, archive, or directory to be made at init(),

'TABLE': Table export selection if appl. ('cases', 'fees', 'charges', 'filing', 'disposition')

'INPUT\_PATH': (path-like obj) path to input file or directory,

'OUTPUT\_PATH': (path-like obj) path to output file or directory,

'OUTPUT\_EXT': (str) output file extension,

'OVERWRITE': (bool) existing file at output path will be overwritten,

'FOUND': (int) cases found in inputs,

'DEDUPE': (bool) remove duplicate cases from exported archives

'LOG': (bool) print logs to console,

'DEBUG': (bool) print detailed logs,

'NO\_PROMPT': (bool) don't prompt user for input,

'NO\_WRITE': (bool) don't write file to output path,

'NO\_BATCH': (bool) don't split task into batches,

'COMPRESS': (bool) compress output if supported,

})

"""

a = setinputs(input\_path, fetch=fetch)

if log:

click.secho(a.ECHO)

b = setoutputs(output\_path, fetch=fetch)

if b.MAKE == "archive": #

compress = True

c = set(a, b, count=count, table=table, overwrite=overwrite, log=log, dedupe=dedupe, no\_write=no\_write, no\_prompt=no\_prompt, debug=debug, no\_batch=no\_batch, compress=compress, fetch=fetch, fetch\_cID=fetch\_cID, fetch\_uID=fetch\_uID, fetch\_pwd=fetch\_pwd, fetch\_qmax=fetch\_qmax, fetch\_qskip=fetch\_qskip, fetch\_speed=fetch\_speed, archive=archive)

if log:

click.secho(c.ECHO)

return c

def setinit(input\_path, output\_path=None, archive=False,count=0, table='', overwrite=False, log=True, dedupe=False, no\_write=False, no\_prompt=False, debug=False, no\_batch=True, compress=False, fetch=False, fetch\_cID="",fetch\_uID="", fetch\_pwd="", fetch\_qmax=0, fetch\_qskip=0, fetch\_speed=1): # DOC

"""

Initialize tasks from paths without calling setinputs(), setoutputs(), or set().

Note additional fetch flags for auth info if task involves alac.fetch()

Args:

input\_path (str): (path-like obj) path to input file or directory,

output\_path (None, optional): (path-like obj) path to output file or directory,

archive (bool, optional): make compressed archive,

count (int, optional): (int) total cases in queue,

table (str, optional): table export setting,

overwrite (bool, optional): overwrite without prompting,

log (bool, optional): print logs to console,

dedupe (bool, optional): remove duplicates from archive export,

no\_write (bool, optional): don't write to output file or directory,

no\_prompt (bool, optional): don't prompt user for input,

debug (bool, optional): print detailed logs,

no\_batch (bool, optional): don't split task into batches,

compress (bool, optional): compress output file (Excel files not supported)

fetch\_cID (str): Alacourt.com Customer ID

fetch\_uID (str): Alacourt.com User ID

fetch\_pwd (str): Alacourt.com Password

fetch\_qmax (int, optional): Max queries to pull from inputs

fetch\_qskip (int, optional): Skip top n queries in inputs

fetch\_speed (int, optional): Fetch rate multiplier

Returns: [out, init\_out]

out = pd.Series({

'GOOD': (bool) configuration succeeded,

'ECHO': (IFrame(str)) log data,

'TIME': timestamp at configuration,

'QUEUE': (list) paths or case texts to process,

'COUNT': (int) total in queue,

'IS\_FULL\_TEXT': (bool) origin is case text (False = PDF directory or query template),

'MAKE': (str) table, archive, or directory to be made at init(),

'TABLE': Table export selection if appl. ('cases', 'fees', 'charges', 'filing', 'disposition')

'INPUT\_PATH': (path-like obj) path to input file or directory,

'OUTPUT\_PATH': (path-like obj) path to output file or directory,

'OUTPUT\_EXT': (str) output file extension,

'OVERWRITE': (bool) existing file at output path will be overwritten,

'FOUND': (int) cases found in inputs,

'DEDUPE': (bool) remove duplicate cases from exported archives

'LOG': (bool) print logs to console,

'DEBUG': (bool) print detailed logs,

'NO\_PROMPT': (bool) don't prompt user for input,

'NO\_WRITE': (bool) don't write file to output path,

'NO\_BATCH': (bool) don't split task into batches,

'COMPRESS': (bool) compress output if supported

})

init\_out = pd.DataFrame() # depends on init() configuration

"""

if fetch:

fetch\_no\_log = not log

if not isinstance(input\_path, pd.core.series.Series) and not isinstance(output\_path, pd.core.series.Series):

fetch(input\_path, output\_path, fetch\_cID, fetch\_uID, fetch\_pwd, fetch\_qmax, fetch\_qskip, fetch\_speed, fetch\_no\_log)

else:

input\_path = setinputs(input\_path)

output\_path = setoutputs(output\_path)

fetch(input\_path, output\_path, fetch\_cID, fetch\_uID, fetch\_pwd, fetch\_qmax, fetch\_qskip, fetch\_speed, fetch\_no\_log)

else:

if not isinstance(input\_path, pd.core.series.Series) and input\_path != None:

input\_path = setinputs(input\_path)

if not isinstance(output\_path, pd.core.series.Series) and output\_path != None:

output\_path = setoutputs(output\_path)

a = set(input\_path, output\_path, count=count, table=table, overwrite=overwrite, log=log, dedupe=dedupe, no\_write=no\_write, no\_prompt=no\_prompt,debug=debug, no\_batch=no\_batch, compress=compress, archive=archive)

if archive == True:

a.MAKE = "archive"

b = init(a)

return b

## CORE PARSE FUNCTIONS

def write(conf, outputs):

"""

Writes (outputs) to file at (conf.OUTPUT\_PATH)

Args:

conf (pd.Series): Configuration object with paths and settings

outputs (pd.Series|pd.DataFrame): Description

Returns:

outputs: DataFrame written to file at conf.OUTPUT\_PATH

DataFrame

"""

if conf.OUTPUT\_EXT == ".xls":

try:

with pd.ExcelWriter(conf.OUTPUT\_PATH) as writer:

outputs.to\_excel(writer, sheet\_name="outputs", engine="openpyxl")

except (ImportError, IndexError, ValueError, ModuleNotFoundError, FileNotFoundError):

try:

with pd.ExcelWriter(conf.OUTPUT\_PATH) as writer:

outputs.to\_excel(writer, sheet\_name="outputs")

except (ImportError, IndexError, ValueError, ModuleNotFoundError, FileNotFoundError):

outputs.to\_json(os.path.splitext(conf.OUTPUT\_PATH)[

0] + "-cases.json.zip", orient='table')

click.echo(conf, f"Fallback export to {os.path.splitext(conf.OUTPUT\_PATH)[0]}-cases.json.zip due to Excel engine failure, usually caused by exceeding max row limit for .xls/.xlsx files!")

if conf.OUTPUT\_EXT == ".xlsx":

try:

with pd.ExcelWriter(conf.OUTPUT\_PATH) as writer:

outputs.to\_excel(writer, sheet\_name="outputs", engine="openpyxl")

except (ImportError, IndexError, ValueError, ModuleNotFoundError, FileNotFoundError):

try:

with pd.ExcelWriter(conf.OUTPUT\_PATH[0:-1]) as writer:

outputs.to\_excel(writer, sheet\_name="outputs", engine="xlsxwriter")

except (ImportError, IndexError, ValueError, ModuleNotFoundError, FileNotFoundError):

outputs.to\_json(os.path.splitext(conf.OUTPUT\_PATH)[

0] + ".json.zip", orient='table', compression="zip")

click.echo(conf, f"Fallback export to {os.path.splitext(conf.OUTPUT\_PATH)}.json.zip due to Excel engine failure, usually caused by exceeding max row limit for .xls/.xlsx files!")

elif conf.OUTPUT\_EXT == ".pkl":

if conf.COMPRESS:

outputs.to\_pickle(conf.OUTPUT\_PATH + ".xz", compression="xz")

else:

outputs.to\_pickle(conf.OUTPUT\_PATH)

elif conf.OUTPUT\_EXT == ".xz":

outputs.to\_pickle(conf.OUTPUT\_PATH, compression="xz")

elif conf.OUTPUT\_EXT == ".json":

if conf.COMPRESS:

outputs.to\_json(conf.OUTPUT\_PATH + ".zip",

orient='table', compression="zip")

else:

outputs.to\_json(conf.OUTPUT\_PATH, orient='table')

elif conf.OUTPUT\_EXT == ".csv":

if conf.COMPRESS:

outputs.to\_csv(conf.OUTPUT\_PATH + ".zip",

escapechar='\\', compression="zip")

else:

outputs.to\_csv(conf.OUTPUT\_PATH, escapechar='\\')

elif conf.OUTPUT\_EXT == ".txt":

outputs.to\_string(conf.OUTPUT\_PATH)

elif conf.OUTPUT\_EXT == ".dta":

outputs.to\_stata(conf.OUTPUT\_PATH)

elif conf.OUTPUT\_EXT == ".parquet":

if conf.COMPRESS:

outputs.to\_parquet(conf.OUTPUT\_PATH, compression="brotli")

else:

outputs.to\_parquet(conf.OUTPUT\_PATH)

else:

pass

return outputs

def archive(conf):

"""

Write full text archive to file.pkl.xz

Args:

conf (pd.Series): Configuration object with paths and settings

Returns:

DataFrame (written to file at conf.OUTPUT\_PATH)

"""

start\_time = time.time()

if conf.LOG or conf['DEBUG']:

click.echo("Writing full text archive from cases...")

if not conf.IS\_FULL\_TEXT:

allpagestext = pd.Series(conf.QUEUE).progress\_map(lambda x: getPDFText(x))

else:

allpagestext = pd.Series(conf.QUEUE)

if (conf.LOG or conf['DEBUG']) and conf.IS\_FULL\_TEXT == False:

click.echo("Exporting archive to file at output path...")

outputs = pd.DataFrame({

'Path': conf.QUEUE if not conf.IS\_FULL\_TEXT else np.nan,

'AllPagesText': allpagestext,

'Timestamp': start\_time,

})

outputs.fillna('', inplace=True)

outputs = outputs.convert\_dtypes()

if conf.DEDUPE:

old = conf.QUEUE.shape[0]

outputs = outputs.drop\_duplicates()

dif = outputs.shape[0] - old

if dif > 0 and conf.LOG:

click.echo(f"Removed {dif} duplicate cases from queue.")

if not conf.NO\_WRITE and conf.OUTPUT\_EXT == ".xz":

outputs.to\_pickle(conf.OUTPUT\_PATH, compression="xz")

if not conf.NO\_WRITE and conf.OUTPUT\_EXT == ".pkl":

if conf.COMPRESS:

outputs.to\_pickle(conf.OUTPUT\_PATH + ".xz", compression="xz")

else:

outputs.to\_pickle(conf.OUTPUT\_PATH)

if not conf.NO\_WRITE and conf.OUTPUT\_EXT == ".csv":

if conf.COMPRESS:

outputs.to\_csv(conf.OUTPUT\_PATH + ".zip",

escapechar='\\', compression="zip")

else:

outputs.to\_csv(conf.OUTPUT\_PATH, escapechar='\\')

if not conf.NO\_WRITE and conf.OUTPUT\_EXT == ".parquet":

if conf.COMPRESS:

outputs.to\_parquet(conf.OUTPUT\_PATH + ".parquet", compression="brotli")

else:

outputs.to\_parquet(conf.OUTPUT\_PATH + ".parquet", compression="brotli")

if not conf.NO\_WRITE and conf.OUTPUT\_EXT == ".json":

if conf.COMPRESS:

outputs.to\_json(conf.OUTPUT\_PATH + ".zip",

orient='table', compression="zip")

else:

outputs.to\_json(conf.OUTPUT\_PATH, orient='table')

complete(conf, outputs)

return outputs

def map(conf, \*args, bar=True, names=[]):

"""

Return DataFrame from config object and custom column 'getter' functions like below:

def getter(full\_case\_text: str):

out = re.search(...)

...

return out

Creates DataFrame with cols: CaseNumber, getter\_1(), getter\_2(), ...

Getter functions must take case text as first parameter. Subsequent paramters can be set in map() after the getter parameter. Getter functions must return string, float, or int outputs to map().

Example:

>> a = alac.map(conf,

alac.getAmtDueByCode, 'D999',

alac.getAmtPaidByCode, 'D999',

alac.getName,

alac.getDOB)

>> print(a)

Args:

conf (pd.Series): Configuration object with paths and settings

\*args: def getter(text: str) -> float,

def getter(text: str) -> int,

def getter(text: str) -> str,

def getter(text: str) -> bool, # check / debug

Returns:

out = pd.DataFrame({

'CaseNumber': (str) full case number with county,

'getter\_1': (float) outputs of getter\_1(),

'getter\_2': (int) outputs of getter\_2(),

'getter\_3': (str) outputs of getter\_2()

})

"""

start\_time = time.time()

df\_out = pd.DataFrame()

temp\_no\_write\_tab = False

if conf.DEDUPE: # remove duplicates from queue

old = conf.QUEUE.shape[0]

conf.QUEUE = conf.QUEUE.drop\_duplicates()

dif = conf.QUEUE.shape[0] - old

if dif > 0 and conf.LOG:

click.secho(f"Removed {dif} duplicate cases from queue.", fg='bright\_yellow', bold=True)

if not conf.NO\_BATCH: # split into batches

batches = batcher(conf)

else:

batches = [conf.QUEUE]

# sort args into functions and their parameters

func = pd.Series(args).map(lambda x: 1 if inspect.isfunction(x) else 0)

funcs = func.index.map(lambda x: args[x] if func[x] > 0 else np.nan)

no\_funcs = func.index.map(lambda x: args[x] if func[x] == 0 else np.nan)

countfunc = func.sum()

column\_getters = pd.DataFrame(columns=['Name', 'Method', 'Arguments'], index=(range(0, countfunc)))

# call methods, return outputs with pandas-friendly dtype

def ExceptionWrapper(getter, text, \*args):

if args:

outputs = pd.Series(getter(text, args))

else:

outputs = pd.Series(getter(text))

return outputs.values

# set name of methods to name w/o "get", i.e. getName() -> 'Name' column in df\_out

for i, x in enumerate(funcs):

if inspect.isfunction(x):

try:

if len(names)>=i:

column\_getters.Name[i] = names[i]

else:

column\_getters.Name[i] = str(x.\_\_name\_\_).replace("get","").upper()

except:

column\_getters.Name[i] = str(x).replace("get","").upper()

column\_getters.Method[i] = x

for i, x in enumerate(args):

if not inspect.isfunction(x):

column\_getters.Arguments[i] = x

# run batch

for i, b in enumerate(batches):

if i > 0:

click.echo(f"Finished batch {i}. Now reading batch {i+1} of {len(batches)}")

b = pd.DataFrame()

# stop slow writes on big files between batches

if bool(conf.OUTPUT\_PATH) and i > 0 and not conf.NO\_WRITE:

if os.path.getsize(conf.OUTPUT\_PATH) > 500:

temp\_no\_write\_tab = True

if i == len(conf.QUEUE) - 1:

temp\_no\_write\_tab = False

# get text

if conf.IS\_FULL\_TEXT:

allpagestext = conf.QUEUE

else:

if bar:

tqdm.pandas(desc="PDF => Text")

allpagestext = pd.Series(conf.QUEUE).progress\_map(lambda x: getPDFText(x))

else:

allpagestext = pd.Series(conf.QUEUE).map(lambda x: getPDFText(x))

# retrieve getter

for i in column\_getters.index:

name = column\_getters.Name[i]

arg = column\_getters.Arguments[i]

getter = column\_getters.Method[i]

# map getter

for i, getter in enumerate(column\_getters.Method.tolist()):

arg = column\_getters.Arguments[i]

name = column\_getters.Name[i]

if bar and name != "CaseNumber":

if arg == pd.NaT:

tqdm.pandas(desc=name)

col = allpagestext.progress\_map(lambda x: getter(x, arg))

else:

tqdm.pandas(desc=name)

col = allpagestext.progress\_map(lambda x: getter(x))

else:

if arg == pd.NaT:

col = allpagestext.map(lambda x: getter(x, arg))

else:

col = allpagestext.map(lambda x: getter(x))

new\_df\_to\_concat = pd.DataFrame({name: col})

df\_out = pd.concat([df\_out, new\_df\_to\_concat], axis=1)

df\_out = df\_out.dropna(axis=1)

df\_out = df\_out.dropna(axis=0)

df\_out = df\_out.convert\_dtypes()

# fix empty -> str error

for col in column\_getters.columns:

column\_getters[col] = column\_getters[col].dropna()

column\_getters[col] = column\_getters[col].map(lambda x: "" if x == "Series([], Name: AmtDue, dtype: float64)" or x == "Series([], Name: AmtDue, dtype: object)" else x)

# write

if conf.NO\_WRITE == False and temp\_no\_write\_tab == False and (i % 5 == 0 or i == len(conf.QUEUE) - 1):

write(conf, df\_out)

if not conf.NO\_WRITE:

return df\_out

write(conf, df\_out)

return df\_out

def stack(dflist, \*old\_df):

try:

dflist = dflist.dropna()

except:

pass

try:

dflist = dflist.tolist() # -> [df, df, df]

except:

pass

dfliststack = pd.concat(dflist, axis=0, ignore\_index=True)

if not old\_df:

return dfliststack

else:

out = pd.concat([old\_df, dfliststack], axis=0,ignore\_index=True)

out = out.dropna()

out = out.fillna('', inplace=True)

return out

def table(conf):

"""

Route config to export function corresponding to conf.TABLE

Args:

conf (pd.Series): Configuration object with paths and settings

Returns:

DataFrame written to file at conf.OUTPUT\_PATH

DataFrame

"""

a = []

if conf.MAKE == "multiexport":

a = cases(conf)

elif conf.TABLE == "cases":

a = cases(conf)

elif conf.TABLE == "fees":

a = fees(conf)

elif conf.TABLE == "charges":

a = charges(conf)

elif conf.TABLE == "disposition":

a = charges(conf)

elif conf.TABLE == "filing":

a = charges(conf)

else:

a = None

return a

def init(conf):

"""

Start export function corresponding to conf.MAKE, conf.TABLE

Args:

conf (pd.Series): Configuration object with paths and settings

Returns:

DataFrame written to file at conf.OUTPUT\_PATH

DataFrame

"""

a = []

if conf.FETCH == True:

fetch(conf.INPUT\_PATH, conf.OUTPUT\_PATH, fetch\_cID=conf.ALA\_CUSTOMER\_ID, fetch\_uID=conf.ALA\_USER\_ID, fetch\_pwd=conf.ALA\_PASSWORD, fetch\_qmax=conf.FETCH\_QMAX, fetch\_qskip=conf.FETCH\_QSKIP,fetch\_speed=conf.FETCH\_SPEED)

elif conf.MAKE == "multiexport" and (conf.TABLE == "" or conf.TABLE == "all"):

a = cases(conf)

elif conf.MAKE == "archive":

a = archive(conf)

elif conf.TABLE == "cases":

a = cases(conf)

elif conf.TABLE == "fees":

a = fees(conf)

elif conf.TABLE == "charges":

a = charges(conf)

elif conf.TABLE == "disposition":

a = charges(conf)

elif conf.TABLE == "filing":

a = charges(conf)

else:

a = None

return a

def batcher(conf, queue=pd.Series()):

"""Splits conf.QUEUE objects into batches

Args:

conf (pd.Series): Configuration object with paths and settings

Returns:

batches: (numpy.array) list of pd.Series()

"""

if queue.shape[0] == 0:

q = conf['QUEUE']

else:

q = queue

if not conf.IS\_FULL\_TEXT:

if conf.FOUND < 1000:

batchsize = 250

elif conf.FOUND > 10000:

batchsize = 2500

else:

batchsize = 1000

batches = np.array\_split(q, 3)

else:

batches = np.array\_split(q, 1)

return batches

## TABLE PARSERS

def cases(conf):

"""

Return [cases, fees, charges] tables as List of DataFrames from batch

See API docs for table-specific output tokens

Args:

conf (pd.Series): Configuration object with paths and settings

Returns:

list = [cases, fees, charges]:

out[0] = cases table (see alac.caseinfo().\_\_str\_\_ for outputs)

out[1] = fees table (see alac.fees().\_\_str\_\_ for outputs)

out[2] = charges table (see alac.charges().\_\_str\_\_ for outputs)

"""

arch = pd.DataFrame()

cases = pd.DataFrame()

fees = pd.DataFrame()

allcharges = pd.DataFrame()

start\_time = time.time()

temp\_no\_write\_arc = False

temp\_no\_write\_tab = False

if conf.DEDUPE:

old = conf.QUEUE.shape[0]

conf.QUEUE = conf.QUEUE.drop\_duplicates()

dif = conf.QUEUE.shape[0] - old

if dif > 0 and conf.LOG:

click.secho(f"Removed {dif} duplicate cases from queue.",

fg='bright\_yellow', bold=True)

queue = pd.Series(conf.QUEUE)

if not conf.IS\_FULL\_TEXT:

tqdm.pandas(desc="PDF => Text")

queue = pd.Series(conf.QUEUE).progress\_map(lambda x: getPDFText(x))

conf.QUEUE = queue

conf.IS\_FULL\_TEXT = True

if not conf['NO\_BATCH']:

batches = batcher(conf, queue)

else:

batches = [pd.Series(queue)]

for i, c in enumerate(batches):

b = pd.DataFrame({'AllPagesText':c})

c = pd.Series(c)

if i > 0:

click.echo(f"Finished batch {i}. Now reading batch {i+1} of {len(batches)}")

b = pd.DataFrame()

b['CaseInfoOutputs'] = c.map(lambda x: getCaseInfo(x))

b['CaseNumber'] = b['CaseInfoOutputs'].map(lambda x: x[0]).astype(str)

b['Name'] = b['CaseInfoOutputs'].map(lambda x: x[1]).astype(str)

b['Alias'] = b['CaseInfoOutputs'].map(lambda x: x[2]).astype(str)

b['DOB'] = b['CaseInfoOutputs'].map(lambda x: x[3]).astype(str)

b['Race'] = b['CaseInfoOutputs'].map(lambda x: x[4]).astype(str)

b['Sex'] = b['CaseInfoOutputs'].map(lambda x: x[5]).astype(str)

b['Address'] = b['CaseInfoOutputs'].map(lambda x: x[6]).astype(str)

b['Phone'] = b['CaseInfoOutputs'].map(lambda x: x[7]).astype(str)

tqdm.pandas(desc="Charges")

allcharges = setinit(conf, conf.OUTPUT\_PATH, archive=False, table="charges", no\_write=True, no\_prompt=True, log=False, no\_batch=True)

tqdm.pandas(desc="Fee Sheets")

b['FeeOutputs'] = c.progress\_map(lambda x: getFeeSheet(x))

b['TotalAmtDue'] = b['FeeOutputs'].map(lambda x: x[0])

b['TotalBalance'] = b['FeeOutputs'].map(lambda x: x[1])

b['PaymentToRestore'] = c.map(

lambda x: getPaymentToRestore(x))

b['FeeCodesOwed'] = b['FeeOutputs'].map(lambda x: x[3]).astype(str)

b['FeeCodes'] = b['FeeOutputs'].map(lambda x: x[4]).astype(str)

b['FeeCodesOwed'] = b['FeeCodesOwed'].map(lambda x: '' if x == "nan" else x).astype(str)

b['FeeCodes'] = b['FeeCodes'].map(lambda x: '' if x == "nan" else x).astype(str)

b['FeeSheet'] = b['FeeOutputs'].map(lambda x: x[5])

feesheet = b['FeeOutputs'].map(lambda x: x[6])

feesheet = feesheet.dropna()

feesheet = feesheet.fillna('')

feesheet = feesheet.tolist() # -> [df, df, df]

feesheet = pd.concat(feesheet, axis=0, ignore\_index=True) # -> batch df

fees = pd.concat([fees, feesheet],axis=0, ignore\_index=True)

allcharges = allcharges.dropna()

if bool(conf.OUTPUT\_PATH) and i > 0 and not conf.NO\_WRITE:

if os.path.getsize(conf.OUTPUT\_PATH) > 1000:

temp\_no\_write\_arc = True

if bool(conf.OUTPUT\_PATH) and i > 0 and not conf.NO\_WRITE:

if os.path.getsize(conf.OUTPUT\_PATH) > 1000:

temp\_no\_write\_tab = True

if i >= len(batches) - 1:

temp\_no\_write\_arc = False

temp\_no\_write\_tab = False

if (i % 5 == 0 or i == len(batches) - 1) and not conf.NO\_WRITE and temp\_no\_write\_arc == False:

if bool(conf.OUTPUT\_PATH) and len(conf.OUTPUT\_EXT) > 2:

q = pd.Series(conf.QUEUE) if conf.IS\_FULL\_TEXT == False else pd.NaT

ar = pd.DataFrame({

'Path': q,

'AllPagesText': c,

'Timestamp': start\_time

}, index=range(0, conf.COUNT))

try:

arch = pd.concat([arch, ar], ignore\_index=True, axis=0)

except:

pass

arch.fillna('', inplace=True)

arch.dropna(inplace=True)

arch.to\_pickle(conf.OUTPUT\_PATH, compression="xz")

b.drop(

columns=['CaseInfoOutputs',

'FeeOutputs', 'FeeSheet'],

inplace=True)

if conf.DEDUPE:

old = conf.QUEUE.shape[0]

cases = cases.drop\_duplicates()

dif = cases.shape[0] - old

if dif > 0 and conf.LOG:

click.secho(f"Removed {dif} duplicate cases from queue.",

fg='bright\_yellow', bold=True)

if conf.LOG:

click.secho(f"Cleaning outputs and writing file to export path...", fg='yellow',italic=True)

b.fillna('', inplace=True)

cases = pd.concat([cases, b], axis=0, ignore\_index=True)

cases['TotalAmtDue'] = pd.to\_numeric(cases['TotalAmtDue'], 'ignore')

cases['TotalBalance'] = pd.to\_numeric(cases['TotalBalance'], 'ignore')

cases['PaymentToRestore'] = pd.to\_numeric(cases['PaymentToRestore'], 'ignore')

if conf.MAKE == "cases":

write(conf, cases)

elif not temp\_no\_write\_tab:

if conf.OUTPUT\_EXT == ".xls":

try:

with pd.ExcelWriter(conf.OUTPUT\_PATH) as writer:

cases.to\_excel(writer, sheet\_name="cases", engine="openpyxl")

fees.to\_excel(writer, sheet\_name="fees", engine="openpyxl")

allcharges.to\_excel(writer, sheet\_name="charges", engine="openpyxl")

except (ImportError, IndexError, ValueError, ModuleNotFoundError, FileNotFoundError):

click.echo(f"openpyxl engine failed! Trying xlsxwriter...")

with pd.ExcelWriter(conf.OUTPUT\_PATH) as writer:

cases.to\_excel(writer, sheet\_name="cases")

fees.to\_excel(writer, sheet\_name="fees")

allcharges.to\_excel(writer, sheet\_name="charges")

elif conf.OUTPUT\_EXT == ".xlsx":

try:

with pd.ExcelWriter(conf.OUTPUT\_PATH, engine="openpyxl") as writer:

cases.to\_excel(writer, sheet\_name="cases", engine="openpyxl")

fees.to\_excel(writer, sheet\_name="fees", engine="openpyxl")

allcharges.to\_excel(writer, sheet\_name="charges", engine="openpyxl")

except (ImportError, IndexError, ValueError, ModuleNotFoundError, FileNotFoundError):

try:

if conf.LOG:

click.echo(f"openpyxl engine failed! Trying xlsxwriter...")

with pd.ExcelWriter(conf.OUTPUT\_PATH) as writer:

cases.to\_excel(writer, sheet\_name="cases", engine="xlsxwriter")

fees.to\_excel(writer, sheet\_name="fees", engine="xlsxwriter")

allcharges.to\_excel(writer, sheet\_name="charges", engine="xlsxwriter")

except (ImportError, FileNotFoundError, IndexError, ValueError, ModuleNotFoundError):

try:

cases.to\_json(os.path.splitext(conf.OUTPUT\_PATH)[

0] + "-cases.json.zip", orient='table')

fees.to\_json(os.path.splitext(conf.OUTPUT\_PATH)[

0] + "-fees.json.zip", orient='table')

allcharges.to\_json(os.path.splitext(conf.OUTPUT\_PATH)[

0] + "-charges.json.zip", orient='table')

click.echo(f"""Fallback export to {os.path.splitext(conf.OUTPUT\_PATH)[0]}-cases.json.zip due to Excel engine failure, usually caused by exceeding max row limit for .xls/.xlsx files!""")

except (ImportError, FileNotFoundError, IndexError, ValueError):

click.echo("Failed to export!")

elif conf.OUTPUT\_EXT == ".json":

if conf.COMPRESS:

cases.to\_json(conf.OUTPUT\_PATH, orient='table', compression="zip")

else:

cases.to\_json(conf.OUTPUT\_PATH, orient='table')

elif conf.OUTPUT\_EXT == ".csv":

if conf.COMPRESS:

cases.to\_csv(conf.OUTPUT\_PATH, escapechar='\\', compression="zip")

else:

cases.to\_csv(conf.OUTPUT\_PATH, escapechar='\\')

elif conf.OUTPUT\_EXT == ".md":

cases.to\_markdown(conf.OUTPUT\_PATH)

elif conf.OUTPUT\_EXT == ".txt":

cases.to\_string(conf.OUTPUT\_PATH)

elif conf.OUTPUT\_EXT == ".dta":

cases.to\_stata(conf.OUTPUT\_PATH)

elif conf.OUTPUT\_EXT == ".parquet":

if conf.COMPRESS:

cases.to\_parquet(conf.OUTPUT\_PATH, compression="brotli")

else:

cases.to\_parquet(conf.OUTPUT\_PATH)

else:

pd.Series([cases, fees, allcharges]).to\_string(conf.OUTPUT\_PATH)

else:

pass

complete(conf, cases, fees, allcharges)

return [cases, fees, allcharges]

def fees(conf):

"""

Return fee sheet with case number as DataFrame from batch

Args:

conf (pd.Series): Configuration object with paths and settings

Returns:

fees = pd.DataFrame({

CaseNumber: Full case number with county number,

Code: 4-digit fee code,

Payor: 3-4-digit Payor code,

AmtDue (float): Amount Due,

AmtPaid (float): Amount Paid,

Balance (float): Current Balance,

AmtHold: (float): Amount Hold

})

"""

fees = pd.DataFrame()

if conf.DEDUPE:

old = conf.QUEUE.shape[0]

conf.QUEUE = conf.QUEUE.drop\_duplicates()

dif = conf.QUEUE.shape[0] - old

if dif > 0 and conf.LOG:

click.secho(f"Removed {dif} duplicate cases from queue.",

fg='bright\_yellow', bold=True)

if not conf['NO\_BATCH']:

batches = batcher(conf)

else:

batches = [conf.QUEUE]

for i, c in enumerate(batches):

if i > 0:

click.echo(f"Finished batch {i}. Now reading batch {i+1} of {len(batches)}")

b = pd.DataFrame()

if conf.IS\_FULL\_TEXT:

b['AllPagesText'] = c

else:

tqdm.pandas(desc="PDF => Text")

b['AllPagesText'] = c.progress\_map(lambda x: getPDFText(x))

b['CaseNumber'] = b['AllPagesText'].map(lambda x: getCaseNumber(x))

tqdm.pandas(desc="Fee Sheets")

b['FeeOutputs'] = b['AllPagesText'].progress\_map(lambda x: getFeeSheet(x))

feesheet = b['FeeOutputs'].map(lambda x: x[6])

feesheet = feesheet.dropna()

fees = fees.dropna()

feesheet = feesheet.tolist() # -> [df, df, df]

feesheet = pd.concat(feesheet, axis=0, ignore\_index=True)

fees = pd.concat([fees, feesheet], axis=0,ignore\_index=True)

fees.fillna('', inplace=True)

fees = fees.convert\_dtypes()

if not conf.NO\_WRITE:

write(conf, fees)

complete(conf, fees)

return fees

def charges(conf, multi=False):

def cleanCat(x):

if len(x) > 1:

if "MISDEMEANOR" in x:

return "MISDEMEANOR"

elif "FELONY" in x:

return "FELONY"

elif "VIOLATION" in x:

return "VIOLATION"

else:

return x[1]

elif len(x) == 1:

return x[0]

else:

return pd.NaT

def segmentCharge(text):

cite\_split = re.split(r'[A-Z0-9]{3}\s{0,1}-[A-Z0-9]{3}\s{0,1}-[A-Z0-9]{3}\({0,1}?[A-Z]{0,1}?\){0,1}?\.{0,1}?\d{0,1}?', text)

if len(cite\_split) > 1:

return [cite\_split[0][9:], cite\_split[1]]

else:

return [text,text]

df = map(conf, getCaseNumber, getCharges, names=['CaseNumber','Charges'])

df = df.explode('Charges') # num :: [ch, ch] -> num :: ch, num :: ch

df['Charges'] = df['Charges'].astype(str) # obj -> str

df['Sort'] = df['Charges'].map(lambda x: str(x)[9] if len(str(x)) > 9 else '') # charge sorter slices at first char after Code: if digit -> Disposition

df = df.dropna() # drop pd.NaT before bool() ambiguity TypeError

df['Disposition'] = df['Sort'].str.isdigit().astype(bool)

df['Filing'] = df['Disposition'].map(lambda x: not x).astype(bool)

df['Felony'] = df['Charges'].str.contains("FELONY")

df['Conviction'] = df['Charges'].map(lambda x: "GUILTY PLEA" in x or "CONVICTED" in x)

df['Num'] = df.Charges.str.slice(0,3)

df['Code'] = df.Charges.str.slice(4,9)

df['CourtActionDate'] = df['Charges'].str.findall(r'\d{1,2}/\d\d/\d\d\d\d') # -> [x]

df['Cite'] = df['Charges'].str.findall(r'[A-Z0-9]{3}-[A-Z0-9]{3}-[A-Z0-9]{3}\({0,1}[A-Z]{0,1}\){0,1}\.{0,1}\d{0,1}') # -> [x]

df['Cite'] = df['Cite'].map(lambda x: x[0] if len(x)>0 else x).astype(str) # [x] -> x

df = df.dropna()

df['CourtAction'] = df['Charges'].str.findall(r'(BOUND|GUILTY PLEA|WAIVED TO GJ|DISMISSED|TIME LAPSED|NOL PROSS|CONVICTED|INDICTED|DISMISSED|FORFEITURE|TRANSFER|REMANDED|WAIVED|ACQUITTED|WITHDRAWN|PETITION|PRETRIAL|COND\. FORF\.)')

df = df.explode('CourtAction')

df = df.reset\_index()

df = df.fillna('')

# split at cite - different parse based on filing/disposition

df['SegmentedCharges'] = df.Charges.map(lambda x: segmentCharge(x))

# whatever segment wasn't the description (now same for disposition and filing)

# try:

# except:

# pass

df['Description'] = df.index.map(lambda x: (df['SegmentedCharges'].iloc[x])[0] if not df['Disposition'].iloc[x] else (df['SegmentedCharges'].iloc[x])[1]).astype("string")

df['Description'] = df['Description'].str.strip()

df['OtherSegment'] = df.index.map(lambda x: (df['SegmentedCharges'].iloc[x])[0] if df['Disposition'].iloc[x] else (df['SegmentedCharges'].iloc[x])[1])

df['Category'] = df['OtherSegment'].str.findall(r'(ALCOHOL|BOND|CONSERVATION|DOCKET|DRUG|GOVERNMENT|HEALTH|MUNICIPAL|OTHER|PERSONAL|PROPERTY|SEX|TRAFFIC)')

df['TypeDescription'] = df['OtherSegment'].str.findall(r'(BOND|FELONY|MISDEMEANOR|OTHER|TRAFFIC|VIOLATION)')

# VRR

df['A\_S\_C\_NON\_DISQ'] = df['Description'].str.contains(r'(A ATT|ATTEMPT|S SOLICIT|CONSP)')

df['CERV\_MATCH'] = df['Code'].str.contains(r'(OSUA|EGUA|MAN1|MAN2|MANS|ASS1|ASS2|KID1|KID2|HUT1|HUT2|BUR1|BUR2|TOP1|TOP2|TPCS|TPCD|TPC1|TET2|TOD2|ROB1|ROB2|ROB3|FOR1|FOR2|FR2D|MIOB|TRAK|TRAG|VDRU|VDRY|TRAO|TRFT|TRMA|TROP|CHAB|WABC|ACHA|ACAL)')

df['PARDON\_DISQ\_MATCH'] = df['Code'].str.contains(r'(RAP1|RAP2|SOD1|SOD2|STSA|SXA1|SXA2|ECHI|SX12|CSSC|FTCS|MURD|MRDI|MURR|FMUR|PMIO|POBM|MIPR|POMA|INCE)')

df['PERM\_DISQ\_MATCH'] = df['Charges'].str.contains(r'(CM\d\d|CMUR)|(CAPITAL)')

df['CERV'] = df.index.map(

lambda x: df['CERV\_MATCH'].iloc[x] == True and df['A\_S\_C\_NON\_DISQ'].iloc[x] == False and df['Felony'].iloc[

x] == True).astype(bool)

df['Pardon'] = df.index.map(

lambda x: df['PARDON\_DISQ\_MATCH'].iloc[x] == True and df['A\_S\_C\_NON\_DISQ'].iloc[x] == False and df['Felony'].iloc[

x] == True).astype(bool)

df['Permanent'] = df.index.map(

lambda x: df['PERM\_DISQ\_MATCH'].iloc[x] == True and df['A\_S\_C\_NON\_DISQ'].iloc[x] == False and df['Felony'].iloc[

x] == True).astype(bool)

# type conversions

df['Category'] = df['Category'].map(lambda x: cleanCat(x))

df['TypeDescription'] = df['TypeDescription'].map(lambda x: cleanCat(x))

df = df.drop(columns=['Sort','SegmentedCharges','OtherSegment','A\_S\_C\_NON\_DISQ','PARDON\_DISQ\_MATCH','PERM\_DISQ\_MATCH','CERV\_MATCH'])

try:

df.drop(columns=['index'])

except:

pass

has\_num = df['Num'].map(lambda x: "0" in str(x))

df = df[has\_num]

df['Cite'] = df['Cite'].map(lambda x: '' if x == "[]" or [] else x)

df['CourtActionDate'] = df['CourtActionDate'].map(lambda x: x[0] if len(x)>0 else pd.NaT) # [x]->x or []->nan

df = df.fillna('')

if conf.TABLE == "filing":

is\_disp = df['Disposition']

is\_filing = is\_disp.map(lambda x: False if x == True else True)

df = df[is\_filing]

df.drop(columns=['CourtAction', 'CourtActionDate'], inplace=True)

if conf.TABLE == "disposition":

is\_disp = df.Disposition.map(lambda x: True if x == True else False)

df = df[is\_disp]

if conf.NO\_WRITE:

return df

write(conf, df)

if not multi:

complete(conf, df)

return df

## GETTERS

def getPDFText(path: str) -> str:

"""Returns PyPDF2 extract\_text() outputs for all pages from path

Args:

path (str): Description

Returns:

str: Description

"""

text = ""

pdf = PyPDF2.PdfReader(path)

for pg in pdf.pages:

text += pg.extract\_text()

return text

def getCaseNumber(text: str):

"""Returns full case number with county number prefix from case text

Args:

text (str): Description

Returns:

TYPE: Description

"""

try:

county: str = re.search(r'(?:County\: )(\d{2})(?:Case)', str(text)).group(1).strip()

case\_num: str = county + "-" + re.search(r'(\w{2}\-\d{4}-\d{6}.\d{2})', str(text)).group(1).strip()

return case\_num

except (IndexError, AttributeError):

return ""

def getName(text: str):

"""Returns name from case text

Args:

text (str): Description

Returns:

TYPE: Description

"""

name = ""

if bool(re.search(r'(?a)(VS\.|V\.{1})(.+)(Case)\*', text, re.MULTILINE)):

name = re.search(r'(?a)(VS\.|V\.{1})(.+)(Case)\*', text, re.MULTILINE).group(2).replace("Case Number:",

"").strip()

else:

if bool(re.search(r'(?:DOB)(.+)(?:Name)', text, re.MULTILINE)):

name = re.search(r'(?:DOB)(.+)(?:Name)', text, re.MULTILINE).group(1).replace(":", "").replace(

"Case Number:", "").strip()

return name

def getDOB(text: str):

"""Returns DOB from case text

Args:

text (str): Description

Returns:

TYPE: Description

"""

dob = ""

if bool(re.search(r'(\d{2}/\d{2}/\d{4})(?:.{0,5}DOB\:)', str(text), re.DOTALL)):

dob: str = re.search(r'(\d{2}/\d{2}/\d{4})(?:.{0,5}DOB\:)', str(text), re.DOTALL).group(1)

return dob

def getTotalAmtDue(text: str):

"""Returns total amt due from case text

Args:

text (str): Description

Returns:

TYPE: Description

"""

try:

trowraw = re.findall(r'(Total.\*\$.\*)', str(text), re.MULTILINE)[0]

totalrow = re.sub(r'[^0-9|\.|\s|\$]', "", trowraw)

if len(totalrow.split("$")[-1]) > 5:

totalrow = totalrow.split(" . ")[0]

tdue = totalrow.split("$")[1].strip().replace("$", "").replace(",", "").replace(" ", "")

except IndexError:

tdue = ""

return tdue

def getAddress(text: str):

"""Returns address from case text

Args:

text (str): Description

Returns:

TYPE: Description

"""

try:

street\_addr = re.search(r'(Address 1\:)(.+)(?:Phone)\*?', str(text), re.MULTILINE).group(2).strip()

except (IndexError, AttributeError):

street\_addr = ""

try:

zip\_code = re.search(r'(Zip\: )(.+)', str(text), re.MULTILINE).group(2).strip()

except (IndexError, AttributeError):

zip\_code = ""

try:

city = re.search(r'(City\: )(.\*)(State\: )(.\*)', str(text), re.MULTILINE).group(2).strip()

except (IndexError, AttributeError):

city = ""

try:

state = re.search(r'(?:City\: ).\*(?:State\: ).\*', str(text), re.MULTILINE).group(4).strip()

except (IndexError, AttributeError):

state = ""

address = street\_addr + " " + city + ", " + state + " " + zip\_code

if len(address) < 5:

address = ""

address = address.replace("00000-0000", "").replace("%", "").strip()

address = re.sub(r'([A-Z]{1}[a-z]+)', '', address)

return address

def getRace(text: str):

"""Return race from case text

Args:

text (str): Description

Returns:

TYPE: Description

"""

racesex = re.search(r'(B|W|H|A)\/(F|M)(?:Alias|XXX)', str(text))

race = racesex.group(1).strip()

return race

def getSex(text: str):

"""Return sex from case text

Args:

text (str): Description

Returns:

TYPE: Description

"""

racesex = re.search(r'(B|W|H|A)\/(F|M)(?:Alias|XXX)', str(text))

sex = racesex.group(2).strip()

return sex

def getNameAlias(text: str):

"""Return name from case text

Args:

text (str): Description

Returns:

TYPE: Description

"""

name = ""

if bool(re.search(r'(?a)(VS\.|V\.{1})(.{5,1000})(Case)\*', text, re.MULTILINE)):

name = re.search(r'(?a)(VS\.|V\.{1})(.{5,1000})(Case)\*', text, re.MULTILINE).group(2).replace("Case Number:",

"").strip()

else:

if bool(re.search(r'(?:DOB)(.{5,1000})(?:Name)', text, re.MULTILINE)):

name = re.search(r'(?:DOB)(.{5,1000})(?:Name)', text, re.MULTILINE).group(1).replace(":", "").replace(

"Case Number:", "").strip()

try:

alias = re.search(r'(SSN)(.{5,75})(Alias)', text, re.MULTILINE).group(2).replace(":", "").replace("Alias 1",

"").strip()

except (IndexError, AttributeError):

alias = ""

if alias == "":

return name

else:

return name + "\r" + alias

def getCaseInfo(text: str):

"""Returns case information from case text -> cases table

Args:

text (str): Description

Returns:

TYPE: Description

"""

case\_num = ""

name = ""

alias = ""

race = ""

sex = ""

try:

county: str = re.search(r'(?:County\: )(\d{2})(?:Case)', str(text)).group(1).strip()

case\_num: str = county + "-" + re.search(r'(\w{2}\-\d{4}-\d{6}.\d{2})', str(text)).group(1).strip()

except (IndexError, AttributeError):

pass

if bool(re.search(r'(?a)(VS\.|V\.{1})(.{5,1000})(Case)\*', text, re.MULTILINE)):

name = re.search(r'(?a)(VS\.|V\.{1})(.{5,1000})(Case)\*', text, re.MULTILINE).group(2).replace("Case Number:",

"").strip()

else:

if bool(re.search(r'(?:DOB)(.{5,1000})(?:Name)', text, re.MULTILINE)):

name = re.search(r'(?:DOB)(.{5,1000})(?:Name)', text, re.MULTILINE).group(1).replace(":", "").replace(

"Case Number:", "").strip()

try:

alias = re.search(r'(SSN)(.{5,75})(Alias)', text, re.MULTILINE).group(2).replace(":", "").replace("Alias 1",

"").strip()

except (IndexError, AttributeError):

pass

else:

pass

try:

dob: str = re.search(r'(\d{2}/\d{2}/\d{4})(?:.{0,5}DOB\:)', str(text), re.DOTALL).group(1)

phone: str = re.search(r'(?:Phone\:)(.\*?)(?:Country)', str(text), re.DOTALL).group(1).strip()

phone = re.sub(r'[^0-9]', '', phone)

if len(phone) < 7:

phone = ""

if len(phone) > 10 and phone[-3:] == "000":

phone = phone[0:9]

except (IndexError, AttributeError):

dob = ""

phone = ""

try:

racesex = re.search(r'(B|W|H|A)\/(F|M)(?:Alias|XXX)', str(text))

race = racesex.group(1).strip()

sex = racesex.group(2).strip()

except (IndexError, AttributeError):

pass

try:

street\_addr = re.search(r'(Address 1\:)(.+)(?:Phone)\*?', str(text), re.MULTILINE).group(2).strip()

except (IndexError, AttributeError):

street\_addr = ""

try:

zip\_code = re.search(r'(Zip\: )(.+)', str(text), re.MULTILINE).group(2).strip()

except (IndexError, AttributeError):

zip\_code = ""

try:

city = re.search(r'(City\: )(.\*)(State\: )(.\*)', str(text), re.MULTILINE).group(2).strip()

except (IndexError, AttributeError):

city = ""

try:

state = re.search(r'(?:City\: ).\*(?:State\: ).\*', str(text), re.MULTILINE).group(4).strip()

except (IndexError, AttributeError):

state = ""

address = street\_addr + " " + city + ", " + state + " " + zip\_code

if len(address) < 5:

address = ""

address = address.replace("00000-0000", "").replace("%", "").strip()

address = re.sub(r'([A-Z]{1}[a-z]+)', '', address)

case = [case\_num, name, alias, dob, race, sex, address, phone]

return case

def getPhone(text: str):

"""Return phone number from case text

Args:

text (str): Description

Returns:

TYPE: Description

"""

try:

phone: str = re.search(r'(?:Phone\:)(.\*?)(?:Country)', str(text), re.DOTALL).group(1).strip()

phone = re.sub(r'[^0-9]', '', phone)

if len(phone) < 7:

phone = ""

if len(phone) > 10 and phone[-3:] == "000":

phone = phone[0:9]

except (IndexError, AttributeError):

phone = ""

return phone

def getFeeSheet(text: str):

"""

Return fee sheet and fee summary outputs from case text

List: [tdue, tbal, d999, owe\_codes, codes, allrowstr, feesheet]

feesheet = feesheet[['CaseNumber', 'FeeStatus', 'AdminFee', 'Total', 'Code', 'Payor', 'AmtDue', 'AmtPaid', 'Balance', 'AmtHold']]

Args:

text (str): Description

Returns:

TYPE: Description

"""

actives = re.findall(r'(ACTIVE.\*\$.\*)', str(text))

if len(actives) == 0:

return [np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan, np.nan]

else:

try:

trowraw = re.findall(r'(Total.\*\$.\*)', str(text), re.MULTILINE)[0]

totalrow = re.sub(r'[^0-9|\.|\s|\$]', "", trowraw)

if len(totalrow.split("$")[-1]) > 5:

totalrow = totalrow.split(" . ")[0]

tbal = totalrow.split("$")[3].strip().replace("$", "").replace(",", "").replace(" ", "").strip()

tdue = totalrow.split("$")[1].strip().replace("$", "").replace(",", "").replace(" ", "").strip()

tpaid = totalrow.split("$")[2].strip().replace("$", "").replace(",", "").replace(" ", "").strip()

thold = totalrow.split("$")[4].strip().replace("$", "").replace(",", "").replace(" ", "").strip()

except IndexError:

totalrow = ""

tbal = ""

tdue = ""

tpaid = ""

thold = ""

fees = pd.Series(actives, dtype=str)

fees\_noalpha = fees.map(lambda x: re.sub(r'[^0-9|\.|\s|\$]', "", x))

srows = fees.map(lambda x: x.strip().split(" "))

drows = fees\_noalpha.map(lambda x: x.replace(",", "").split("$"))

coderows = srows.map(lambda x: str(x[5]).strip() if len(x) > 5 else "")

payorrows = srows.map(lambda x: str(x[6]).strip() if len(x) > 6 else "")

amtduerows = drows.map(lambda x: str(x[1]).strip() if len(x) > 1 else "")

amtpaidrows = drows.map(lambda x: str(x[2]).strip() if len(x) > 2 else "")

balancerows = drows.map(lambda x: str(x[-1]).strip() if len(x) > 5 else "")

amtholdrows = drows.map(lambda x: str(x[3]).strip() if len(x) > 5 else "")

amtholdrows = amtholdrows.map(lambda x: x.split(" ")[0].strip() if " " in x else x)

adminfeerows = fees.map(lambda x: x.strip()[7].strip() if 'N' else '')

feesheet = pd.DataFrame({'CaseNumber': getCaseNumber(text), 'Total': '', 'FeeStatus': 'ACTIVE', 'AdminFee': adminfeerows.tolist(), 'Code': coderows.tolist(), 'Payor': payorrows.tolist(), 'AmtDue': amtduerows.tolist(), 'AmtPaid': amtpaidrows.tolist(), 'Balance': balancerows.tolist(), 'AmtHold': amtholdrows.tolist() })

totalrdf = pd.DataFrame({'CaseNumber': getCaseNumber(text), 'Total': 'TOTAL', 'FeeStatus': '', 'AdminFee': '', 'Code': '', 'Payor': '', 'AmtDue': tdue, 'AmtPaid': tpaid, 'Balance': tbal, 'AmtHold': thold },index=[0])

feesheet = feesheet.dropna()

feesheet = pd.concat([feesheet, totalrdf], axis = 0, ignore\_index=True)

try:

d999 = feesheet[feesheet['Code'] == 'D999']['Balance']

except (TypeError, IndexError):

d999 = ""

owe\_codes = " ".join(feesheet['Code'][feesheet.Balance.str.len() > 0])

codes = " ".join(feesheet['Code'])

allrows = actives

allrows.append(totalrow)

allrowstr = "\n".join(allrows)

feesheet = feesheet[

['CaseNumber', 'FeeStatus', 'AdminFee', 'Total', 'Code', 'Payor', 'AmtDue', 'AmtPaid', 'Balance',

'AmtHold']]

feesheet['AmtDue'] = pd.to\_numeric(feesheet['AmtDue'], errors='coerce')

feesheet['AmtPaid'] = pd.to\_numeric(feesheet['AmtPaid'], errors='coerce')

feesheet['Balance'] = pd.to\_numeric(feesheet['Balance'], errors='coerce')

feesheet['AmtHold'] = pd.to\_numeric(feesheet['AmtHold'], errors='coerce')

return [tdue, tbal, d999, owe\_codes, codes, allrowstr, feesheet]

def getFeeCodes(text: str):

"""Return fee codes from case text

Args:

text (str): Description

Returns:

TYPE: Description

"""

return getFeeSheet(text)[4]

def getFeeCodesOwed(text: str):

"""Return fee codes with positive balance owed from case text

Args:

text (str): Description

Returns:

TYPE: Description

"""

return getFeeSheet(text)[3]

def getTotals(text: str):

"""Return totals from case text -> List: [totalrow,tdue,tpaid,tdue,thold]

Args:

text (str): Description

Returns:

TYPE: Description

"""

try:

trowraw = re.findall(r'(Total.\*\$.\*)', str(text), re.MULTILINE)[0]

totalrow = re.sub(r'[^0-9|\.|\s|\$]', "", trowraw)

if len(totalrow.split("$")[-1]) > 5:

totalrow = totalrow.split(" . ")[0]

tbal = totalrow.split("$")[3].strip().replace("$", "").replace(",", "").replace(" ", "")

tdue = totalrow.split("$")[1].strip().replace("$", "").replace(",", "").replace(" ", "")

tpaid = totalrow.split("$")[2].strip().replace("$", "").replace(",", "").replace(" ", "")

thold = totalrow.split("$")[4].strip().replace("$", "").replace(",", "").replace(" ", "")

try:

tdue = pd.to\_numeric(tdue, 'coerce')

tpaid = pd.to\_numeric(tpaid, 'coerce')

thold = pd.to\_numeric(thold, 'coerce')

except:

pass

except IndexError:

totalrow = 0

tdue = 0

tpaid = 0

thold = 0

return [totalrow, tdue, tpaid, tdue, thold]

def getTotalBalance(text: str):

"""Return total balance from case text

Args:

text (str): Description

Returns:

TYPE: Description

"""

try:

trowraw = re.findall(r'(Total.\*\$.\*)', str(text), re.MULTILINE)[0]

totalrow = re.sub(r'[^0-9|\.|\s|\$]', "", trowraw)

if len(totalrow.split("$")[-1]) > 5:

totalrow = totalrow.split(" . ")[0]

tbal = totalrow.split("$")[3].strip().replace("$", "").replace(",", "").replace(" ", "")

except:

tbal = ""

return str(tbal)

def getPaymentToRestore(text: str):

"""

Return (total balance - total d999) from case text -> str

Does not mask misc balances!

Args:

text (str): Description

Returns:

TYPE: Description

"""

totalrow = "".join(re.findall(r'(Total.\*\$.+\$.+\$.+)', str(text), re.MULTILINE)) if bool(

re.search(r'(Total.\*\$.\*)', str(text), re.MULTILINE)) else "0"

try:

tbalance = totalrow.split("$")[3].strip().replace("$", "").replace(",", "").replace(" ", "").strip()

try:

tbal = pd.Series([tbalance]).astype(float)

except ValueError:

tbal = 0.0

except (IndexError, TypeError):

tbal = 0.0

try:

d999raw = re.search(r'(ACTIVE.\*?D999\$.\*)', str(text), re.MULTILINE).group() if bool(

re.search(r'(ACTIVE.\*?D999\$.\*)', str(text), re.MULTILINE)) else "0"

d999 = pd.Series([d999raw]).astype(float)

except (IndexError, TypeError):

d999 = 0.0

t\_out = pd.Series(tbal - d999).astype(float).values[0]

return str(t\_out)

def getBalanceByCode(text: str, code: str):

"""

Return balance by code from case text -> str

Args:

text (str): Description

code (str): Description

Returns:

TYPE: Description

"""

actives = re.findall(r'(ACTIVE.\*\$.\*)', str(text))

fees = pd.Series(actives, dtype=str)

fees\_noalpha = fees.map(lambda x: re.sub(r'[^0-9|\.|\s|\$]', "", x))

srows = fees.map(lambda x: x.strip().split(" "))

drows = fees\_noalpha.map(lambda x: x.replace(",", "").split("$"))

coderows = srows.map(lambda x: str(x[5]).strip() if len(x) > 5 else "")

balancerows = drows.map(lambda x: str(x[-1]).strip() if len(x) > 5 else "")

codemap = pd.DataFrame({

'Code': coderows,

'Balance': balancerows

})

matches = codemap[codemap.Code == code].Balance

return str(matches.sum())

def getAmtDueByCode(text: str, code: str):

"""

Return total amt due from case text -> str

Args:

text (str): Description

code (str): Description

Returns:

TYPE: Description

"""

actives = re.findall(r'(ACTIVE.\*\$.\*)', str(text))

fees = pd.Series(actives, dtype=str)

fees\_noalpha = fees.map(lambda x: re.sub(r'[^0-9|\.|\s|\$]', "", x))

srows = fees.map(lambda x: x.strip().split(" "))

drows = fees\_noalpha.map(lambda x: x.replace(",", "").split("$"))

coderows = srows.map(lambda x: str(x[5]).strip() if len(x) > 5 else "")

payorrows = srows.map(lambda x: str(x[6]).strip() if len(x) > 6 else "")

amtduerows = drows.map(lambda x: str(x[1]).strip() if len(x) > 1 else "")

codemap = pd.DataFrame({

'Code': coderows,

'Payor': payorrows,

'AmtDue': amtduerows

})

codemap.AmtDue = codemap.AmtDue.map(lambda x: pd.to\_numeric(x, 'coerce'))

due = codemap.AmtDue[codemap.Code == code]

return str(due)

def getAmtPaidByCode(text: str, code: str):

"""

Return total amt paid from case text -> str

Args:

text (str): Description

code (str): Description

Returns:

TYPE: Description

"""

actives = re.findall(r'(ACTIVE.\*\$.\*)', str(text))

fees = pd.Series(actives, dtype=str)

fees\_noalpha = fees.map(lambda x: re.sub(r'[^0-9|\.|\s|\$]', "", x))

srows = fees.map(lambda x: x.strip().split(" "))

drows = fees\_noalpha.map(lambda x: x.replace(",", "").split("$"))

coderows = srows.map(lambda x: str(x[5]).strip() if len(x) > 5 else "")

payorrows = srows.map(lambda x: str(x[6]).strip() if len(x) > 6 else "")

amtpaidrows = drows.map(lambda x: str(x[2]).strip() if len(x) > 2 else "")

codemap = pd.DataFrame({

'Code': coderows,

'Payor': payorrows,

'AmtPaid': amtpaidrows

})

codemap.AmtPaid = codemap.AmtPaid.map(lambda x: pd.to\_numeric(x, 'coerce'))

paid = codemap.AmtPaid[codemap.Code == code]

return str(paid)

def getCaseYear(text):

"""

Return case year

Args:

text (TYPE): Description

Returns:

TYPE: Description

"""

cnum = getCaseNumber(text)

return float(cnum[6:10])

def getCounty(text):

"""

Return county

Args:

text (TYPE): Description

Returns:

TYPE: Description

"""

cnum = getCaseNumber(text)

return int(cnum[0:2])

def getLastName(text):

"""

Return last name

Args:

text (TYPE): Description

Returns:

TYPE: Description

"""

name = getName(text)

return name.split(" ")[0].strip()

def getFirstName(text):

"""

Return first name

Args:

text (TYPE): Description

Returns:

TYPE: Description

"""

name = getName(text)

if len(name.split(" ")) > 1:

return name.split(" ")[1].strip()

else:

return name

def getMiddleName(text):

"""

Return middle name or initial

Args:

text (TYPE): Description

Returns:

TYPE: Description

"""

name = getName(text)

if len(name.split(" ")) > 2:

return name.split(" ")[2].strip()

else:

return ""

def getCharges(text):

b = re.findall(r'(\d{3}\s{1}[A-Z0-9]{4}.{1,200}?.{3}-.{3}-.{3}.{10,75})', text, re.MULTILINE)

b = [re.sub(r'[A-Z][a-z][a-z\s]+.+','',x) for x in b]

b = [re.sub(r'\:\s\:.+','',x) for x in b]

# b = [re.sub(r'^\d{3}\n.+','',x) for x in b]

# b = ['' if re.search(r'\d{1,2}\:{1,2}\s\D{2}',x) else x for x in b]

# btest = [True if len(str(x))>9 else False for x in b]

# b = pd.Series(b)

# b = b[btest]

return b

## FETCH

def fetch(listpath, path, cID, uID, pwd, qmax=0, qskip=0, speed=1, no\_log=False, no\_update=False, debug=False):

"""

Use headers NAME, PARTY\_TYPE, SSN, DOB, COUNTY, DIVISION, CASE\_YEAR, and FILED\_BEFORE in an Excel spreadsheet to submit a list of queries for Alacorder to fetch.

USE WITH CHROME (TESTED ON MACOS)

KEEP YOUR COMPUTER POWERED ON AND CONNECTED TO THE INTERNET.

Args:

listpath: (path-like obj) Query template path / input path

path: (path-like obj) Path to output/downloads directory

cID (str): Alacourt.com Customer ID

uID (str): Alacourt.com User ID

pwd (str): Alacourt.com Password

qmax (int, optional): Max queries to pull from inputs

qskip (int, optional): Skip top n queries in inputs

speed (int, optional): fetch rate multiplier

no\_log (bool, optional): Do not print logs to console

no\_update (bool, optional): Do not update input query file with completion status

debug (bool, optional): Print detailed logs to console

Returns:

[driver, query\_out, query\_writer]:

driver[0]: Google Chrome WebDriver() object

query\_out[1]: (pd.Series) fetch queue

query\_writer[2]: (pd.DataFrame) Updated input query file

"""

rq = readPartySearchQuery(listpath, qmax, qskip, no\_log)

query = pd.DataFrame(rq[0]) # for fetch - only search columns

query\_writer = pd.DataFrame(rq[1]) # original sheet for write completion

incomplete = query.RETRIEVED\_ON.map(lambda x: True if x == "" else False)

query = query[incomplete]

options = webdriver.ChromeOptions()

options.add\_experimental\_option('prefs', {

"download.default\_directory": path, #Change default directory for downloads

"download.prompt\_for\_download": False, #To auto download the file

"download.directory\_upgrade": True,

"plugins.always\_open\_pdf\_externally": True #It will not display PDF directly in chrome

})

# start browser session, login

if not no\_log:

click.secho("Starting browser... Do not close while in progress!",fg='bright\_yellow',bold=True)

driver = webdriver.Chrome(options=options)

login(driver, cID, uID, pwd, speed)

if not no\_log:

click.secho("Authentication successful. Fetching cases via party search...",fg='bright\_green')

# search, retrieve from URL, download to path

for i, n in enumerate(query.index):

if driver.current\_url == "https://v2.alacourt.com/frmlogin.aspx":

login(driver, cID, uID, pwd, speed, no\_log)

driver.implicitly\_wait(4/speed)

results = party\_search(driver, name=query.NAME[n], party\_type=query.PARTY\_TYPE[n], ssn=query.SSN[n], dob=query.DOB[n], county=query.COUNTY[n], division=query.DIVISION[n], case\_year=query.CASE\_YEAR[n], filed\_before=query.FILED\_BEFORE[n], filed\_after=query.FILED\_AFTER[n], speed=speed, no\_log=no\_log)

driver.implicitly\_wait(4/speed)

if len(results) == 0:

query\_writer['RETRIEVED\_ON'][n] = str(math.floor(time.time()))

query\_writer['CASES\_FOUND'][n] = "0"

if not no\_log:

click.echo(f"{query.NAME[n]}: Found no results.")

continue

results = pd.Series(results)

tqdm.pandas(desc=query.NAME[n])

results.progress\_map(lambda x: downloadPDF(driver, x))

if not no\_update:

query\_writer['RETRIEVED\_ON'][n] = str(math.floor(time.time()))

query\_writer['CASES\_FOUND'][n] = str(len(results))

query\_writer = query\_writer.convert\_dtypes()

query\_writer.to\_excel(listpath,sheet\_name="PartySearchQuery",index=False)

return [driver, query\_writer]

def party\_search(driver, name = "", party\_type = "", ssn="", dob="", county="", division="", case\_year="", filed\_before="", filed\_after="", speed=1, no\_log=False, debug=False):

"""

Collect PDFs via SJIS Party Search Form from Alacourt.com

Returns list of URLs for downloadPDF() to download

Args:

driver (WebDriver): selenium/chrome web driver object

name (str, optional): Name (LAST FIRST)

party\_type (str, optional): "Defendants" | "Plaintiffs" | "ALL"

ssn (str, optional): Social Security Number

dob (str, optional): Date of Birth

county (str, optional): County

division (str, optional): "All Divisions"

"Criminal Only"

"Civil Only"

"CS - CHILD SUPPORT"

"CV - CIRCUIT - CIVIL"

"CC - CIRCUIT - CRIMINAL"

"DV - DISTRICT - CIVIL"

"DC - DISTRICT - CRIMINAL"

"DR - DOMESTIC RELATIONS"

"EQ - EQUITY-CASES"

"MC - MUNICIPAL-CRIMINAL"

"TP - MUNICIPAL-PARKING"

"SM - SMALL CLAIMS"

"TR - TRAFFIC"

case\_year (str, optional): YYYY

filed\_before (str, optional): M/DD/YYYY

filed\_after (str, optional): M/DD/YYYY

speed (int, optional): fetch rate multiplier

no\_log (bool, optional): Do not print logs.

debug (bool, optional): Print detailed logs.

Returns:

URL list to PDFs

"""

speed = speed \* 1.5

if "frmIndexSearchForm" not in driver.current\_url:

driver.get("https://v2.alacourt.com/frmIndexSearchForm.aspx")

driver.implicitly\_wait(5/speed)

# connection error

try:

party\_name\_box = driver.find\_element(by=By.NAME,value="ctl00$ContentPlaceHolder1$txtName")

except selenium.common.exceptions.NoSuchElementException:

if not no\_log:

click.secho("Connection error. Attempting reconnection...",fg='red')

driver.refresh()

driver.implicitly\_wait(10/speed)

party\_name\_box = driver.find\_element(by=By.NAME,value="ctl00$ContentPlaceHolder1$txtName")

if not no\_log:

click.secho("Successfully connected and logged into Alacourt!",fg='green',bold=True)

# field search

if name != "":

party\_name\_box.send\_keys(name)

if ssn != "":

ssn\_box = driver.find\_element(by=By.NAME, value="ctl00$ContentPlaceHolder1$txtSSN")

ssn\_box.send\_keys(ssn)

if dob != "":

date\_of\_birth\_box = driver.find\_element(by=By.NAME,value="ctl00$ContentPlaceHolder1$txtDOB")

date\_of\_birth\_box.send\_keys(dob)

if party\_type != "":

party\_type\_select = driver.find\_element(by=By.NAME, value="ctl00$ContentPlaceHolder1$rdlPartyType")

pts = Select(party\_type\_select)

if party\_type == "plaintiffs":

pts.select\_by\_visible\_text("Plaintiffs")

if party\_type == "defendants":

pts.select\_by\_visible\_text("Defendants")

if party\_type == "all":

pts.select\_by\_visible\_text("ALL")

if county != "":

county\_select = driver.find\_element(by=By.NAME, value="ctl00$ContentPlaceHolder1$ddlCounties")

scounty = Select(county\_select)

scounty.select\_by\_visible\_text(county)

if division != "":

division\_select = driver.find\_element(by=By.NAME, value="ctl00$ContentPlaceHolder1$UcddlDivisions1$ddlDivision")

sdivision = Select(division\_select)

sdivision.select\_by\_visible\_text(division)

if case\_year != "":

case\_year\_select = driver.find\_element(by=By.NAME, value="ctl00$ContentPlaceHolder1$ddlCaseYear")

scase\_year = Select(case\_year\_select)

scase\_year.select\_by\_visible\_text(case\_year)

no\_records\_select = driver.find\_element(by=By.NAME, value="ctl00$ContentPlaceHolder1$ddlNumberOfRecords")

sno\_records = Select(no\_records\_select)

sno\_records.select\_by\_visible\_text("1000")

if filed\_before != "":

filed\_before\_box = driver.find\_element(by=By.NAME, value="ctl00$ContentPlaceHolder1$txtFrom")

filed\_before\_box.send\_keys(filed\_before)

if filed\_after != "":

filed\_after\_box = driver.find\_element(by=By.NAME, value="ctl00$ContentPlaceHolder1$txtTo")

filed\_after\_box.send\_keys(filed\_after)

driver.implicitly\_wait(1/speed)

# submit search

search\_button = driver.find\_element(by=By.ID,value="searchButton")

driver.implicitly\_wait(1/speed)

try:

search\_button.click()

except:

driver.implicitly\_wait(5/speed)

time.sleep(5)

if debug:

click.echo("Submitted party search form...")

driver.implicitly\_wait(1/speed)

# count pages

try:

page\_counter = driver.find\_element(by=By.ID,value="ContentPlaceHolder1\_dg\_tcPageXofY").text

pages = int(page\_counter.strip()[-1])

except:

pages = 1

# count results

try:

results\_indicator = driver.find\_element(by=By.ID, value="ContentPlaceHolder1\_lblResultCount")

results\_count = int(results\_indicator.text.replace("Search Results: ","").replace(" records returned.","").strip())

if results\_count == 1000 and debug or no\_log:

click.echo(f"Max records (1000) returned for party {name}!")

except:

pass

if debug:

click.echo(f"Found {results\_count} results, fetching URLs and downloading PDFs...")

# get PDF links from each page

pdflinks = []

i = 0

for i in range(0,pages):

driver.implicitly\_wait(0.5/speed)

hovers = driver.find\_elements(By.CLASS\_NAME, "menuHover")

for x in hovers:

try:

a = x.get\_attribute("href")

if "PDF" in a:

pdflinks.append(a)

except:

pass

driver.implicitly\_wait(0.5/speed)

try:

pager\_select = Select(driver.find\_element(by=By.NAME, value="ctl00$ContentPlaceHolder1$dg$ctl18$ddlPages"))

next\_pg = int(pager\_select.text) + 1

driver.implicitly\_wait(0.5/speed)

except:

try:

driver.implicitly\_wait(0.5/speed)

time.sleep(0.5/speed)

next\_button = driver.find\_element(by=By.ID, value = "ContentPlaceHolder1\_dg\_ibtnNext")

next\_button.click()

except:

continue

return pdflinks

def downloadPDF(driver, url, no\_log=False, cID="", uID="", pwd=""):

"""

With (driver), download PDF at (url)

Args:

driver (WebDriver): Google Chrome selenium.WebDriver() object

url (TYPE): Description

no\_log (bool, optional): Description

Deleted Parameters:

speed (int, optional): fetch rate multiplier

"""

if driver.current\_url == "https://v2.alacourt.com/frmlogin.aspx":

login(driver, cID, uID, pwd, speed, no\_log)

a = driver.get(url)

driver.implicitly\_wait(0.5)

def login(driver, cID, username, pwd, speed, no\_log=False, path=""):

"""Login to Alacourt.com using (driver) and auth (cID, username, pwd) at (speed) for browser download to directory at (path)

Args:

driver (WebDriver): Google Chrome selenium.WebDriver() object

cID (str): Alacourt.com Customer ID

username (str): Alacourt.com User ID

pwd (str): Alacourt.com Password

speed (TYPE): fetch rate multiplier

no\_log (bool, optional): Do not print logs

path (str, optional): Set browser download path

Returns:

driver (WebDriver): Google Chrome selenium.WebDriver() object

"""

if driver == None:

options = webdriver.ChromeOptions()

options.add\_experimental\_option('prefs', {

"download.default\_directory": path, #Change default directory for downloads

"download.prompt\_for\_download": False, #To auto download the file

"download.directory\_upgrade": True,

"plugins.always\_open\_pdf\_externally": True #It will not display PDF directly in chrome

})

driver = webdriver.Chrome(options=options)

if not no\_log:

click.echo("Connecting to Alacourt...")

login\_screen = driver.get("https://v2.alacourt.com/frmlogin.aspx")

if not no\_log:

click.echo("Logging in...")

driver.implicitly\_wait(0.5/speed)

cID\_box = driver.find\_element(by=By.NAME,

value="ctl00$ContentPlaceHolder$txtCusid")

username\_box = driver.find\_element(by=By.NAME, value="ctl00$ContentPlaceHolder$txtUserId")

pwd\_box = driver.find\_element(by=By.NAME, value="ctl00$ContentPlaceHolder$txtPassword")

login\_button = driver.find\_element(by=By.ID, value="ContentPlaceHolder\_btLogin")

cID\_box.send\_keys(cID)

username\_box.send\_keys(username)

pwd\_box.send\_keys(pwd)

driver.implicitly\_wait(1/speed)

login\_button.click()

driver.implicitly\_wait(1/speed)

try:

continueLogIn = driver.find\_element(by=By.NAME, value="ctl00$ContentPlaceHolder$btnContinueLogin")

continueLogIn.click()

except:

pass

driver.get("https://v2.alacourt.com/frmIndexSearchForm.aspx")

if not no\_log:

click.secho("Successfully connected and logged into Alacourt!",fg='bright\_green')

driver.implicitly\_wait(0.5/speed)

return driver

def readPartySearchQuery(path, qmax=0, qskip=0, speed=1, no\_log=False):

"""Reads and interprets query template spreadsheets for `alacorder fetch` to queue from. Use headers NAME, PARTY\_TYPE, SSN, DOB, COUNTY, DIVISION, CASE\_YEAR, and FILED\_BEFORE in an Excel spreadsheet, CSV, or JSON file to submit a list of queries for Alacorder to fetch.

Args:

path (TYPE): Description

qmax (int, optional): Description

qskip (int, optional): Description

speed (int, optional): Description

no\_log (bool, optional): Description

Returns:

[query\_out, writer\_df]:

query\_out: (pd.DataFrame) queue object for alac.fetch()

writer\_df: (pd.DataFrame) progress log to be written back to (path)

Raises:

Exception: Connection error!

"""

good = os.path.exists(path)

ext = os.path.splitext(path)[1]

if ext == ".xlsx" or ".xls":

query = pd.read\_excel(path, dtype=pd.StringDtype())

if ext == ".csv":

query = pd.read\_csv(path, dtype=pd.StringDtype())

if ext == ".json":

query = pd.read\_json(path, orient='table', dtype=pd.StringDtype())

if qskip > 0:

query = query.truncate(before=qskip)

if qmax > 0:

query = query.truncate(after=qmax+qskip)

writer\_df = pd.DataFrame(query)

if "RETRIEVED\_ON" not in writer\_df.columns:

writer\_df['RETRIEVED\_ON'] = pd.NaT

writer\_df['CASES\_FOUND'] = pd.NaT

query\_out = pd.DataFrame(columns=["NAME", "PARTY\_TYPE", "SSN", "DOB", "COUNTY", "DIVISION", "CASE\_YEAR", "NO\_RECORDS", "FILED\_BEFORE", "FILED\_AFTER", "RETRIEVED\_ON", "CASES\_FOUND"])

clist = []

for c in query.columns:

if c.upper().strip().replace(" ","\_") in ["NAME", "PARTY", "DATE\_OF\_BIRTH", "BIRTHDATE", "PARTY\_TYPE", "SSN", "DOB", "COUNTY", "DIVISION", "CASE\_YEAR", "NO\_RECORDS", "FILED\_BEFORE", "FILED\_AFTER", "RETRIEVED\_ON", "CASES\_FOUND"]:

ce = c.replace("DATE\_OF\_BIRTH","DOB").replace("BIRTHDATE","DOB").replace("PARTY","PARTY\_TYPE").replace("PARTY\_TYPE\_TYPE","PARTY\_TYPE").strip()

clist += [ce]

query\_out[c.upper().strip().replace(" ","\_")] = query[c]

query\_out[ce] = query[c]

clist = pd.Series(clist).drop\_duplicates().tolist()

if clist == []:

raise Exception("Invalid template! Use headers NAME, PARTY\_TYPE, SSN, DOB, COUNTY, DIVISION, CASE\_YEAR, and FILED\_BEFORE in a spreadsheet or JSON file to submit a list of queries for Alacorder to fetch.")

click.echo(f"Field columns {clist} identified in query file.")

query\_out = query\_out.fillna('')

return [query\_out, writer\_df]

## LOGS

def echo\_conf(input\_path, make, output\_path, overwrite, no\_write, dedupe, no\_prompt, compress):

"""

Logs configuration details to console

Args:

input\_path (TYPE): Description

make (TYPE): Description

output\_path (TYPE): Description

overwrite (TYPE): Description

no\_write (TYPE): Description

dedupe (TYPE): Description

no\_prompt (TYPE): Description

compress (TYPE): Description

Returns:

TYPE: Description

"""

return f"""{"ARCHIVE is enabled. Alacorder will write full text case archive to output path instead of data tables. " if make == "archive" else ''}{"NO-WRITE is enabled. Alacorder will NOT export outputs. " if no\_write else ''}{"OVERWRITE is enabled. Alacorder will overwrite existing files at output path! " if overwrite else ''}{"REMOVE DUPLICATES is enabled. At time of export, all duplicate cases will be removed from output. " if dedupe and make == "archive" else ''}{"NO\_PROMPT is enabled. All user confirmation prompts will be suppressed as if set to default by user." if no\_prompt else ''}{"COMPRESS is enabled. Alacorder will try to compress output file." if compress == True else ''}""".strip()

def pick\_table():

upick\_table = ('''

For compressed archive, enter:

[A] Full text archive

To export a data table, enter:

[B] Case Details

[C] Fee Sheets

[D] Charges (all)

[E] Charges (disposition only)

[F] Charges (filing only)

Enter selection to continue. [A-F]

''')

return upick\_table

def pick\_table\_only():

upick\_table\_only = ('''

To export a data table, enter:

[B] Case Details

[C] Fee Sheets

[D] Charges (all)

[E] Charges (disposition only)

[F] Charges (filing only)

Enter selection to continue. [B-F]

''')

return upick\_table\_only

def just\_table():

ujust\_table = ('''

EXPORT DATA TABLE: To export data table from case inputs, enter full output path. Use .xls or .xlsx to export all tables, or, if using another format (.csv, .json, .dta), select a table after entering output file path.

Enter path.

''')

return ujust\_table

def just\_archive():

ujust\_archive = ('''

EXPORT ARCHIVE: Compressed archives can store thousands of cases' data using a fraction of the original PDF storage. To export full text archive, enter full output path. Supported file extensions are archive.pkl.xz, archive.json(.zip), archive.csv(.zip), and archive.parquet.

Enter path.

''')

return ujust\_archive

def both():

uboth = ('''

EXPORT FULL TEXT ARCHIVE: To process case inputs into a full text archive (recommended), enter archive path below with file extension .pkl.xz or .json.zip.

EXPORT DATA TABLE: To export data table from case inputs, enter full output path. Use .xls or .xlsx to export all tables, or, if using another format (.csv, .json, .dta), select a table after entering output file path.

Enter output path.

''')

return click.style(uboth)

def title():

utitle = """ALACORDER beta 77

Alacorder retrieves case detail PDFs from Alacourt.com and processes them into text archives and data tables suitable for research purposes.

ACCEPTED /pdfs/path/ PDF directory

INPUTS: .pkl(.xz) Pickle archive

.json(.zip) JSON archive

.csv(.zip) CSV archive

.parquet Apache Parquet archive

Enter input path.

"""

return utitle

def smalltitle():

usmalltitle = """ALACORDER beta 77

Alacorder retrieves case detail PDFs from Alacourt.com and processes them into text archives and data tables suitable for research purposes.

"""

return usmalltitle

def text\_p():

utext\_p = ('''

Enter path to output text file (must be .txt).

''')

return utext\_p

def complete(conf, \*outputs):

"""

Logs completion

Args:

conf (TYPE): Description

\*outputs: Description

"""

elapsed = math.floor(time.time() - conf.TIME)

if conf['LOG'] != False and conf['MAKE'] != "archive":

click.secho(f"Task completed in {elapsed} seconds.", bold=True, fg='green')

def log(msg, fg="", bold=False, italic=False, \*conf):

if isinstance(conf, pd.core.series.Series):

try:

if conf['LOG']:

click.secho(msg, fg=fg, bold=bold, italic=italic)

except:

pass

else:

click.secho(msg, fg=fg, bold=bold, italic=italic)

# \_\_main\_\_.py

# main 77

# sam robson

import warnings

from alacorder import alac

import os

import sys

import math

import click

import pandas as pd

import time

import selenium

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import Select

from selenium.webdriver.chrome.options import Options

warnings.filterwarnings('ignore')

pd.set\_option("mode.chained\_assignment", None)

pd.set\_option("display.notebook\_repr\_html", True)

pd.set\_option('display.expand\_frame\_repr', True)

pd.set\_option('display.max\_rows', 100)

## COMMAND LINE INTERFACE

@click.group()

@click.version\_option("77.6.8", package\_name="alacorder")

def cli():

"""

ALACORDER beta 77.6

Alacorder retrieves case detail PDFs from Alacourt.com and processes them into text archives and data tables suitable for research purposes.

"""

pass

@cli.command(help="Export data tables from archive or directory")

@click.option('--input-path', '-in', required=True, type=click.Path(), prompt=alac.title(),

help="Path to input archive or PDF directory", show\_choices=False)

@click.option('--output-path', '-out', required=True, type=click.Path(), prompt=alac.both(), help="Path to output table (.xls, .xlsx, .csv, .json, .dta)")

@click.option('--table', '-t', help="Table export choice (cases, fees, charges, disposition, filing, or all)")

@click.option('--count', '-c', default=0, help='Total cases to pull from input', show\_default=False)

@click.option('--compress','-z', default=False, is\_flag=True,

help="Compress exported file (Excel files not supported)")

@click.option('--overwrite', '-o', default=False, help="Overwrite existing files at output path", is\_flag=True,show\_default=False)

@click.option('--no-prompt','-s', default=False, is\_flag=True, help="Skip user input / confirmation prompts")

@click.option('--no-batch','-b', default=True, is\_flag=True, help="Process all inputs as one batch")

@click.option('--no-log','-q','log', default=False, is\_flag=True, help="Don't print logs or progress to console")

@click.option('--no-write', default=False, is\_flag=True, help="Do not export to output path", hidden=True)

@click.option('--debug','-d', default=False, is\_flag=True, help="Print extensive logs to console for developers")

@click.version\_option(package\_name='alacorder', prog\_name='ALACORDER', message='%(prog)s beta %(version)s')

def table(input\_path, output\_path, count, table, overwrite, log, no\_write, no\_prompt, debug, no\_batch, compress): # dropped dedupe, archive

ogtable = table

archive = False

show\_options\_menu = False

log = not log

show\_options\_menu = True if no\_prompt == False and overwrite == False and log == True and no\_write == False and no\_prompt == False and debug == False and no\_batch == False and compress == False else False

# suppress tracebacks unless debug

if not debug:

sys.tracebacklimit = 0

warnings.filterwarnings('ignore')

else:

sys.tracebacklimit = 10

# inputs - configure and log

inputs = alac.setinputs(input\_path)

if debug:

click.echo(inputs)

if log:

click.secho(inputs.ECHO, fg='yellow', italic=True)

if not inputs.GOOD:

raise Exception("Invalid input path!")

# outputs - configure and log

outputs = alac.setoutputs(output\_path,archive=False)

if debug:

click.echo(outputs)

if log:

click.secho(outputs.ECHO, fg='yellow', italic=True)

if not outputs.GOOD:

raise Exception("Invalid output path!")

if outputs.OUTPUT\_EXT != ".xlsx" and outputs.OUTPUT\_EXT != ".xls" and outputs.OUTPUT\_EXT != ".dta" and outputs.OUTPUT\_EXT != ".json" and outputs.OUTPUT\_EXT != ".csv" and outputs.OUTPUT\_EXT != ".zip" and outputs.OUTPUT\_EXT != ".pkl" and outputs.OUTPUT\_EXT != ".xz" and outputs.OUTPUT\_EXT != ".parquet":

raise Exception("Bad format!")

# prompt overwrite

if outputs.EXISTING\_FILE and not overwrite:

if no\_prompt:

raise Exception("Existing file at output path! Repeat with flag --overwrite to replace file.")

else:

if click.confirm(click.style("Existing file at output path will be written over! Continue?",fg='bright\_yellow',bold=True)):

pass

else:

raise Exception("Existing file at output path!")

# prompt table

if outputs.MAKE == "multiexport" and table != "cases" and table != "fees" and table != "charges" and table != "disposition" and table != "filing":

table = "all"

if outputs.MAKE == "singletable" and table != "cases" and table != "fees" and table != "charges" and table != "disposition" and table != "filing":

if no\_prompt:

raise Exception("Invalid/missing table selection!")

else:

pick = click.prompt(alac.pick\_table\_only()) # add str

if pick == "B" or pick == "cases":

table = "cases"

elif pick == "C" or pick == "fees":

table = "fees"

elif pick == "D" or pick == "charges":

table = "charges"

elif pick == "E" or pick == "disposition":

table = "disposition"

elif pick == "F" or pick == "filing":

table = "filing"

else:

click.secho("Invalid table selection!", fg='red', bold=True)

# finalize config

cf = alac.set(inputs, outputs, count=count, table=table, overwrite=overwrite, log=log, no\_write=no\_write, no\_prompt=no\_prompt, no\_batch=no\_batch, debug=debug, compress=compress)

if cf.MAKE == "multiexport" and cf.TABLE == "all":

o = alac.cases(cf)

if cf.TABLE == "fees":

o = alac.fees(cf)

if cf.TABLE == "charges" or cf.TABLE == "disposition" or cf.TABLE == "filing":

o = alac.charges(cf)

if cf.TABLE == "cases":

o = alac.cases(cf)

@cli.command(help="Create full text archive from case PDFs")

@click.option('--input-path', '-in', required=True, type=click.Path(), prompt=alac.title(), help="Path to input archive or PDF directory", show\_choices=False)

@click.option('--output-path', '-out', required=True, type=click.Path(), prompt="alac.just\_archive()", help="Path to archive (.pkl.xz, .json.zip, .csv.zip, .parquet)")

@click.option('--count', '-c', default=0, help='Total cases to pull from input', show\_default=False)

@click.option('--dedupe / --ignore','dedupe', default=True, is\_flag=True, help="Remove duplicate cases from archive outputs")

@click.option('--compress','-z', default=False, is\_flag=True,

help="Compress exported file (archives compress with or without flag)")

@click.option('--overwrite', '-o', default=False, help="Overwrite existing files at output path", is\_flag=True,show\_default=False)

@click.option('--no-log','-q','log', default=False, is\_flag=True, help="Don't print logs or progress to console")

@click.option('--no-write','-n', default=False, is\_flag=True, help="Do not export to output path", hidden=True)

@click.option('--no-prompt', default=False, is\_flag=True, help="Skip user input / confirmation prompts")

@click.option('--debug','-d', default=False, is\_flag=True, help="Print extensive logs to console for developers")

@click.option('--no-batch','-b', default=True, is\_flag=True, help="Process all inputs as one batch")

@click.version\_option(package\_name='alacorder', prog\_name='ALACORDER', message='%(prog)s beta %(version)s')

def archive(input\_path, output\_path, count, overwrite, dedupe, log, no\_write, no\_batch, no\_prompt, debug, compress):

# show\_options\_menu = False

table = ""

archive = True

log = not log

# suppress tracebacks unless debug

if not debug:

sys.tracebacklimit = 0

warnings.filterwarnings('ignore')

else:

sys.tracebacklimit = 10

# inputs - configure and log

inputs = alac.setinputs(input\_path)

if debug:

click.echo(inputs)

if log:

click.secho(inputs.ECHO, fg='yellow', italic=True)

if not inputs.GOOD:

raise Exception("Invalid input path!")

# outputs - configure and log

outputs = alac.setoutputs(output\_path,archive=True)

if debug:

click.echo(outputs)

if log:

click.secho(outputs.ECHO, fg='yellow', italic=True)

if not outputs.GOOD:

raise Exception("Invalid output path!")

if outputs.OUTPUT\_EXT != ".xlsx" and outputs.OUTPUT\_EXT != ".xls" and outputs.OUTPUT\_EXT != ".dta" and outputs.OUTPUT\_EXT != ".json" and outputs.OUTPUT\_EXT != ".csv" and outputs.OUTPUT\_EXT != ".zip" and outputs.OUTPUT\_EXT != ".pkl" and outputs.OUTPUT\_EXT != ".xz" and outputs.OUTPUT\_EXT != ".parquet":

raise Exception("Bad format!")

# prompt overwrite

if outputs.EXISTING\_FILE and not overwrite:

if no\_prompt:

raise Exception("Existing file at output path! Repeat with flag --overwrite to replace file.")

else:

if click.confirm("Existing file at output path will be written over! Continue?"):

pass

else:

raise Exception("Existing file at output path!")

cf = alac.set(inputs, outputs, count=count, table="", overwrite=overwrite, log=log, dedupe=dedupe, no\_write=no\_write, no\_prompt=no\_prompt, no\_batch=no\_batch, debug=debug, compress=compress)

if debug:

click.echo(cf)

o = alac.archive(cf)

# fetch

@cli.command(help="Fetch cases from Alacourt.com with input query spreadsheet headers NAME, PARTY\_TYPE, SSN, DOB, COUNTY, DIVISION, CASE\_YEAR, and FILED\_BEFORE.")

@click.option("--input-path", "-in", "listpath", required=True, prompt="Path to query table", help="Path to query table/spreadsheet (.xls, .xlsx, .csv, .json)", type=click.Path())

@click.option("--output-path", "-out", "path", required=True, prompt="PDF download path", type=click.Path(), help="Desired PDF output directory")

@click.option("--customer-id", "-c","cID", required=True, prompt="Alacourt Customer ID", help="Customer ID on Alacourt.com")

@click.option("--user-id", "-u","uID", required=True, prompt="Alacourt User ID", help="User ID on Alacourt.com")

@click.option("--password", "-p","pwd", required=True, prompt="Alacourt Password", help="Password on Alacourt.com", hide\_input=True)

@click.option("--max", "-max","qmax", required=False, type=int, help="Maximum queries to conduct on Alacourt.com",default=0)

@click.option("--skip", "-skip", "qskip", required=False, type=int, help="Skip entries at top of query file",default=0)

@click.option("--speed", default=1, type=float, help="Speed multiplier")

@click.option("--no-log","-nl", is\_flag=True, default=False, help="Do not print logs to console")

@click.option("--no-update","-w", is\_flag=True, default=False, help="Do not update query template after completion")

@click.option("--ignore-complete","-g", is\_flag=True, default=False, help="Ignore initial completion status in query template")

@click.option("--debug","-d", is\_flag=True, default=False, help="Print detailed runtime information to console")

def fetch(listpath, path, cID, uID, pwd, qmax, qskip, speed, no\_log, no\_update, ignore\_complete, debug):

"""

Use headers NAME, PARTY\_TYPE, SSN, DOB, COUNTY, DIVISION, CASE\_YEAR, and FILED\_BEFORE in an Excel spreadsheet to submit a list of queries for Alacorder to fetch.

USE WITH CHROME (TESTED ON MACOS)

KEEP YOUR COMPUTER POWERED ON AND CONNECTED TO THE INTERNET.

Args:

listpath: (path-like obj) Query template path / input path

path: (path-like obj) Path to output/downloads directory

cID (str): Alacourt.com Customer ID

uID (str): Alacourt.com User ID

pwd (str): Alacourt.com Password

qmax (int, optional): Max queries to pull from inputs

qskip (int, optional): Skip top n queries in inputs

speed (int, optional): Fetch rate multiplier

no\_log (bool, optional): Do not print logs to console

no\_update (bool, optional): Do not update input query file with completion status

debug (bool, optional): Print detailed logs to console

Returns:

[driver, query\_out, query\_writer]:

driver[0]: Google Chrome WebDriver() object

query\_out[1]: (pd.Series) Fetchr queue

query\_writer[2]: (pd.DataFrame) Updated input query file

"""

if debug:

sys.tracebacklimit = 10

rq = alac.readPartySearchQuery(listpath, qmax, qskip, no\_log)

query = pd.DataFrame(rq[0]) # for fetchr - only search columns

query\_writer = pd.DataFrame(rq[1]) # original sheet for write completion

incomplete = query.RETRIEVED\_ON.map(lambda x: True if x == "" else False)

query = query[incomplete]

options = webdriver.ChromeOptions()

options.add\_experimental\_option('prefs', {

"download.default\_directory": path, #Change default directory for downloads

"download.prompt\_for\_download": False, #To auto download the file

"download.directory\_upgrade": True,

"plugins.always\_open\_pdf\_externally": True #It will not show PDF directly in chrome

})

driver = webdriver.Chrome(options=options)

# start browser session, auth

if not no\_log:

click.secho("Starting browser... Do not close while in progress!",fg='bright\_yellow',bold=True)

alac.login(driver, cID, uID, pwd, speed)

if not no\_log:

click.secho("Authentication successful. Fetching cases via party search...",fg='bright\_green',bold=True)

for i, n in enumerate(query.index):

if debug:

click.secho(driver.current\_url)

if driver.current\_url == "https://v2.alacourt.com/frmlogin.aspx":

alac.login(driver, cID, uID, pwd, speed, no\_log)

driver.implicitly\_wait(4/speed)

results = alac.party\_search(driver, name=query.NAME[n], party\_type=query.PARTY\_TYPE[n], ssn=query.SSN[n], dob=query.DOB[n], county=query.COUNTY[n], division=query.DIVISION[n], case\_year=query.CASE\_YEAR[n], filed\_before=query.FILED\_BEFORE[n], filed\_after=query.FILED\_AFTER[n], speed=speed, no\_log=no\_log, debug=debug)

driver.implicitly\_wait(4/speed)

if len(results) == 0:

query\_writer['RETRIEVED\_ON'][n] = str(math.floor(time.time()))

query\_writer['CASES\_FOUND'][n] = "0"

if not no\_log:

click.secho(f"Found no results for query: {query.NAME[n]}")

continue

with click.progressbar(results, show\_eta=False, label=f"#{n}: {query.NAME[n]}") as bar:

for url in bar:

alac.downloadPDF(driver, url)

driver.implicitly\_wait(0.5/speed)

time.sleep(2/speed)

if not no\_update:

query\_writer['RETRIEVED\_ON'][n] = str(math.floor(time.time()))

query\_writer['CASES\_FOUND'][n] = str(len(results))

query\_writer.to\_excel(listpath,sheet\_name="PartySearchQuery",index=False)

if \_\_name\_\_ == "\_\_main\_\_":

cli()

# \_\_init\_\_.py

# jalapeno.py

## SETUP: Install all dependencies, then run this code block to load Alacorder in the notebook interface.

import IPython.display

import ipywidgets, itables

from itables import init\_notebook\_mode

init\_notebook\_mode(all\_interactive=True)

load\_out = ipywidgets.Output()

with load\_out:

display(IPython.display.Markdown("This notebook can be used to interface with Alacorder directly. Once Alacorder is installed, launch Jupyter Notebook with the command line prompt `python3 -m jupyter notebook` or `python -m jupyter notebook`, then open `ALACORDER.ipynb` in the browser window that opens to begin. If your Python kernel is active and the notebook is trusted, Alacorder will load in this notebook automatically. Alacorder can run on most devices. If your device can run Python 3.7 or later, it can run Alacorder. The `fetch` tool for PDF retrieval requires an up-to-date installation of Google Chrome. Learn more at [PyPI](https://pypi.org/project/alacorder/) or [GitHub](https://github.com/sbrobson959/alacorder). \*\*Click `Run > Run All Cells` to start notebook interface.\*\* If installation fails, scroll to bottom of notebook and run frozen `%pip` cell (may have to \"unfreeze\" cell using toolbar), then restart kernel (`Kernel > Restart & Run All`)"))

import os, sys, warnings

import pandas as pd

from alacorder import alac

warnings.filterwarnings('ignore')

fetch\_launcher = ipywidgets.Output() # frame: user input for fetch

fetch\_logs = ipywidgets.Output() # frame: logs, progress bars for fetch

fetch\_table = ipywidgets.Output() # frame: logs, table for fetch

archive\_launcher = ipywidgets.Output() # frame: user input for archive

archive\_logs = ipywidgets.Output() # frame: logs, progress bars for archive

archive\_table = ipywidgets.Output() # frame: logs, progress bars for archive

multitable\_launcher = ipywidgets.Output() # frame: user input for multitable

multitable\_logs = ipywidgets.Output() # frame: logs, progress bars for multitable

multitable\_table = ipywidgets.Output() # frame: logs, table for multitable

singletable\_launcher = ipywidgets.Output() # frame: user input for singletable

singletable\_logs = ipywidgets.Output() # frame: logs, progress bars for singletable

singletable\_table = ipywidgets.Output() # frame: logs, progress bars for singletable

debug\_console = ipywidgets.Output() # feed debug logs here, pop 'er at the bottom

query\_path = ipywidgets.Text(description="Input Path:", layout=ipywidgets.Layout(width='90%'), tooltip="Path to query template spreadsheet (list of names, etc. to search)")

output\_dir = ipywidgets.Text(description="Output Path:", layout=ipywidgets.Layout(width='90%'))

cID = ipywidgets.Text(description="Customer ID: ")

uID = ipywidgets.Text(description="User ID: ")

pwd = ipywidgets.Password(description="Password: ")

btn = ipywidgets.Button(description="Login")

speed = ipywidgets.FloatText(description="Speed", value=1, min=0.1, max=3, step=0.1, layout=ipywidgets.Layout(width='40%'))

qskip = ipywidgets.IntText(description="Skip rows", value=0, min=0, step=1, layout=ipywidgets.Layout(width='40%'))

qmax = ipywidgets.IntText(description="Max rows", value=0, min=0, step=1, layout=ipywidgets.Layout(width='40%'))

fetch\_opts = ipywidgets.HBox([speed, qskip, qmax], layout=ipywidgets.Layout(width='95%'))

def startFetch(\*args):

global clicked, query\_path, output\_dir, cID, uID, pwd, btn, speed, qskip, qmax, fetch\_launcher, fetch\_logs, fetch\_table

if os.path.isdir(output\_dir.value) and os.path.isfile(query\_path.value) and cID.value != "" and uID.value != "" and pwd.value != "":

if qskip.value > 0 and qmax.value > 0:

return alac.fetch(query\_path.value,output\_dir.value,cID=cID.value, uID=uID.value, pwd=pwd.value, speed=speed.value, qskip=qskip.value, qmax=qmax.value)

elif qskip.value > 0:

return alac.fetch(query\_path.value,output\_dir.value,cID=cID.value, uID=uID.value, pwd=pwd.value, speed=speed.value, qskip=qskip.value)

elif qmax.value > 0:

return alac.fetch(query\_path.value,output\_dir.value,cID=cID.value, uID=uID.value, pwd=pwd.value, speed=speed.value, qmax=qmax.value)

else:

return alac.fetch(query\_path.value,output\_dir.value,cID=cID.value, uID=uID.value, pwd=pwd.value, speed=speed.value)

else:

print("Ensure all fields are correctly filled, then try again.")

return None

oQueryTable = ipywidgets.Output()

oQueryTable\_hidden = True

def viewQueryTable(\*args):

global oQueryTable

global oQueryTable\_hidden

if oQueryTable\_hidden:

inputstab = """<style> table {float:left} </style> <table> <thead> <tr> <th>Field</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><code>NAME</code></td> <td>Last Name First</td> </tr> <tr> <td><code>PARTY\_TYPE</code></td> <td>Social Security Number (Optional)</td> </tr> <tr> <td><code>SSN</code></td> <td>Applies to civil cases only (Optional)</td> </tr> <tr> <td><code>DOB</code></td> <td>Date of Birth (M/DD/YYYY)</td> </tr> <tr> <td><code>COUNTY</code></td> <td>Select a county if not statewide</td> </tr> <tr> <td><code>DIVISION</code></td> <td>Select a division if not all divisions.</td> </tr> <tr> <td><code>CASE\_YEAR</code></td> <td>Four digit case year to limit results</td> </tr> <tr> <td><code>FILED\_BEFORE</code></td> <td>Do not include cases filed after M/DD/YYYY</td> </tr> <tr> <td><code>FILED\_AFTER</code></td> <td>Do not include cases filed after M/DD/YYYY</td> </tr> </tbody> </table> """

oQueryTable\_hidden = False

with oQueryTable:

display(IPython.display.HTML(inputstab))

else:

oQueryTable.clear\_output()

oQueryTable\_hidden = True

pass

alacfetchhead = """<a id="fetch"></a>

<h2 id="collect-case-pdfs-in-bulk-from-alacourt-com-from-a-list-of-names-or-search-parameters-">Collect case PDFs in bulk from Alacourt.com from a list of names or search parameters.</h2>

<p><strong>Use column headers <code>NAME</code>, <code>PARTY\_TYPE</code>, <code>SSN</code>, <code>DOB</code>, <code>COUNTY</code>, <code>DIVISION</code>, <code>CASE\_YEAR</code>, and/or <code>FILED\_BEFORE</code> in an Excel spreadsheet to submit a list of queries for Alacorder to scrape. Each column corresponds to a search field in Party Search. Missing columns and entries will be left empty, i.e. if only the <code>NAME</code>&#39;s and <code>CASE\_YEAR</code>&#39;s are relevant to the search, a file with two columns will work.</strong></p>

"""

btn2 = ipywidgets.Button(description="Read more")

def showFetch(\*args):

global fetch\_launcher, fetch\_opts, query\_path, output\_dir, cID, uID, pwd, btn, btn2

with fetch\_launcher:

display(IPython.display.HTML(alacfetchhead))

display(btn2, oQueryTable)

btn2.on\_click(viewQueryTable)

display(fetch\_opts)

display(query\_path)

display(output\_dir)

display(cID)

display(uID)

display(pwd)

btn.on\_click(startFetch)

display(btn)

arc\_title = IPython.display.HTML("""<a id="arc"></a>

<h2 id="case-text-archives-require-a-fraction-of-the-storage-capacity-and-processing-time-used-to-process-pdf-directories-before-exporting-your-data-to-tables-create-an-archive-with-supported-file-extensions-pkl-xz-json-zip-parquet-and-csv-zip-">Case text archives require a fraction of the storage capacity and processing time used to process PDF directories. Before exporting your data to tables, create an archive with supported file extensions <code>.pkl.xz</code>, <code>.json(.zip)</code>, <code>.parquet</code> and <code>.csv(.zip)</code>.</h2>

<p><strong>Once archived, use your case text archive as an input for multitable or single table export.</strong></p>

""")

arc\_inpath = ipywidgets.Text(description="Input Path",

layout=ipywidgets.Layout(width='90%'),

tooltip="Path to PDF directory")

arc\_outpath = ipywidgets.Text(description="Output Path", layout=ipywidgets.Layout(width='90%'))

arc\_overwrite = ipywidgets.Checkbox(description="Don't allow overwrite")

arc\_dedupe = ipywidgets.Checkbox(description="Remove duplicates")

arc\_compress = ipywidgets.Checkbox(description="ZIP export (must be .json, .csv)")

arc\_btn = ipywidgets.Button(description="Start archiving")

arc\_count = ipywidgets.IntText(description="Max count", value=0, min=0, step=1, layout=ipywidgets.Layout(width='40%'))

def startArchive(\*args):

global arc\_title, arc\_inpath, arc\_outpath, arc\_count, arc\_overwrite, arc\_btn

arcov = not arc\_overwrite.value

if arc\_inpath.value.strip() != "" and arc\_outpath.value.strip() != "":

a = alac.setinit(arc\_inpath.value, arc\_outpath.value, archive=True, overwrite=arcov, no\_prompt=arcov, no\_batch=True, count=arc\_count.value)

return a

else:

return None

def showArchive():

global arc\_title, arc\_inpath, arc\_outpath, arc\_count, arc\_overwrite, arc\_dedupe, arc\_compress, arc\_btn

with archive\_launcher:

display(arc\_title)

display(arc\_count)

display(arc\_inpath)

display(arc\_outpath)

arc\_chk = ipywidgets.HBox([arc\_overwrite, arc\_dedupe, arc\_compress])

display(arc\_chk)

display(arc\_btn)

arc\_btn.on\_click(startArchive)

return None

mtab\_title = IPython.display.HTML("""<a id="mtab"></a>

<h2 id="multitable-export-processes-case-detail-pdfs-and-case-text-archives-into-data-tables-suitable-for-research-purposes-export-an-excel-spreadsheet-with-detailed-cases-information-cases-fee-sheets-fees-and-charges-information-charges-disposition-filing-">Multitable export processes case detail PDFs and case text archives into data tables suitable for research purposes. Export an Excel spreadsheet with detailed cases information (<code>cases</code>), fee sheets (<code>fees</code>), and charges information (<code>charges</code>, <code>disposition</code>, <code>filing</code>).</h2>

<p><strong><em>Note: It is recommended that you create a case text archive from your target PDF directory before exporting tables. Case text archives can be processed into tables at a much faster rate and require far less storage.</em></strong></p>""")

mtab\_inpath = ipywidgets.Text(description="Input Path",

layout=ipywidgets.Layout(width='95%'),

tooltip="Path to input directory or archive")

mtab\_outpath = ipywidgets.Text(description="Output Path", layout=ipywidgets.Layout(width='95%'))

mtab\_count = ipywidgets.IntText(

value=0,

min=0,

max=10000,

step=1,

description='Max count:',

orientation='horizontal',

readout=True,

readout\_format='d'

)

mtab\_overwrite = ipywidgets.Checkbox(description="Don't allow overwrite")

mtab\_btn = ipywidgets.Button(description="Start export")

mtab\_obox = ipywidgets.HBox([mtab\_count, mtab\_overwrite])

def startMulti(\*args):

global mtab\_title, mtab\_inpath, mtab\_outpath, mtab\_count, mtab\_overwrite, mtab\_btn

mtov = not mtab\_overwrite.value

if mtab\_inpath.value.strip() != "":

cf = alac.setpaths(mtab\_inpath.value, mtab\_outpath.value, overwrite=mtov, no\_prompt=mtov, no\_batch=True)

a = alac.init(cf)

with multitable\_table:

display(a[0],a[1],a[2])

return a

else:

return None

def showMulti():

with multitable\_launcher:

display(mtab\_title)

display(mtab\_obox)

display(mtab\_inpath)

display(mtab\_outpath)

mtab\_btn.on\_click(startMulti)

display(mtab\_btn)

return None

stab\_title = IPython.display.HTML("""<a id="stab"></a>

<h2 id="export-charges-including-disposition-only-and-filing-only-cases-or-fees-table-only-single-table-export-enables-file-types-without-support-for-multiple-sheets-this-mode-allows-export-to-csv-json-dta-xls-xlsx-pkl-and-parquet-files-">Export <code>charges</code> (including <code>disposition</code> only and <code>filing</code> only), <code>cases</code>, or <code>fees</code> table only. Single table export enables file types without support for multiple sheets. This mode allows export to <code>.csv</code>, <code>.json</code>, <code>.dta</code>, <code>.xls</code>, <code>.xlsx</code>, <code>.pkl</code>, and <code>.parquet</code> files.</h2>

<p><strong>Once archived, use your case text archive as an input for multitable or single table export.</strong></p>""")

stab\_inpath = ipywidgets.Text(description="Input Path",

layout=ipywidgets.Layout(width='90%'),

tooltip="Path to input directory or archive")

stab\_table = ipywidgets.Dropdown(options=['cases', 'charges', 'disposition', 'filing', 'fees'],description='Table:',layout=ipywidgets.Layout(orientation='horizontal'))

stab\_outpath = ipywidgets.Text(description="Output Path", layout=ipywidgets.Layout(width='90%'))

stab\_count = ipywidgets.IntText(

value=0,

min=0,

max=10000,

step=1,

description='Max count:',

orientation='horizontal',

readout=True,

readout\_format='d'

)

stab\_overwrite = ipywidgets.Checkbox(description="Don't allow overwrite")

stab\_btn = ipywidgets.Button(description="Start export")

stab\_obox = ipywidgets.HBox([stab\_count, stab\_overwrite])

def startSingle(\*args):

global stab\_title, stab\_inpath, stab\_outpath, stab\_count, stab\_overwrite, stab\_btn

stov = not stab\_overwrite.value

if stab\_inpath.value.strip() != "":

a = alac.setinit(stab\_inpath.value, stab\_outpath.value, table=stab\_table.value, overwrite=stov, no\_prompt=stov, no\_batch=True, count=stab\_count.value)

with singletable\_table:

display(a)

return a

else:

return None

def showSingle():

global singletable\_launcher

with singletable\_launcher:

display(stab\_title)

display(stab\_obox)

display(stab\_inpath)

display(stab\_outpath)

display(stab\_table)

stab\_btn.on\_click(startSingle)

display(stab\_btn)

return None

with load\_out:

display(IPython.display.Markdown("\*\*Notebook configuration succeeded. Run `%pip` cell at bottom of document, restart kernel and run all cells if `ALACORDER` does not function properly.\*\*"))

# pyproject.toml

[build-system]

requires = ["setuptools"]

build-backend = "setuptools.build\_meta"

[project]

name = "alacorder"

version = "77.6.8"

authors = [

{ name="Sam Robson", email="sbrobson@crimson.ua.edu" },

]

description = "Alacorder collects and processes case detail PDFs into data tables suitable for research purposes. Alacorder also generates compressed text archives from the source PDFs to speed future data collection from the same set of cases. Google Chrome required for direct access to case PDFs via query template (see /templates on GitHub)."

readme = "README.md"

requires-python = ">=3.9"

classifiers = [

"Programming Language :: Python :: 3",

"License :: OSI Approved :: MIT License",

"Operating System :: OS Independent",

]

dependencies = ["xlrd","openpyxl","pandas==2.0.0rc0","PyPDF2","numpy","xlwt","numexpr","openpyxl","bottleneck","xarray","click","pyarrow","selenium","tqdm"]

[project.optional-dependencies]

all = ["xlsxwriter>=3.0.1", "jupyter", "ipython", "jupyter\_nbextensions\_configurator", "tabulate", "itables", "ipywidgets"]

# README.md

[![Binder](https://mybinder.org/badge\_logo.svg)](https://mybinder.org/v2/gh/sbrobson959/alacorder/main?labpath=%2Fsrc%2Falacorder%2FALACORDER.ipynb)

```

\_\_\_ \_\_ \_\_

/ | / /\_\_\_ \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_/ /\_\_ \_\_\_\_\_

/ /| | / / \_\_ `/ \_\_\_/ \_\_ \/ \_\_\_/ \_\_ / \_ \/ \_\_\_/

/ \_\_\_ |/ / /\_/ / /\_\_/ /\_/ / / / /\_/ / \_\_/ /

/\_/ |\_/\_/\\_\_,\_/\\_\_\_/\\_\_\_\_/\_/ \\_\_,\_/\\_\_\_/\_/

ALACORDER beta 77

```

# \*\*Getting Started with Alacorder\*\*

### Alacorder collects and processes case detail PDFs into data tables suitable for research purposes. Alacorder also generates compressed text archives from the source PDFs to speed future data collection from the same set of cases.

<sup>[GitHub](https://github.com/sbrobson959/alacorder) | [PyPI](https://pypi.org/project/alacorder/) | [Report an issue](mailto:sbrobson@crimson.ua.edu)

</sup>

```

Usage: python -m alacorder COMMAND [OPTIONS]...

Commands:

archive Create full text archive from case PDFs

fetch Search Alacourt.com with query template (see /templates)

table Export data tables from archive or directory

Options:

-in, --input-path PATH Path to input archive or PDF directory [required]

-out, --output-path PATH Path to output table (.xls, .xlsx, .csv, .json,

.dta) or archive (.pkl.xz, .json.zip, .parquet)

[required]

-t, --table TEXT Table export choice (cases, fees, charges,

disposition, filing, or all)

-c, --count INTEGER Total cases to pull from input

--dedupe / --ignore Remove duplicate cases from archive outputs

-z, --compress Compress exported file (archives compress with or

without flag)

-o, --overwrite Overwrite existing files at output path

-q, --no-log Don't print logs or progress to console

-p, --no-prompt Skip user input / confirmation prompts

-b, --no-batch Process all inputs as one batch

```

## \*\*Installation\*\*

\*\*Alacorder can run on most devices. If your device can run Python 3.9 or later, it can run Alacorder.\*\*

\* To install on Windows and Mac, open Command Prompt (Terminal) and enter `pip install alacorder` or `pip3 install alacorder`.

\* On Mac, open the Terminal and enter `pip install alacorder` or `pip3 install alacorder`.

\* Install [Anaconda Distribution](https://www.anaconda.com/products/distribution) to install Alacorder if the above methods do not work, or if you would like to open an interactive browser notebook equipped with Alacorder on your desktop.

\* After installation, create a virtual environment, open a terminal, and then repeat these instructions. If your copy of Alacorder is corrupted, use `pip uninstall alacorder` or `pip3 uninstall alacorder` and then reinstall it. There may be a newer version available.

```python

pip install alacorder

```

## \*\*Using the command line interface\*\*

#### \*\*Once you have a Python environment up and running, you can launch the guided interface in two ways:\*\*

1. \*Utilize the `alacorder` module in your command line:\* Use the command line tool `python -m alacorder`, or `python3 -m alacorder`. If the guided version is launched instead of the command line tool, update your installation with `pip install --upgrade alacorder`.

2. \*Conduct custom searches with `alac`:\* Use the import statement `from alacorder import alac` to use the Alacorder APIs to collect custom data from case detail PDFs. See how you can make `alacorder` work for you in the code snippets below.

#### \*\*Alacorder can be used without writing any code, and exports to common formats like Excel (`.xls`, `.xlsx`), Stata (`.dta`), CSV (`.csv`), and JSON (`.json`).\*\*

\* Alacorder compresses case text into `pickle` archives (`.pkl.xz`) to save storage and processing time. If you need to unpack a `pickle` archive without importing `alac`, use a `.xz` compression tool, then read the `pickle` into Python with the `pandas` method [`pd.read\_pickle()`](https://pandas.pydata.org/docs/reference/api/pandas.read\_pickle.html).

# \*\*Special Queries with `alac`\*\*

```python

from alacorder import alac

```

### \*\*For more advanced queries, the `alac` module can extract fields and tables from case records with just a few lines of code.\*\*

\* Call `alac.setinputs("/pdf/dir/")` and `alac.setoutputs("/to/table.xlsx")` to configure your input and output paths. Then call `alac.set(input\_conf, output\_conf, \*\*kwargs)` to complete the configuration process. Feed the output to any of the `alac.write...()` functions to start a task.

\* Call `alac.archive(config)` to export a full text archive. It's recommended that you create a full text archive (`.pkl.xz`) file before making tables from your data. Full text archives can be scanned faster than PDF directories and require less storage. Full text archives can be imported to Alacorder the same way as PDF directories.

\* Call `alac.tables(config)` to export detailed case information tables. If export type is `.xls` or `.xlsx`, the `cases`, `fees`, and `charges` tables will be exported.

\* Call `alac.charges(config)` to export `charges` table only.

\* Call `alac.fees(config)` to export `fees` table only.

\* Call `alac.caseinfo(config)` to export `cases` table only.

```python

import warnings

warnings.filterwarnings('ignore')

from alacorder import alac

pdf\_directory = "/Users/crimson/Desktop/Tutwiler/"

archive = "/Users/crimson/Desktop/Tutwiler.pkl.xz"

tables = "/Users/crimson/Desktop/Tutwiler.xlsx"

pdfconf = alac.setinputs(pdf\_directory)

arcconf = alac.setoutputs(archive)

# write archive to Tutwiler.pkl.xz

c = alac.set(pdfconf, arcconf)

alac.archive(c)

print("Full text archive complete. Now processing case information into tables at " + tables)

d = alac.setpaths(archive, tables) # runs setinputs(), setoutputs() and set() at once

alac.tables(d)

# write tables to Tutwiler.xlsx

alac.tables(tabconf)

```

## \*\*Custom Parsing with `alac.map()`\*\*

### If you need to conduct a custom search of case records, Alacorder has the tools you need to extract custom fields from case PDFs without any fuss. Try out `alac.map()` to search thousands of cases in seconds.

```python

from alacorder import alac

import re

archive = "/Users/crimson/Desktop/Tutwiler.pkl.xz"

tables = "/Users/crimson/Desktop/Tutwiler.xlsx"

def findName(text):

name = ""

if bool(re.search(r'(?a)(VS\.|V\.)(.+)(Case)\*', text, re.MULTILINE)) == True:

name = re.search(r'(?a)(VS\.|V\.)(.+)(Case)\*', text, re.MULTILINE).group(2).replace("Case Number:","").strip()

else:

if bool(re.search(r'(?:DOB)(.+)(?:Name)', text, re.MULTILINE)) == True:

name = re.search(r'(?:DOB)(.+)(?:Name)', text, re.MULTILINE).group(1).replace(":","").replace("Case Number:","").strip()

return name

c = alac.setpaths(archive, tables, count=2000) # set configuration

alac.map(c, findName, alac.getConvictions) # Name, Convictions table

```

| Method | Description |

| ------------- | ------ |

| `getPDFText(path) -> text` | Returns full text of case |

| `getCaseInfo(text) -> [case\_number, name, alias, date\_of\_birth, race, sex, address, phone]` | Returns basic case details |

| `getFeeSheet(text, cnum = '') -> [total\_amtdue, total\_balance, total\_d999, feecodes\_w\_bal, all\_fee\_codes, table\_string, feesheet: pd.DataFrame]` | Returns fee sheet and summary as `str` and `pd.DataFrame` |

| `getCharges(text, cnum = '') -> [convictions\_string, disposition\_charges, filing\_charges, cerv\_eligible\_convictions, pardon\_to\_vote\_convictions, permanently\_disqualifying\_convictions, conviction\_count, charge\_count, cerv\_charge\_count, pardontovote\_charge\_count, permanent\_dq\_charge\_count, cerv\_convictions\_count, pardontovote\_convictions\_count, charge\_codes, conviction\_codes, all\_charges\_string, charges: pd.DataFrame]` | Returns charges table and summary as `str`, `int`, and `pd.DataFrame` |

| `getCaseNumber(text) -> case\_number` | Returns case number

| `getName(text) -> name` | Returns name

| `getFeeTotals(text) -> [total\_row, tdue, tpaid, tbal, tdue]` | Return totals without parsing fee sheet

# \*\*Working with case data in Python\*\*

### Out of the box, Alacorder exports to `.xlsx`, `.xls`, `.csv`, `.json`, and `.dta`. But you can use `alac`, `pandas`, and other python libraries to create your own data collection workflows and design custom exports.

\*\*\*The snippet below prints the fee sheets from a directory of case PDFs as it reads them.\*\*\*

```python

from alacorder import alac

c = alac.setpaths("/Users/crimson/Desktop/Tutwiler/","/Users/crimson/Desktop/Tutwiler.xls")

for path in c['contents']:

text = alac.getPDFText(path)

cnum = alac.getCaseNumber(text)

charges\_outputs = alac.getCharges(text, cnum)

if len(charges\_outputs[0]) > 1:

print(charges\_outputs[0])

```

## Extending Alacorder with `pandas` and other tools

Alacorder runs on [`pandas`](https://pandas.pydata.org/docs/getting\_started/index.html#getting-started), a python library you can use to perform calculations, process text data, and make tables and charts. `pandas` can read from and write to all major data storage formats. It can connect to a wide variety of services to provide for easy export. When Alacorder table data is exported to `.pkl.xz`, it is stored as a `pd.DataFrame` and can be imported into other python [modules](https://www.anaconda.com/open-source) and scripts with `pd.read\_pickle()` like below:

```python

import pandas as pd

contents = pd.read\_pickle("/path/to/pkl")

```

If you would like to visualize data without exporting to Excel or another format, create a `jupyter notebook` and import a data visualization library like `matplotlib` to get started. The resources below can help you get started. [`jupyter`](https://docs.jupyter.org/en/latest/start/index.html) is a Python kernel you can use to create interactive notebooks for data analysis and other purposes. It can be installed using `pip install jupyter` or `pip3 install jupyter` and launched using `jupyter notebook`. Your device may already be equipped to view `.ipynb` notebooks.

## \*\*Resources\*\*

\* [`pandas` cheat sheet](https://pandas.pydata.org/Pandas\_Cheat\_Sheet.pdf)

\* [regex cheat sheet](https://www.rexegg.com/regex-quickstart.html)

\* [anaconda (tutorials on python data analysis)](https://www.anaconda.com/open-source)

\* [The Python Tutorial](https://docs.python.org/3/tutorial/)

\* [`jupyter` introduction](https://realpython.com/jupyter-notebook-introduction/)

-------------------------------------

© 2023 Sam Robson

# LICENSE

MIT License

Copyright (c) 2023 Sam Robson

Permission is hereby granted, free of charge, to any person obtaining a copy

of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights

to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

copies of the Software, and to permit persons to whom the Software is

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.