Sample Code: Curtis Von Gunten, PhD

**Contents**

[Analyzing Stratified Multistate Area Sampling Survey Data (SAS, Python) 2](#_Toc56962321)

[Utilizing Mixed-Effects Models to Assess Unconscious Influence of Images on Brain Recordings (SAS) 5](#_Toc56962322)

[Simulating Experiments to Examine Statistical Power Using Bootstrap Sampling (Python) 6](#_Toc56962323)

[Automating 30 Structural Equation Models to Examine Self-control and Health Behaviors (R, Mplus) 9](#_Toc56962324)

[PubMed Dashboard Application (Python, Heroku, Flask, HTML, CSS) 13](#_Toc56962325)

(Please see my GitHub for additional code: <https://github.com/Curt-Von-Gunten>)

# Analyzing Stratified Multistate Area Sampling Survey Data (SAS, Python)

This SAS code runs an adjusted logistic regression model on clustered, stratified survey data. The code exports the SAS results to HTML format. This allowed me to parse the code in Python using a web scraping module, to produce a manuscript table (see snippet below code; manuscript in press at *Journal of Studies on Alcohol and Drugs*).

SAS code:

ODS HTML PATH = "S:\IRB\Wu\VonGunten\_2018\NSDUH Projects\Multiple SUD and Treatment Project\Output\Pythong Parsing\Table 4" BODY ="Table 4\_TreatOdds\_Adjusted.html" STYLE = HTMLBlue;

**PROC** **SURVEYLOGISTIC** DATA = MultSUDTreat\_Comb NOMCAR;

STRATA VESTR; CLUSTER VEREP; WEIGHT NEWWTS; DOMAIN SubSample3;

CLASS AlcCategory DNICNSP Sex Age Race Education FamInc Work Insurance Mar COUTYP4 SurveyYear/ REF = FIRST PARAM = ref;

model TXYRRECVD2 (REF = '0') = AlcCategory DNICNSP Sex Age Race Education FamInc Work Insurance Mar COUTYP4 SurveyYear/ LINK = glogit DF=DESIGN ;

**RUN**;

ODS HTML CLOSE;

Parsing HTML SAS output and wrangling it into a publication-ready table.

Python Code:

from bs4 import BeautifulSoup as soup

import os

import numpy as np

import pandas as pd

from decimal import Decimal, ROUND\_HALF\_UP

path = "S:/IRB/Wu/VonGunten\_2018/NSDUH Projects/Multiple SUD and Treatment Project/Output/Pythong Parsing/Table 3"

os.chdir(path)

############################################################File 2###########################################

HtmlFile = open("Table 3\_PercTreat.html", 'r', encoding='utf-8')

content = HtmlFile.read()

HtmlFile.close()

page\_soup = soup(content, "html.parser")

AllCrossTables = page\_soup.find\_all(summary="Procedure Surveyfreq: CrossTabulation Table")

TableNumber = len(AllCrossTables)

DrugAmountDF = pd.DataFrame(columns = ("Row Percent", "Std Err of Row Percent"))

newListTreatPerc = []

newListTreatCI = []

for x in range(1, len(AllCrossTables), 2):

    temp = AllCrossTables[x]

    tempText = temp.find\_all('td', class\_="r data")

    tempValues = [pt.get\_text() for pt in tempText]

    tempDF = pd.DataFrame(np.array(tempValues).reshape(int(len(tempValues)/11),11),

                              columns = ("Frequency", "Weighted Frequency", "Std Err of Wgt Freq",

                                  "Percent", "Std Err of Percent", "Percent Lower CI", "Percent Upper  CI",

                                  "Row Percent", "Std Err of Row Percent", "Row Percent Lower CI", "Row Percent Upper CI"))

##############Treatment Percent##############

    tempPerElem = tempDF.iloc[4,7]

    tempPerElem = Decimal(str(tempPerElem)).quantize(Decimal('1.11'), rounding=ROUND\_HALF\_UP)

    tempPerStr = str(tempPerElem)

    newListTreatPerc.append(tempPerStr)

################Treatment CI################

    tempCIElem = tempDF.iloc[4,[9,10]]

    tempCIElem = [Decimal(str(CI)).quantize(Decimal('1.11'), rounding=ROUND\_HALF\_UP) for CI in tempCIElem]

    tempCIString = str('[' + str(tempCIElem[0]) + "-" + str(tempCIElem[1]) + "]")

    newListTreatCI .append(tempCIString)

############################################################Treatment Percent###########################################

Table3\_df = pd.DataFrame(columns=['Variables', 'n', 'Percentage', 'Percentage CI', 'Treatment Percentage', 'Treatment CI'])

Table3\_df['Variables'] = variableList

Table3\_df['n'] = newListCount

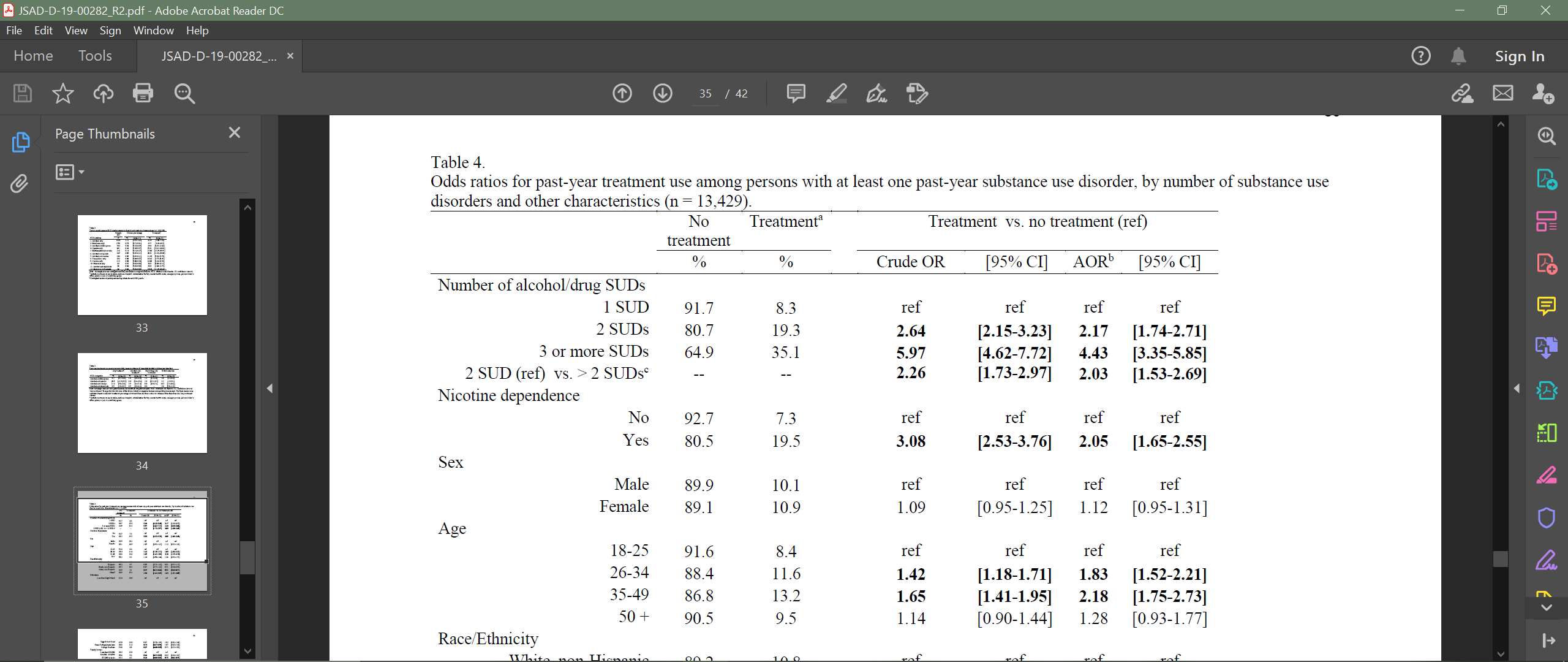
Table3\_df['Percentage'] = newListPerc

Table3\_df['Percentage CI'] = newListPercCI

Table3\_df['Treatment Percentage'] = newListTreatPerc

Table3\_df['Treatment CI'] = newListTreatCI

Table:



# Utilizing Mixed-Effects Models to Assess Unconscious Influence of Images on Brain Recordings (SAS)

This SAS code uses a mixed-effects model (hierarchical linear model) to test the influence of positive, negative, and neutral images on event-related potential waveforms (brain recordings). The results were used for this published paper: https://onlinelibrary.wiley.com/doi/abs/10.1111/psyp.12773

SAS code:

**proc** **mixed** data=BothBlocks\_TargetLocked;

class sub channel group BothBlocks\_byPrimes block coronal condition sagittal valence binary\_val hemisphere Ps\_Os;

model mean = BothBlocks\_byPrimes|block|valence;

random intercept / sub=sub; \*random intercept of subject;

random intercept / sub=sub(channel); \*random intercept of channel within subject;

lsmeans BothBlocks\_byPrimes;

lsmeans block;

lsmeans valence;

lsmeans BothBlocks\_byPrimes\*block\*valence/ tdiff pdiff e;

lsmestimate BothBlocks\_byPrimes\*block\*valence'NegSubs Neutrals difference' **0** **1** **0** **0** -**1** **0** **0** **0** **0** **0**;

lsmestimate BothBlocks\_byPrimes\*block\*valence'PosSubs Neutrals difference' **0** **0** **0** **0** **0** **0** **1** **0** -**1** **0**;

lsmestimate BothBlocks\_byPrimes\*block\*valence'NegSubs val minus neut difference across block comparison' **1** **1** **0** -**1** -**1** **0** **0** **0** **0** **0**;

lsmestimate BothBlocks\_byPrimes\*block\*valence'PosSubs val minus neut difference across block comparison' **0** **0** **0** **0** **0** **0** **1** **1** -**1** -**1**;

lsmestimate BothBlocks\_byPrimes\*block\*valence'Neut diffs comparison across condition' **0** **1** **0** **0** **1** **0** -**1** **0** -**1** **0**;

lsmestimate BothBlocks\_byPrimes\*block\*valence'NegSub/PosSub neut comparison first block' **0** **1** **0** **0** **0** **0** -**1** **0** **0** **0**;

lsmestimate BothBlocks\_byPrimes\*block\*valence'NegSub/PosSub neut comparison second block' **0** **0** **0** **0** **1** **0** **0** **0** -**1** **0**;

lsmestimate BothBlocks\_byPrimes\*block\*valence'NegSubs val minus neut difference across block comparison' **1** -**1** **0** -**1** **1** **0** **0** **0** **0** **0**;

lsmestimate BothBlocks\_byPrimes\*block\*valence'PosSubs val minus neut difference across block comparison' **0** **0** **0** **0** **0** **0** -**1** **1** **1** -**1**;

lsmestimate BothBlocks\_byPrimes\*block\*valence'NegSubs neg diff' **1** **0** **0** -**1** **0** **0** **0** **0** **0** **0**;

title 'Both Block TargetLocked';

**run**;

# Simulating Experiments to Examine Statistical Power Using Bootstrap Sampling (Python)

This is a tool written in Python that uses bootstrap sampling to examine statistical power and standardized effect size as a function of number of subjects, number of trials, effect magnitude (in original units), and design-type (between- vs. within-subjects). It is 329 lines of code; I provide a snippet. Here is the advance online publication: <https://www.sciencedirect.com/science/article/abs/pii/S0167876019304726>

Python code:

Here is a snippet of a call to the custom ‘PowerSimulation’ function.

###Anti###

###Between, diff=False###

path = "…\_2018/Simulation Project\_10-32/Data\_Code\_Output/Anti/Revision"

dataset = "Anti\_Ready\_Revision"

subjectNumber = [20,35,50,100,150]

trialNumber = [26,50,100,200]

effectMag = [.05,.075,.10,.125]

simulationNumber = 501

designType = 'between'

differenceScore = False

Result1 = pf.PowerSimulation(path,dataset,subjectNumber,trialNumber,effectMag,simulationNumber,designType,differenceScore)

Here is a snippet of the function (‘PowerSimulation’) itself.

# -\*- coding: utf-8 -\*-

import os

import pandas as pd

import numpy as np

from scipy import stats

def PowerSimulation(path,dataset,subjectNumber,trialNumber,effectMag,simulationNumber=50,designType='between',differenceScore=False,stopSignal=False):

    os.chdir(path)

    data = pd.read\_csv(str(dataset + '.txt'), sep="\t")

    datasubs = list(data.Subject.unique())

    powerResults = pd.DataFrame(columns=['Numb\_of\_Subs', 'Numb\_of\_Trials', 'Effect\_Magnitude', 'Effect\_Size', 'Power'])

    if stopSignal == True and designType == 'between':

        for e in subjectNumber:

            for f in trialNumber:

                for g in effectMag:

                    powercounter = 0

                    d = []

                    for h in range(1,simulationNumber):

                        SubLevDF1 = pd.DataFrame(columns=['SSRT'])

                        SubLevDF2 = pd.DataFrame(columns=['SSRT'])

                        SubList1 = list(np.random.choice(datasubs, int(e/2)))

                        for subject in SubList1:

                            temp = data[data.Subject == subject]

                            tempnotstop = temp[temp['TrialType'] == 'NotStop']

                            tempstop = temp[temp['TrialType'] == 'Stop']

                            TrialListNotStop = list(np.random.choice(list(tempnotstop.Trial), int(f) \* 3))

                            TrialListStop = list(np.random.choice(list(tempstop.Trial), int(f)))

                            tempnotstop.set\_index("Trial", inplace=True)

                            tempstop.set\_index("Trial", inplace=True)

                            tempnotstop = tempnotstop.loc[TrialListNotStop]

                            tempstop = tempstop.loc[TrialListStop]

                            tempnotstop.reset\_index(inplace=True)

                            tempstop.reset\_index(inplace=True)

                            RTNotStop = float(tempnotstop[["RT"]].mean())

                            SSDStop = float(tempstop[["SSD"]].mean())

                            SSRTList = [[RTNotStop - SSDStop]]

                            tempDF = pd.DataFrame(SSRTList,columns=['SSRT'])

                            SubLevDF1 = SubLevDF1.append(tempDF)

                        SubList2 = list(np.random.choice(datasubs, int(e/2)))

                        for subject in SubList2:

                            temp = data[data.Subject == subject]

                            tempnotstop = temp[temp['TrialType'] == 'NotStop']

                            tempstop = temp[temp['TrialType'] == 'Stop']

                            TrialListNotStop = list(np.random.choice(list(tempnotstop.Trial), int(f) \* 3))

                            TrialListStop = list(np.random.choice(list(tempstop.Trial), int(f)))

                            tempnotstop.set\_index("Trial", inplace=True)

                            tempstop.set\_index("Trial", inplace=True)

                            tempnotstop = tempnotstop.loc[TrialListNotStop]

                            tempstop = tempstop.loc[TrialListStop]

                            tempnotstop.reset\_index(inplace=True)

                            tempstop.reset\_index(inplace=True)

                            RTNotStop = float(tempnotstop[["RT"]].mean())

                            SSDStop = float(tempstop[["SSD"]].mean())

                            SSRTList = [[RTNotStop - SSDStop]]

                            tempDF = pd.DataFrame(SSRTList,columns=['SSRT'])

                        SubLevDF2 = SubLevDF2.append(tempDF)

                        SubLevDF2['SSRT'] = SubLevDF2['SSRT'].apply(lambda x: x + g)

                        SSRT1 = SubLevDF1[["SSRT"]]

                        SSRT2 = SubLevDF2[["SSRT"]]

                        SSE1 = ((e/2)-1)\*(np.var(SSRT1, ddof=1))

                        SSE2 = ((e/2)-1)\*(np.var(SSRT2, ddof=1))

                        pooledvar = (SSE1 + SSE2) / ((e/2) + (e/2) -2)

                        pooledSD = np.sqrt(pooledvar)

                        d.append(float(np.mean(SSRT1 - SSRT2)/pooledSD))

                        ttest = stats.ttest\_ind(SSRT1, SSRT2, nan\_policy='propagate')

                        print(str(h) + "th power test:" + str(e) + "," + str(f) + "," + str(g))

                        if ttest.pvalue < .050:

                            powercounter = powercounter + 1

                    CohenD = abs(np.mean(d))

                    finalpower = powercounter/h

                    print("I Have The POWER: " + str(e) + "," + str(f) + "," + str(g) + ": " + str(finalpower) + "%")

                    Experiment = [[e,f,g,CohenD,finalpower]]

                    ExpPower = pd.DataFrame(Experiment,columns=['Numb\_of\_Subs', 'Numb\_of\_Trials', 'Effect\_Magnitude', 'Effect\_Size', 'Power'])

                    powerResults = powerResults.append(ExpPower)

    elif stopSignal == True and designType == 'within':

# Automating 30 Structural Equation Models to Examine Self-control and Health Behaviors (R, Mplus)

This R code uses the MPlusAutomation package to run 30 structural equation models from a source MPlus (latent variable modeling software) file. The code then wrangles the data into tabular form (see table and model below code). This work was conducted for this published manuscript: http://scanlab.missouri.edu/docs/pub/2020/VonGunten\_etal\_2020\_EJP.pdf

R code:

library(MplusAutomation)

####### A) Setting working directory #######

setwd("C:/Users/Curt/Box Sync/Bruce Projects/Dissertation/Manuscript/For Publication/EJoP/Revision/SEM/Automation")

####### B) Creating Mplus INP files #######

createModels(templatefile="Master Structural Model\_v2.inp")

####### C) Running Mplus INP files #######

runModels(target = paste0(getwd(),"/Output"),

          recursive=TRUE,

          showOutput=FALSE,

          replaceOutfile="modifiedDate")

####### D) Reading the the Mplus OUT files #######

ReadModels <- readModels(target = paste0(getwd(), "/Output"))

####### E) Extracting summary and model fit statistics #######

library(plyr)

library(tibble)

fitSummary <- as\_tibble(do.call("rbind.fill", sapply(ReadModels, "[", "summaries")))

#View(fitSummary)

#######  F) Extracting parameter statistics #######

paramSummary <- sapply(sapply(ReadModels,

                                      "[", "parameters"),

                               "[", "stdyx.standardized")

####### G) Tabulating all predictor associations with the outcomes #######

params = c(3,6)

variables = c(9:14)

allPred = c()

for (outcome in 1:length(paramSummary)){

  for (variable in variables){

    for (parameter in params){

      allPred = c(allPred, paramSummary[[outcome]][variable, parameter])

    }}}

allPred\_df <- data.frame(matrix(unlist(allPred), nrow=28, byrow=TRUE))

outcomeNames = fitSummary['Title']

names(outcomeNames) = c("Outcome")

allPred\_df$Outcome = outcomeNames

allPred\_df = allPred\_df[,c(13,1:12)]

names(allPred\_df) = c("Outcome", "Inhib(est)", "Inhib(p)", "SC(est)", "SC(p)",

                       "Sex(est)", "Sex(p)", "SES(est)", "SES(p)",

                       "Raven(est)", "Raven(p)", "SocDes(est)", "SocDes(p)")

####### H) Calculating and tabulating mean coef and number of significant pvalues for each predictor across all 28 outcomes #######

allPred\_df = allPred\_df[-c(25),] #Removing SexRisk.

means = c()

mins = c()

maxes = c()

for (coef in seq(2,12,2)){

  tempMean = mean(allPred\_df[,c(coef)])

  tempmin = min(allPred\_df[,c(coef)])

  tempmax = max(allPred\_df[,c(coef)])

  means = c(means, tempMean)

  mins = c(mins, tempmin)

  maxes = c(maxes, tempmax)

}

counts = c()

for (ps in seq(3,13,2)){

  tempCount = count(allPred\_df[,c(ps)] <= .05)[2][2,1]

  counts = c(counts, tempCount)

}

counts

summaries = rbind(means,mins,maxes,counts)

summaries\_df = data.frame(summaries)

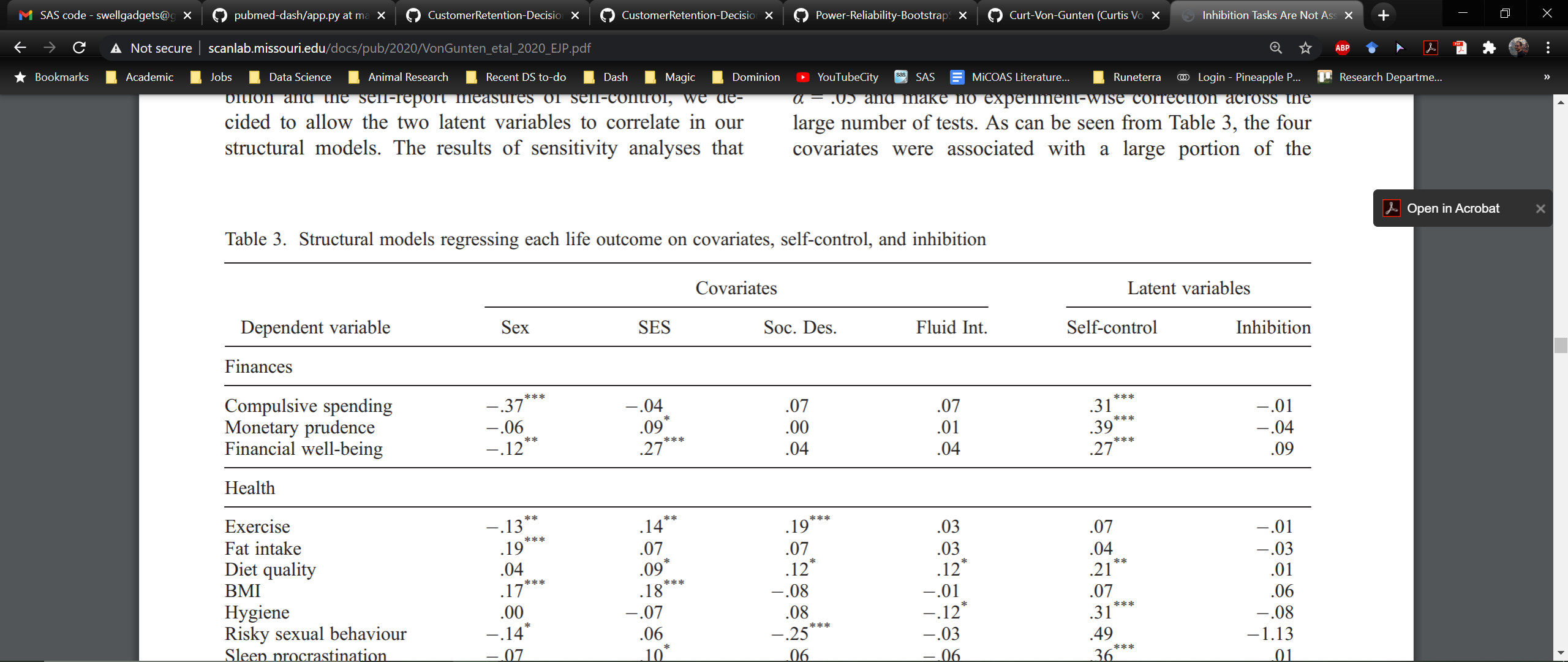
names(summaries\_df) = c("Inhib", "SC", "Sex", "SES", "Raven", "SocDes")

####### I) Outputting DFs to txt files #######

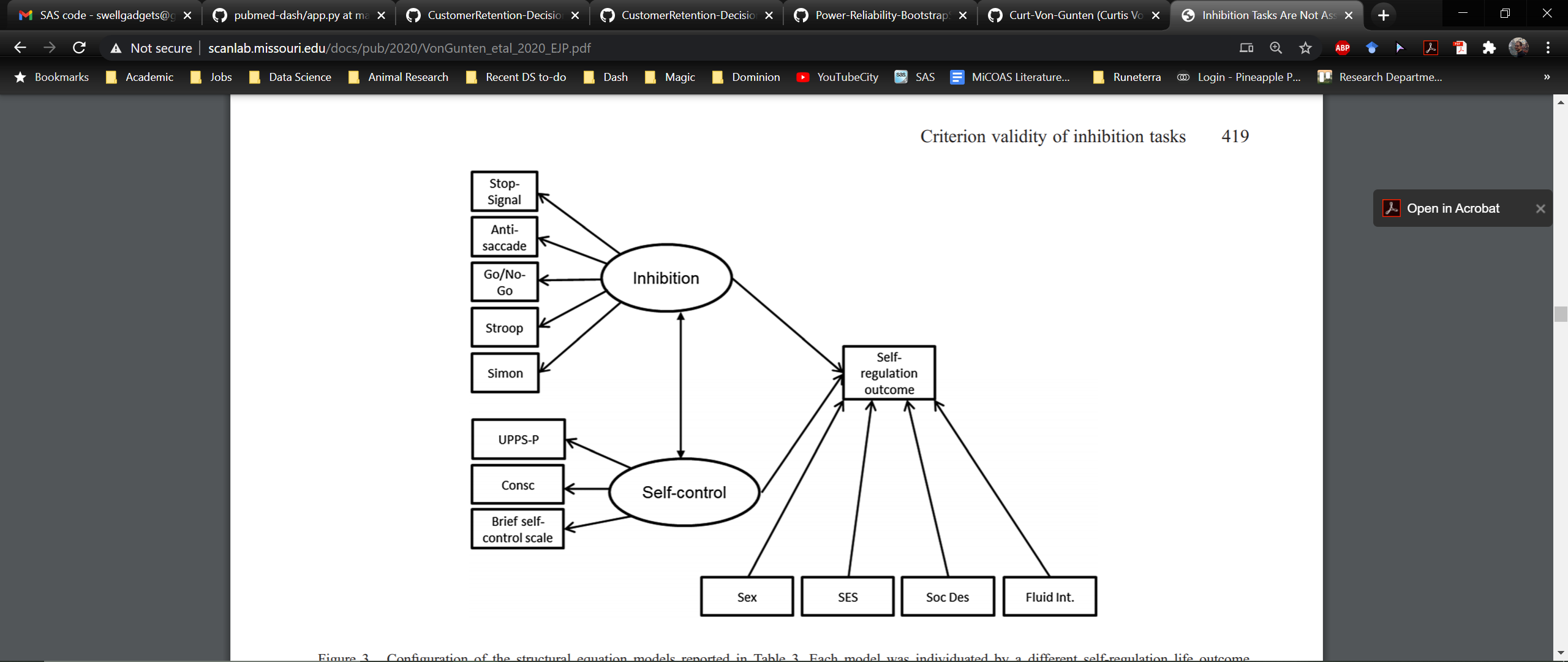
write.table(x=allPred\_df[,-1], file='AllPred\_Corr.txt',sep='\t',row.names=F)

write.table(x=summaries\_df, file='PredSummaries\_Corr.txt',sep='\t',row.names=F)

Table:

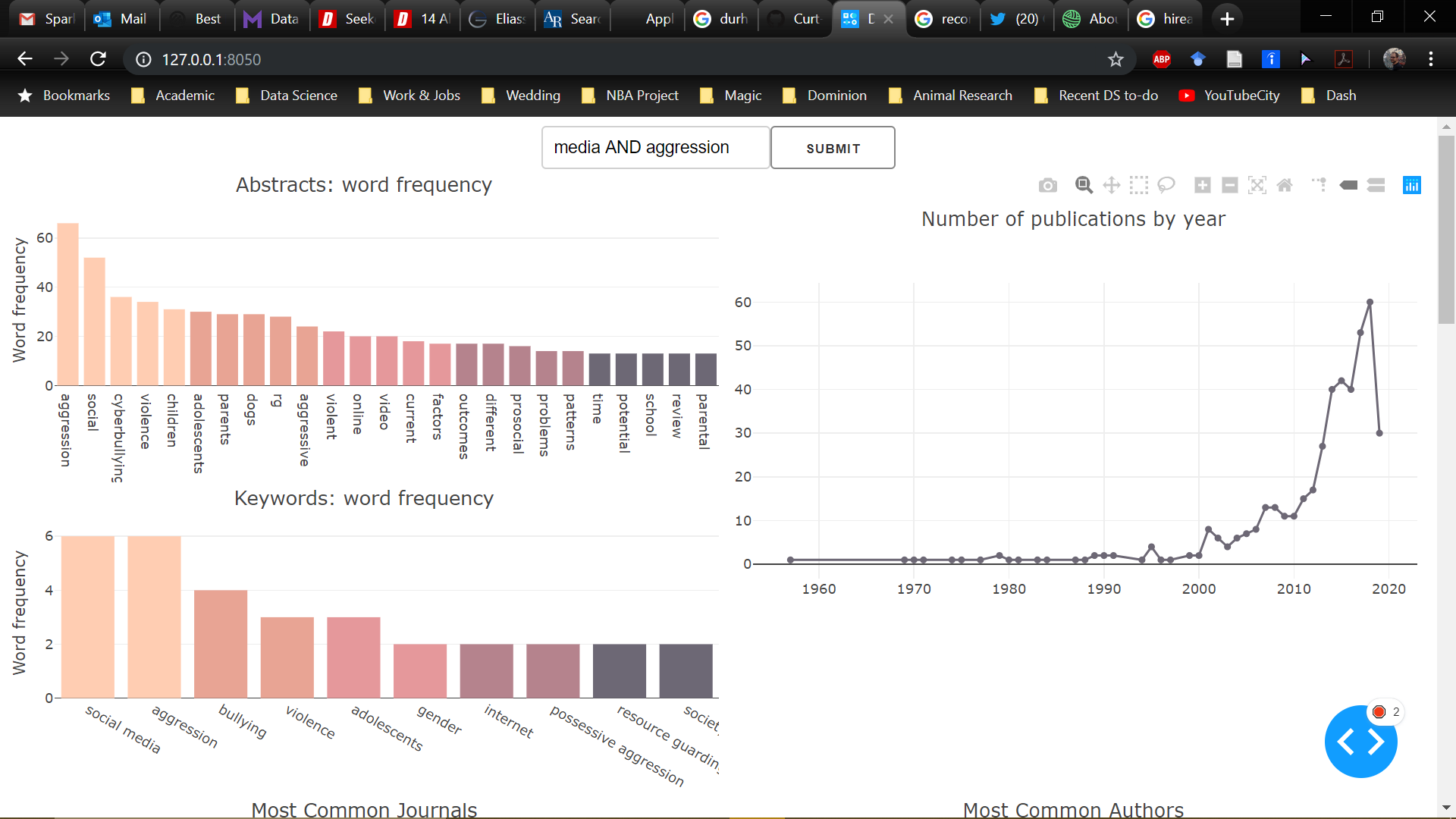


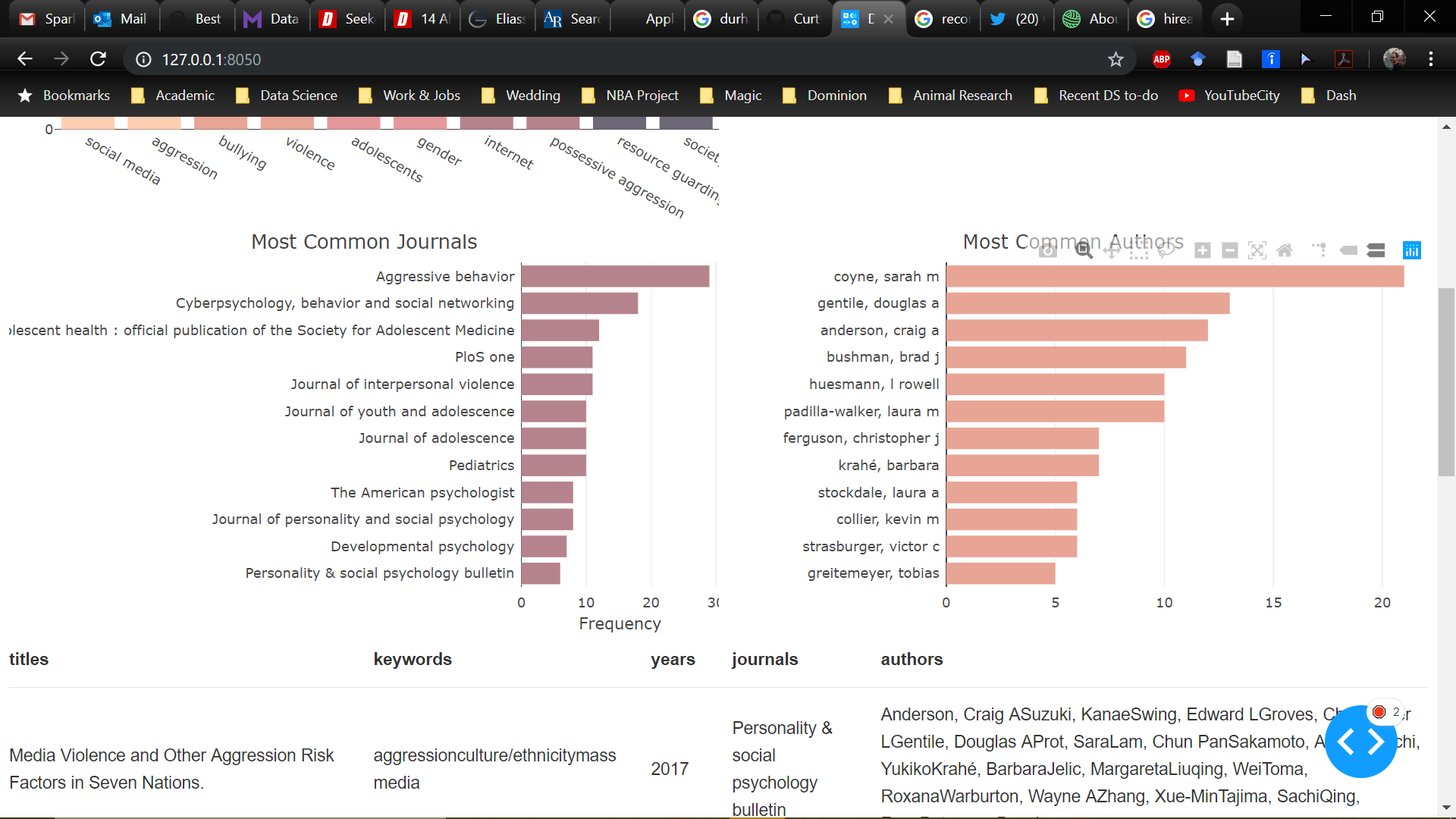
Model:



# PubMed Dashboard Application (Python, Heroku, Flask, HTML, CSS)

This Python code is an app I developed. It uses Python, Flask, and Heroku to query the PubMed database for a research topic provided by the user. It performs basic to intermediate natural language processing to make research trends easier to observe. The app is 413 lines long. Here I provide a snippet and photos of a query (‘media AND aggression’).





Python PubMed app:

# -\*- coding: utf-8 -\*-

"""

Created on Mon Jul 29 13:09:01 2019

@author: Curt

"""

import dash

import dash\_core\_components as dcc

import dash\_html\_components as html

from dash.dependencies import Input, Output, State

import pandas as pd

from Bio import Entrez

import string

import nltk

from collections import Counter

from plotly.offline import plot

import plotly.graph\_objects as go

external\_stylesheets = ['https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap-grid.min.css']

app = dash.Dash(\_\_name\_\_, external\_stylesheets=external\_stylesheets)

def get\_keywords(xml\_data):

        kewWordList = []

        for article in xml\_data:

            tempKewWordList = []

            try:

                kewWords = article['MedlineCitation']['KeywordList'][0]

                for keyword in kewWords:

                    tempKewWordList.append(str(keyword))

            except:

                tempKewWordList.append('empty')

                print('No keywords for this article')

            kewWordList.append(tempKewWordList)

        return kewWordList

def get\_dois(xml\_data, article\_idtype='doi'):

        doiList = []

        for i, article in enumerate(xml\_data):

            try:

                dois = article['PubmedData']['ArticleIdList']

                for doi in dois:

                    if doi.attributes['IdType'].lower() == article\_idtype:

                        doiList.append(str(doi))

            except:

                doiList.append(str('empty'))

            if len(doiList) != i+1:

                doiList.append(str('empty'))

        return doiList

# =============================================================================

# =============================================================================

# =============================================================================

app.layout = html.Div([

    html.Div([

        dcc.Input(id='input', value='self-control', type='text'),

        html.Button(id='submit-button', n\_clicks=0, children='Submit')],

    style={'textAlign': 'center'}),

    html.Div(id='intermediate-value', style={'display': 'none'}),

    html.Div([

        html.Div([

            html.Div([

                dcc.Graph(id='graph\_abstract'),

                dcc.Graph(id='graph\_keywords')],

            style={'margin': 'auto', 'display': 'inline-block', 'width': '100%'})]),

        html.Div(dcc.Graph(id='graph\_scatter', clickData={'points': [{'customdata': '2018'}]}),

        style={'margin': 'auto', 'display': 'inline-block', 'width': '100%'})],

    style={'columnCount': 2, 'columnGap': '0%', 'columnRule': '4px'}),

    html.Div([

        dcc.Graph(id='graph\_journal'),

        dcc.Graph(id='graph\_author')],

    style={'columnCount': 2, 'columnGap': '0%'}),

    html.Div(id='table')

    ])

# =============================================================================

# =============================================================================

# =============================================================================

@app.callback(Output('intermediate-value', 'children'),

              [Input(component\_id='submit-button', component\_property='n\_clicks')],

              [State(component\_id='input', component\_property='value')])

def get\_data(n\_clicks, query):

    query = str(query)

    howmany = 500

    searchResults = Entrez.read(Entrez.esearch(db='pubmed', retmode='xml', term=query, retmax=howmany, sort='relevance'))

    ids = str(searchResults['IdList'])

    handle = Entrez.efetch(db='pubmed', retmode='xml', id=ids)

    xml\_data = Entrez.read(handle)['PubmedArticle']

    titles = get\_titles(xml\_data)

    keywords = get\_keywords(xml\_data)

    abstracts = get\_abstracts(xml\_data)

    years = get\_years(xml\_data)

    journals = get\_journals(xml\_data)

    authors = get\_authors(xml\_data)

    pages = get\_pages(xml\_data)

    dois = get\_dois(xml\_data)

    return df.to\_json(orient='split')

@app.callback(Output('table', 'children'), [Input('intermediate-value', 'children')])

def update\_table(jsonified\_cleaned\_data, max\_rows=10):

    dataframe = pd.read\_json(jsonified\_cleaned\_data, orient='split')

    dataframe = dataframe[['titles','keywords','years','journals','authors']]

    return html.Table(

        # Header

        [html.Tr([html.Th(col) for col in dataframe.columns])] +

        # Body

        [html.Tr([

            html.Td(dataframe.iloc[i][col]) for col in dataframe.columns

        ]) for i in range(min(len(dataframe), max\_rows))]

    )

@app.callback(Output('graph\_keywords', 'figure'),

              [Input('intermediate-value', 'children'),

               Input('graph\_scatter', 'clickData')])

def update\_keyword\_graph(jsonified\_cleaned\_data, clickData):

    dataframe = pd.read\_json(jsonified\_cleaned\_data, orient='split')

    filtered\_df = dataframe[dataframe['years'] == clickData['points'][0]['customdata']]

    key\_ser = filtered\_df['keywords']

    key\_strings = [','.join(words) for words in key\_ser]

    key\_joined = ','.join(words for words in key\_strings)

    keywords\_tokens = [t.lower() for t in key\_joined.split(',')]

    word\_counts = Counter(keywords\_tokens)

    freq\_df = pd.DataFrame(word\_counts.items(), columns=['word', 'count'])

    freq\_df = freq\_df[freq\_df['word'] != 'empty']

    freq\_df.sort\_values(by=['count'], inplace=True, ascending=False)

    #freq\_df = freq\_df[freq\_df['count'] >= 10]

    freq\_df = freq\_df.iloc[:12]

    freq\_df = freq\_df[freq\_df['count'] != max(freq\_df['count'])]

    rem = len(freq\_df) % 5

    freq\_df\_rem = freq\_df.iloc[:-rem]

    NinBin = int(len(freq\_df\_rem) / 5)

    binIndices = [int(x \* NinBin) for x in range(1,6)]

    freq\_df\_binned = [freq\_df\_rem.iloc[x-NinBin:x] for x in binIndices]

    return {

        'data': [go.Bar(

            x = list(df['word'].values),

            y = list(df['count'].values),

            marker={'color': colors[i]})

            for i, df in enumerate(freq\_df\_binned)],

        'layout': {

            'yaxis': {'title': 'Word frequency'},

            'height': 275,

            'margin': {'l': 40, 'b': 85, 'r': 0, 't': 40},

            'title': 'Keywords: word frequency',

            'showlegend': False,

            }

        }

@app.callback(Output('graph\_journal', 'figure'),

              [Input('intermediate-value', 'children')])

def graph\_journal(jsonified\_cleaned\_data):

    dataframe = pd.read\_json(jsonified\_cleaned\_data, orient='split')

    filtered\_df = dataframe[dataframe['journals'] != 'empty']

    count\_df = filtered\_df[['titles','journals']].groupby('journals', as\_index=False).count()

    count\_df.sort\_values(by=['titles'], inplace=True, ascending=True)

    count\_df = count\_df.iloc[-12:]

    return {

        'data': [go.Bar(

            x = count\_df['titles'],

            y = count\_df['journals'],

            marker={'color': '#B5838D'},

            orientation='h',

            )],

        'layout': {

            'xaxis': {'title': 'Frequency'},

            'height': 350,

            'margin': {'l': 450, 'b': 40, 'r': 0, 't': 25, 'pad': 5},

            'title': 'Most Common Journals',

            'showlegend': False,

            }

        }

if \_\_name\_\_ == '\_\_main\_\_':

    app.run\_server(debug=True)