Codebook for Web-scraping of State Legislative Info

Contents

Web scraping is an interactive process between computers and websites using HyperText Markup Language (HTML). Through HTML, web scraping can use bots to extract content and data from websites. Several softwares, like Python, R, and STATA can be used to do web scraping. Here legislative information web scraping is achieved by Python.

In this codebook, special Python language terms like Python module, library, function,and syntax are all italic and texts in the box represent code. Full codes without interpretation are in the appendix.

Codebook for State-Level Policies Affecting Agricultural Costs and Revenues in the U.S.: a

Skip to main content

comprehensive database (1975-2021), patterns, and analysis of policy correlates

```
![](pics/booklogo.png)
```

Chapter 1

Introduction

Introduction

Chapter 2 Web-scraping tool setup

In this chapter, we introduce the webscraping tool set.

Python Language

Python is a programming language that lets you work more quickly and integrate your systems more effectively (https://www.python.org/).

IDEs

An IDE (Integrated Development Environment) understand your code much better than a text editor. It usually provides features such as build automation, code linting, testing and debugging. This can significantly speed up your work. The downside is that IDEs can be complicated to use. There are several IDEs that users can consider. Here we recommend PyCharm or Visual Studio Code. You can download them from JetBrains' website https://www.jetbrains.com/pycharm/ or Visual Studio Code' websites https://code.visualstudio.com/.

Skip to main content

Libraries

A Python library, like packages in R, is a reusable chunk of code to save time. Several useful libraries are needed to install for Python to run this script.

Selenium is a powerful web scraping tool by automating browsers to load the target website, retrieve the required data, and take screenshots or assert that certain actions happen on the website. This script relies heavily on Selenium. In addition to Selenium, we also need to import other necessary packages such as pandas,time, os, PyPDF2, glob, pick

```python
# import libraries
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import Select
import time
from time import sleep
import pandas as pd
import datefinder
import calendar
import os
import unittest
from random import randint
import PyPDF2
import glob
import pickle
import numpy as np
import fitz
```

Our goal is to webscrape all session laws during 1975-2022 from state-level legislature websites. The General and Special Laws of Texas, often referred to as the “session laws,” constitute a complete set of all bills passed into law by each session of the state legislature.Session laws are organized into chapters, with each chapter consisting of a single “Act,” or bill. As bills are passed into law during a legislative session, the Secretary of State assigns each Act a corresponding chapter number.

Skip to main content

Name	FIPS State Numeric Code	Official USPS Code	Website Address
Alabama	1	AL	https://arc-sos.state.al.us/CGI/actyear.mbr/input
Alaska	2	AK	http://www.akleg.gov/basis/Home/BillsandLaws
Arizona	4	AZ	https://apps.azleg.gov/BillStatus/BillOverview?Sessionid=123
Arkansas	5	AR	https://www.arkleg.state.ar.us/Acts/
California	6	CA	https://leginfo.legislature.ca.gov/faces/home.xhtml
Colorado	8	CO	https://leg.colorado.gov/bills
Connecticut	9	CT	https://www.cga.ct.gov/asp/menu/legdownload.asp
Delaware	10	DE	https://legis.delaware.gov/AllLegislation
District of Columbia	11	DC	https://lims.dccouncil.us/searchresult/
Florida	12	FL	http://www.leg.state.fl.us/Welcome/index.cfm
Georgia	13	GA	https://www.legis.ga.gov/legislation/all
Hawaii	15	HI	https://www.capitol.hawaii.gov/
Idaho	16	ID	https://legislature.idaho.gov/statutesrules/sessionlaws/
Illinois	17	IL	https://www.ilga.gov/previousga.asp
Indiana	18	IN	http://iga.in.gov/results/#
Iowa	19	IA	https://www.legis.iowa.gov/legislation/billTracking
Kansas	20	KS	http://www.kslegislature.org/li/historical/

Skip to main content

Name	FIPS State Numeric Code	Official USPS Code	Website Address
Louisiana	22	LA	https://www.legis.la.gov/Legis/SessionInfo/SessionInfo.aspx
Maine	23	ME	https://legislature.maine.gov/ros/LOM/LOMpdfDirectory.htm
Maryland	24	MD	http://mgaleg.maryland.gov/mgawebsite/Search/FullText
Massachusetts	25	MA	https://malegislature.gov/Laws/SessionLaws
Michigan	26	MI	https://www.legislature.mi.gov/
Minnesota	27	MN	https://www.revisor.mn.gov/search/?search=stat
Mississippi	28	MS	http://billstatus.ls.state.ms.us/sessions.htm
Missouri	29	MO	https://house.mo.gov/LegislationSP.aspx?focusedID=Bill List
Montana	30	MT	http://laws.leg.mt.gov/legprd/law0203w$.startup?P_SESS=19991
Nebraska	31	NE	https://nebraskalegislature.gov/laws/laws.php
Nevada	32	NV	https://www.leg.state.nv.us/Site/Search/search.cfm
New Hampshire	33	NH	http://www.gencourt.state.nh.us/bill_status/legacy/bs2016/
New Jersey	34	NJ	https://www.njleg.state.nj.us/bills/Bills_ADVS.aspx#
New Mexico	35	NM	https://www.nmlegis.gov/Search
New York	36	NY	https://nyassembly.gov/leg/?sh=advanced
North Carolina	37	NC	https://www.ncleg.gov/Search/BillText/
North Dakota	38	ND	https://www.legis.nd.gov/search

Skip to main content

Name	FIPS State Numeric Code	Official USPS Code	Website Address
Oklahoma	40	OK	http://www.oklegislature.gov/AdvancedSearchForm.aspx
Oregon	41	OR	https://www.oregonlegislature.gov/bills_laws/Pages/Oregon-Laws.aspx
Pennsylvania	42	PA	https://www.legis.state.pa.us/cfdocs/legis/home/bills/
Rhode Island	44	RI	http://webserver.rilegislature.gov/search/
South Carolina	45	SC	https://www.scstatehouse.gov/query.php
South Dakota	46	SD	https://sdlegislature.gov/Statutes/Archived
Tennessee	47	TN	https://www.capitol.tn.gov/legislation/archives.html
Texas	48	TX	https://lrl.texas.gov/legis/billsearch/lrlhome.cfm
Utah	49	UT	https://le.utah.gov/asp/passedbills/passedbills.asp
Vermont	50	VT	https://legislature.vermont.gov/bill/search/2022
Virginia	51	VA	https://lis.virginia.gov/cgi-bin/legp604.exe?941+men+BIL
Washington	53	WA	http://search.leg.wa.gov/search.aspx#document
West Virginia	54	WV	https://www.wvlegislature.gov/Bill_Status/bill_status.cfm
Wisconsin	55	WI	https://docs.legis.wisconsin.gov/search
Wyoming	56	WY	https://www.wyoleg.gov/Legislation/archives

Since each state-level legislature website has

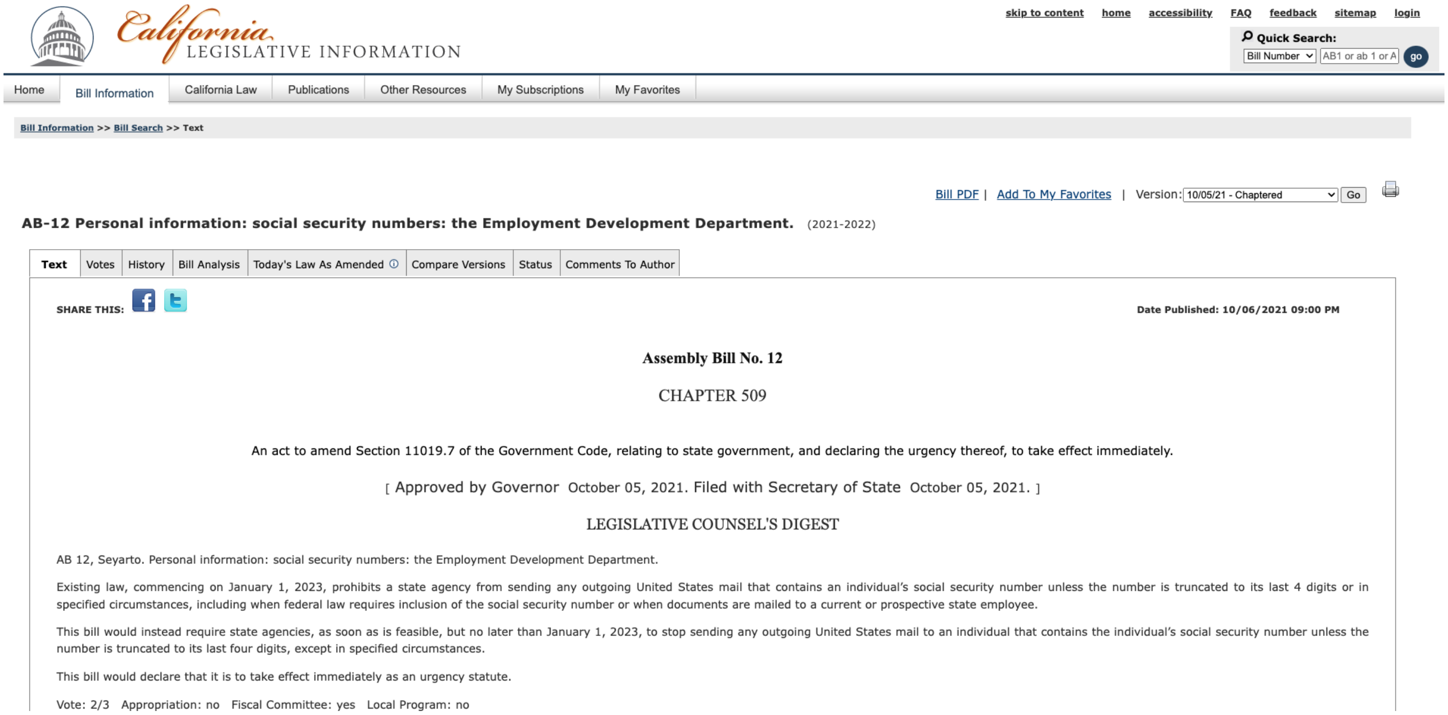
Skip to main content

Name	FIPS State Numeric Code	Official USPS Code	Website Address
website structure, we adopted three different webscraping strategies: (i) direct webscraping, (ii) downloading act PDF files and extraction, and (iii) mixing them.			

Direct Webscraping

Some legislature websites store acts on their websites as html files or PDF files. So, we can webscrape acts' full texts directly.

Skip to main content



AB-12 Personal information: social security numbers: the Employment Development Department. (2021-2022)

Date Published: 10/06/2021 09:00 PM

Assembly Bill No. 12

CHAPTER 509

An act to amend Section 11019.7 of the Government Code, relating to state government, and declaring the urgency thereof, to take effect immediately.

[Approved by Governor October 05, 2021. Filed with Secretary of State October 05, 2021.]

LEGISLATIVE COUNSEL'S DIGEST

AB 12, Seyarto. Personal information: social security numbers: the Employment Development Department.

Existing law, commencing on January 1, 2023, prohibits a state agency from sending any outgoing United States mail that contains an individual's social security number unless the number is truncated to its last 4 digits or in specified circumstances, including when federal law requires inclusion of the social security number or when documents are mailed to a current or prospective state employee.

This bill would instead require state agencies, as soon as is feasible, but no later than January 1, 2023, to stop sending any outgoing United States mail to an individual that contains the individual's social security number unless the number is truncated to its last four digits, except in specified circumstances.

This bill would declare that it is to take effect immediately as an urgency statute.

Vote: 2/3 Appropriation: no Fiscal Committee: yes Local Program: no

Chapter 3 Data cleaning

Skip to main content

Chapter 4 Topic Classification

Chapter 4.1 Keywords Generation

```
################################################################################
##############################   select gtm_py11 env   ##########################
################################################################################
#import libraries
# Check the Python version
import sys

sys.version
import os
import pandas as pd

working_directory = '../Guided-Topic-Modeling'
working_directory = '/Users/long/Library/CloudStorage/OneDrive-Personal/Projects/Gu
print(os.path.abspath(working_directory))
sys.path.append(working_directory)
print(sys.path)
import glob
from bertopic import BERTopic
#from sklearn.datasets import fetch_20newsgroups
import gensim
import gensim.corpora as corpora
import pickle
import matplotlib.pyplot as plt
import nltk
from nltk import bigrams
from nltk.tokenize import word_tokenize
import re
import openpyxl
import numpy as np

nltk.download('punkt')

# set up working directory
# Change the working directory
os.chdir(working_directory)
# Get the current working directory
cwd = os.getcwd()
# Print the current working directory
print("Current working directory: {0}".format(cwd))

sys.argv = [
    'gtm.py'
```

Skip to main content

```
    '--ps2', 'farm',
    '--pw1', '1.0',
    '--pw2', '0.000000000000000000000000001',
    '--size', '1000',
    '--gravity', '0.1'
    # Add '--ns1', '--ns2', '--nw1', '--nw2' and their values if needed
]

exec(open("gtm.py").read())

sys.argv = [
    'gtm.py',
    '--ps1', 'agriculture',
    '--ps2', 'farm',
    '--pw1', '1.0',
    '--pw2', '0.000000000000000000000000001',
    '--size', '2000',
    '--gravity', '0.1'
    # Add '--ns1', '--ns2', '--nw1', '--nw2' and their values if needed
]

exec(open("gtm.py").read())

# Extracting values from sys.argv
ps1_index = sys.argv.index('--ps1') + 1
size_index = sys.argv.index('--size') + 1
gravity_index = sys.argv.index('--gravity') + 1

ps1_value = sys.argv[ps1_index]
size_value = sys.argv[size_index]
gravity_value = sys.argv[gravity_index]

# Specify the folder path
folder_path = 'output'  # Replace with the actual folder path
print(os.path.abspath(folder_path))

# Find the last CSV file
csv_files = glob.glob(os.path.join(folder_path, '*.csv'))
latest_csv_file = max(csv_files, key=os.path.getmtime)

# Construct new file name
new_file_name = f"{ps1_value}_{size_value}_{gravity_value}.csv"
new_file_path = os.path.join(folder_path, new_file_name)

# Rename the file
os.rename(latest_csv_file, new_file_path)
print(f"File '{latest_csv_file}' has been renamed to '{new_file_path}'")

# Read the CSV file (if needed)
topics_dict = pd.read_csv(new_file_path)
topics_dict.rename(columns={'Unnamed: 0': 'keyword'}, inplace=True)

folder_path_csd = '/Volumes/SSD/AERI/Data/Meachine Learning/gtm'
```

Skip to main content

```
# Read the CSV file (if needed)
topics_dict.to_csv(new_file_path_ssd)
print(f"File '{latest_csv_file}' has been renamed to '{new_file_path_ssd}'")
```

Skip to main content

4.2 Topic Classification

```
import pandas as pd
import os
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
from nltk.util import bigrams
import codecs
import ast  # Module for literal string evaluation

#from word_forms.word_forms import get_word_forms
from itertools import islice
import numpy as np
import matplotlib.pyplot as plt
import geopandas as gpd

###
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter
import matplotlib.colors as mcolors
import seaborn as sns
import geopandas as gpd
from shapely.geometry import Polygon
import missingno as msno
import os
import wget
import openpyxl
import math
from wordcloud import WordCloud

import pickle
from collections import defaultdict

###

# Set the working directory
os.chdir('/Volumes/SSD/AFRI/Data/')

# Verify the current working directory
print(os.getcwd())


######## functions

def generate_uni_bigrams_individual(text):
```

Skip to main content

```
    # Try decoding the text using ISO-8859-1 encoding
    try:
        text = text.decode('ISO-8859-1')
    except AttributeError:
        pass  # Skip if the text is already a string
    except UnicodeDecodeError:
        # If decoding with ISO-8859-1 fails, try UTF-8 with errors='replace'
        text = text.decode('utf-8', errors='replace')

    # Tokenize the text
    tokens = word_tokenize(text)

    # Remove non-alphabetic tokens and lowercase the alphabetic tokens
    unigrams = [word.lower() for word in tokens if word.isalpha()]

    # Generate bigrams from the cleaned tokens
    bigram_tuples = list(bigrams(unigrams))

    # Join the words in each bigram with an underscore
    bigrams_formatted = ['_'.join(bigram) for bigram in bigram_tuples]

    # Combine unigrams and bigrams into one list
    combined_list = unigrams + bigrams_formatted

    return combined_list


def generate_uni_bigrams_folder(source_folder, destination_folder):
    # Create the destination folder if it doesn't exist
    if not os.path.exists(destination_folder):
        os.makedirs(destination_folder)

    # Iterate over each file in the source folder
    for filename in os.listdir(source_folder):
        if filename.endswith(".csv") and not filename.startswith('._'):
            print(filename)
            # Read the CSV file into a DataFrame
            source_filepath = os.path.join(source_folder, filename)
            df = pd.read_csv(source_filepath)

            # Drop the 'Unnamed: 0' column if it exists
            if 'Unnamed: 0' in df.columns:
                df.drop('Unnamed: 0', axis=1, inplace=True)

            # Apply the function generate_uni_bigrams to create 'uni_bigrams' colum
            df['uni_bigrams'] = df['original_text'].apply(generate_uni_bigrams_indi

            # Define the new filename for the destination
            csv_filename = filename.replace("_clean.csv", "_processed.csv")
            destination_filepath_csv = os.path.join(destination_folder, csv_filenam

            json_filename = filename.replace("_clean.csv", "_processed.json")
```

Skip to main content

```python
            pkl_filename = filename.replace("_clean.csv", "_processed.pkl")
            destination_filepath_pkl = os.path.join(destination_folder, pkl_filenam

            # Save the modified DataFrame to a new CSV file in the destination fold
            df.to_csv(destination_filepath_csv, index=False)
            df.to_json(destination_filepath_json, orient='records')
            df.to_pickle(destination_filepath_pkl)

            print(f"Processed file '{filename}' successfully saved")


def count_total_occurrences(text, keywords):
    # Initialize a dictionary to store word counts
    word_counts = {word: 0 for word in keywords}

    for word in text:
        if word.lower() in word_counts:  # Convert to lowercase for case-insensitiv
            word_counts[word.lower()] += 1

    # Sum all the occurrences
    total_occurrences = sum(word_counts.values())

    return total_occurrences


def count_individual_occurrences(text, keywords):
    # Initialize a dictionary to store counts of keywords, starting at 0
    word_counts = {word: 0 for word in keywords}

    # Count occurrences of each keyword in the text
    for word in text:
        word_lower = word.lower()  # Convert word to lowercase to ensure case-insen
        if word_lower in word_counts:
            word_counts[word_lower] += 1

    # Filter out keywords with zero occurrences
    word_counts = {word: count for word, count in word_counts.items() if count > 0}

    return word_counts


def count_individual_score(text, keyword_weights):
    # Initialize a dictionary to store counts of keywords, starting at 0
    word_counts = {word: 0 for word in keywords}

    # Count occurrences of each keyword in the text
    for word in text:
        word_lower = word.lower()  # Convert word to lowercase to ensure case-insen
        if word_lower in word_counts:
            word_counts[word_lower] += 1

    # Filter out keywords with zero occurrences
```

Skip to main content

```
    return word_counts


def calculate_keyword_scores(text, keywords, keyword_weights):
    # Initialize a dictionary to store counts of keywords
    word_counts = {word: 0 for word in keywords}

    # Count occurrences of each keyword in the text
    for word in text:
        word_lower = word.lower()
        if word_lower in word_counts:
            word_counts[word_lower] += 1

    # Calculate scores for each keyword based on its weight and occurrence count
    keyword_scores = {word: count * keyword_weights[word] for word, count in word_c

    # Sort the keywords by their scores in descending order
    sorted_keyword_scores = sorted(keyword_scores.items(), key=lambda item: item[1]

    # Convert the sorted list of tuples back into a dictionary
    sorted_keyword_scores_dict = dict(sorted_keyword_scores)

    return sorted_keyword_scores_dict


def calculate_weighted_score(text, keyword_weights):
    # Initialize the total score
    total_score = 0
    #print(text)
    # Split the text into words
    for word in text:
        # If the word is in the list and has a weight, add its weighted score
        # Check if the word is in the keyword_weights dictionary and add its weight
        if word in keyword_weights:
            total_score += keyword_weights[word]
    return total_score


def extract_top_five_items(dictionary):
    # Convert the string representation of dictionary to a dictionary object
    dictionary = ast.literal_eval(dictionary)
    # Sort the dictionary by values in descending order and take the first five ite
    top_five_items = dict(sorted(dictionary.items(), key=lambda item: item[1], reve
    return top_five_items


def extract_top_five_items(dictionary):
    sorted_items = sorted(dictionary.items(), key=lambda item: item[1], reverse=Tru

    # Convert sorted items back to a dictionary if needed
    top_five_items = dict(sorted_items)
    return top_five_items
```

Skip to main content

```
def count_words(text):
    text = str(text)
    words = text.split()
    return len(words)


def classification(source_folder, destination_folder, keywords, keyword_weights):
    # Create the destination folder if it doesn't exist
    if not os.path.exists(destination_folder):
        os.makedirs(destination_folder)

    # Iterate over each file in the source folder
    for filename in os.listdir(source_folder):
        if filename.endswith(".pkl") and not filename.startswith('._'):
            print(filename)
            # Read the CSV file into a DataFrame
            source_filepath = os.path.join(source_folder, filename)

            with open(source_filepath, 'rb') as file:
                df = pickle.load(file)

                # Drop the 'Unnamed: 0' column if it exists
                if 'Unnamed: 0' in df.columns:
                    df.drop('Unnamed: 0', axis=1, inplace=True)

                # Count occurrences of keywords in uni_bigrams
                df['uni_bigrams_occurrences'] = df['uni_bigrams'].apply(lambda x: c

                # Count occurrences of each keyword in the text
                df['uni_bigrams_word_counts'] = df['uni_bigrams'].apply(
                    lambda x: count_individual_occurrences(x, keywords))

                # top 5 keywords with scores
                df['top5_uni_bigrams_word_counts'] = df['uni_bigrams_word_counts'].
                    lambda x: extract_top_five_items(x))

                # Calculate scores of each keyword in the text
                df['uni_bigrams_word_scores'] = df['uni_bigrams'].apply(
                    lambda x: calculate_keyword_scores(x, keywords, keyword_weights

                # Calculate total scores of keywords in the text
                df['total_weighted_score'] = df['uni_bigrams'].apply(
                    lambda x: calculate_weighted_score(x, keyword_weights))

                # Count the words
                df['original_text_word_count'] = df['original_text'].apply(count_wo

                # Calculate total scores of keywords per word in the text
                df['total_weighted_score_per_word'] = df.apply(
                    lambda row: row['total_weighted_score'] / row['original_text_wo

                    axis=1)
```

Skip to main content

```
# top 5 keywords with scores
df['top5_uni_bigrams_word_scores'] = df['uni_bigrams_word_scores'].
    lambda x: extract_top_five_items(x))

#.apply(lambda x: extract_top_five_items(x) if isinstance(x, str) e

# Define the new filename for the destination
csv_filename = filename.replace("_processed.pkl", "_classified.csv"
destination_filepath_csv = os.path.join(destination_folder, csv_fil

json_filename = filename.replace("_processed.pkl", "_classified.jso
destination_filepath_json = os.path.join(destination_folder, json_f

pkl_filename = filename.replace("_processed.pkl", "_classified.pkl"
destination_filepath_pkl = os.path.join(destination_folder, pkl_fil

# Save the modified DataFrame to a new CSV file in the destination
#df.to_csv(destination_filepath_csv, index=False)
#df.to_json(destination_filepath_json, orient='records')
df.to_pickle(destination_filepath_pkl)

# This will hold the combined results
combined_dict = defaultdict(int)

# Iterate through each dictionary in the DataFrame's column
for index, row in df.iterrows():
    for key, value in row['uni_bigrams_word_counts'].items():
        combined_dict[key] += value

# Convert the defaultdict back to a regular dictionary for display
result_dict_count = dict(combined_dict)

# Create a WordCloud object
wordcloud = WordCloud(width=800, height=400, background_color='whit

# Generate a word cloud from frequencies
wordcloud.generate_from_frequencies(result_dict_count)

# Display the generated image:
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')  # Do not show axes to keep it clean

pic_filename = filename.replace("_processed.pkl", "_wordcloud_count
destination_filepath_pic = os.path.join(destination_folder, pic_fil

# Save the figure to a file
plt.savefig(destination_filepath_pic, format='png', bbox_inches='ti

dict_count_filename = filename.replace("_processed.pkl", "_dic_coun
destination_filepath_dict = os.path.join(destination_folder, dict_c

# Convert dictionary to DataFrame
```

Skip to main content

```
                # Save to CSV
                result_list_count.to_csv(destination_filepath_dict, index=False)

                # This will hold the combined results
                combined_dict = defaultdict(int)

                # Iterate through each dictionary in the DataFrame's column
                for index, row in df.iterrows():
                    for key, value in row['uni_bigrams_word_scores'].items():
                        combined_dict[key] += value

                # Convert the defaultdict back to a regular dictionary for display
                result_dict_score = dict(combined_dict)

                # Create a WordCloud object
                wordcloud = WordCloud(width=800, height=400, background_color='whit

                # Generate a word cloud from frequencies
                wordcloud.generate_from_frequencies(result_dict_score)

                # Display the generated image:
                plt.figure(figsize=(10, 5))
                plt.imshow(wordcloud, interpolation='bilinear')
                plt.axis('off')  # Do not show axes to keep it clean

                pic_filename = filename.replace("_processed.pkl", "_wordcloud_score
                destination_filepath_pic = os.path.join(destination_folder, pic_fil

                # Save the figure to a file
                plt.savefig(destination_filepath_pic, format='png', bbox_inches='ti

                dict_count_filename = filename.replace("_processed.pkl", "_dic_scor
                destination_filepath_dict = os.path.join(destination_folder, dict_c

                # Convert dictionary to DataFrame
                result_list_score = pd.DataFrame(list(result_dict_score.items()), c

                # Save to CSV
                result_list_score.to_csv(destination_filepath_dict, index=False)

                print(f"Classified file '{filename}' successfully saved")


source_folder = "./Raw_Data/Cleaned"
destination_folder = "./Processed_Data"

generate_uni_bigrams_folder(source_folder, destination_folder)

keywords = pd.read_excel('./Meachine Learning/gtm/agriculture_1000_0.1_human_refine
    'keyword'].values.tolist()
keyword_weights = dict(
    zip(pd.read_excel('./Meachine Learning/gtm/agriculture_1000_0.1_human_refinemen
```

Skip to main content

```
source_folder = "./Processed_Data"
destination_folder = "./Outcomes"

classification(source_folder, destination_folder, keywords, keyword_weights)
########
destination_folder = "./Outcomes"

# Define the columns to keep
columns_to_keep = ['state', 'year', 'act_num', 'uni_bigrams_occurrences', 'uni_bigr
                   'top5_uni_bigrams_word_counts', 'uni_bigrams_word_scores', 'tota
                   'original_text_word_count',
                   'total_weighted_score_per_word', 'top5_uni_bigrams_word_scores']

# Initialize an empty DataFrame to hold all the data
df = pd.DataFrame()

# Loop through each file in the folder
for filename in os.listdir(destination_folder):
    if filename.endswith('count.csv') and not filename.startswith('._'):  # Check i
        print(filename)
        file_path = os.path.join(destination_folder, filename)
        data = pd.read_csv(file_path)
        #with open(file_path, 'rb') as file:
        #data = pickle.load(file)
        #data = data[columns_to_keep]
        # Append the data to the main DataFrame
        df = df._append(data, ignore_index=True)

#df['top5_uni_bigrams_word_scores'] = df['uni_bigrams_word_scores'].apply(lambda d:
df = df.groupby('Word')['Frequency'].sum().reset_index()
df.to_csv('./Outcomes/all_words_count.csv')

# Convert DataFrame to dictionary
word_freq = dict(zip(df['Word'], df['Frequency']))

# Create a WordCloud object
wordcloud = WordCloud(width=800, height=400, background_color='white')

# Generate a word cloud
wordcloud.generate_from_frequencies(word_freq)

# Display the generated image:
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')  # Do not show axes to keep it clean
plt.show()
plt.savefig('./Outcomes/counts.png', format='png', bbox_inches='tight', dpi=300)

# Initialize an empty DataFrame to hold all the data
df = pd.DataFrame()

# Loop through each file in the folder
for filename in os.listdir(destination_folder):
```

Skip to main content

```
        print(filename)
        file_path = os.path.join(destination_folder, filename)
        data = pd.read_csv(file_path)
        #with open(file_path, 'rb') as file:
        #data = pickle.load(file)
        #data = data[columns_to_keep]
        # Append the data to the main DataFrame
        df = df._append(data, ignore_index=True)

#df['top5_uni_bigrams_word_scores'] = df['uni_bigrams_word_scores'].apply(lambda d:
df = df.groupby('Word')['Frequency'].sum().reset_index()
df.to_csv('./Outcomes/all_words_score.csv')

# Convert DataFrame to dictionary
word_freq = dict(zip(df['Word'], df['Frequency']))

# Create a WordCloud object
wordcloud = WordCloud(width=800, height=400, background_color='white')

# Generate a word cloud
wordcloud.generate_from_frequencies(word_freq)

# Display the generated image:
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')  # Do not show axes to keep it clean
plt.show()
plt.savefig('./Outcomes/scores.png', format='png', bbox_inches='tight', dpi=300)

df = pd.read_csv('./Outcomes/classification_results.csv')

try:
    df = df.drop(['Unnamed: 0'], axis=1)
except:
    pass

df['year'] = pd.to_numeric(df['year'], errors='coerce')
df.dropna(subset=['year'], inplace=True)

df['year'] = df['year'].astype(str).str[:4].astype(int)

df = df[(df['year'] >= 1975) & (df['year'] <= 2021)]

# Group by 'year' and 'state' and calculate the count of positive scores and the to
grouped_data = df.groupby(['year', 'state']).agg(
    positive_score_count=('total_weighted_score_per_word', lambda x: (x > 0).sum())
    total_score_count=('total_weighted_score_per_word', 'count')
)

# Calculate the proportion of the count of positive scores to the total count of sc
grouped_data['proportion'] = grouped_data['positive_score_count'] / grouped_data['t

# Reset the index to make 'year' and 'state' columns again
```

Skip to main content

```
grouped_data.to_csv('./Outcomes/classification_results_short.csv')

# Pivot the data for plotting
pivot_data = grouped_data.pivot(index='year', columns='state', values='proportion')

plt.figure(figsize=(30, 24))  # Increase figure size for better clarity and space
pivot_data.plot(kind='line')

# Increase font sizes for better readability
plt.title('Proportion of Ag Legislation per Year by State', fontsize=12)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Proportion of Ag Legislation', fontsize=12)

# Increase tick label size
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Expand the right margin to ensure the legend and plot do not overlap
plt.subplots_adjust(right=0.65)

# Place the legend to the right of the plot, making it larger to ensure legibility
plt.legend(title='State', loc='center left', bbox_to_anchor=(1.05, 0.5), ncol=2, fo

# Add grid for better readability of the plot
plt.grid(True)

plt.show()

state_data = df.groupby('state').agg(
    ag_count=('total_weighted_score_per_word', lambda x: (x > 0).sum()),
    ag_count_0004=('total_weighted_score_per_word', lambda x: (x > 0.004).sum()),
    total_count=('total_weighted_score_per_word', 'count')
)

state_data['ag_proportion_0000'] = state_data['ag_count'] / state_data['total_count
state_data['ag_proportion_0004'] = state_data['ag_count_0004'] / state_data['total_
state_data.to_csv('./Outcomes/state.csv')

state_data = pd.read_csv('./Outcomes/state.csv')

gdf = gpd.read_file('./Geo/cb_2018_us_state_500k.shp')

fips = pd.read_excel('./Geo/statefp.xlsx')
fips['STATEFP'] = fips['STATEFP'].astype(str).apply(lambda x: x.zfill(2))

gdf = gdf.merge(fips, how='left', on='STATEFP')
gdf = gdf.merge(state_data, how='left', on='state')


# Apply this to the gdf to ensure all states are assigned colors by the same func
def makeColorColumn(gdf, variable, vmin, vmax):
    # apply a function to a column to create a new column of assigned colors & retu
```

Skip to main content

```
    mapper = plt.cm.ScalarMappable(norm=norm, cmap=plt.cm.YlOrBr)
    gdf['value_determined_color'] = gdf[variable].apply(lambda x: mcolors.to_hex(ma
    return gdf


# set the value column that will be visualised
variable = 'ag_proportion_0000'
#variable = 'ag_proportion_0004'

# make a column for value_determined_color in gdf
# set the range for the choropleth values with the upper bound the rounded up maxim

if variable == 'ag_proportion_0000':
    gdf['ag_proportion'] = gdf['ag_proportion_0000']
else:
    gdf['ag_proportion'] = gdf['ag_proportion_0004']

vmin, vmax = gdf.ag_proportion.min(), gdf.ag_proportion.max()

# Choose the continuous colorscale "YlOrBr" from https://matplotlib.org/stable/tuto
colormap = "YlOrBr"
gdf = makeColorColumn(gdf, variable, vmin, vmax)

# create "visframe" as a re-projected gdf using EPSG 2163 for CONUS
#visframe = gdf.to_crs({'init':'epsg:2163'})
visframe = gdf.to_crs({'proj': 'aea', 'lat_1': 29.5, 'lat_2': 45.5, 'lon_0': -96, '

# create figure and axes for Matplotlib
fig, ax = plt.subplots(1, figsize=(20, 20))
# remove the axis box around the vis
ax.axis('off')
# set the font for the visualization to Helvetica
hfont = {'fontname': 'Helvetica'}

# add a title and annotation
ax.set_title('State-Level Agricultural Legislation\n1975-2021', **hfont, fontdict={

# Create colorbar legend
fig = ax.get_figure()
# add colorbar axes to the figure
# This will take some iterating to get it where you want it [l,b,w,h] right
# l:left, b:bottom, w:width, h:height; in normalized unit (0-1)
cbax = fig.add_axes([0.89, 0.21, 0.03, 0.31])

cbax.set_title('Percentage \n of Ag Legislation \n (1975-2021)', **hfont,
               fontdict={'fontsize': '12', 'fontweight': '0'})

# add color scale
sm = plt.cm.ScalarMappable(cmap=colormap, \
                           norm=plt.Normalize(vmin=vmin, vmax=vmax))
# reformat tick labels on legend
sm._A = []
#comma_fmt = FuncFormatter(lambda x, p: format(x/100, '%'))
```

Skip to main content

```
fig.colorbar(sm, cax=cbax, format=comma_fmt)
tick_font_size = 16
cbax.tick_params(labelsize=tick_font_size)
# annotate the data source, date of access, and hyperlink
ax.annotate("Data: Authors", xy=(0.5, .085), xycoords='figure fraction', fontsize=1

# create map
# Note: we're going state by state here because of unusual coloring behavior when t
for row in visframe.itertuples():
    if row.state not in ['AK', 'HI']:  # Exclude Alaska and Hawaii for this part
        vf = visframe[visframe.state == row.state]
        if pd.isna(row.ag_proportion):  # Check if the ag_proportion is NaN
            color = 'lightgrey'  # Set color to grey for missing data
        else:
            color = gdf.loc[gdf.state == row.state, 'value_determined_color'].iloc[
        vf.plot(color=color, linewidth=1.5, ax=ax, edgecolor='0.8')

# add Alaska
akax = fig.add_axes([0.4, 0.25, 0.2, 0.13])
akax.axis('off')
# polygon to clip western islands
polygon = Polygon([(-170, 50), (-170, 72), (-140, 72), (-140, 50)])
alaska_gdf = gdf[gdf.state == 'AK']
alaska_gdf.clip(polygon).plot(color=gdf[gdf.state == 'AK'].value_determined_color,
                              edgecolor='0.8')

# add Hawaii
hiax = fig.add_axes([.58, 0.25, 0.1, 0.1])
hiax.axis('off')
`
# polygon to clip western islands
hipolygon = Polygon([(-160, 0), (-160, 90), (-120, 90), (-120, 0)])
hawaii_gdf = gdf[gdf.state == 'HI']

# Clip the Hawaii GeoDataFrame to the desired area
clipped_hawaii_gdf = hawaii_gdf.clip(hipolygon)

# Plot the clipped Hawaii GeoDataFrame using the 'value_determined_color' column fo
#color = hawaii_gdf['value_determined_color'].iloc[0] if not hawaii_gdf.empty else
clipped_hawaii_gdf.plot(ax=hiax, color='lightgrey', linewidth=0.8, edgecolor='0.8')

if variable == 'ag_proportion_0000':
    fig.savefig(os.path.join(os.getcwd(), './Outcomes/ag_legislation_1975_2021_0000
else:
    fig.savefig(os.path.join(os.getcwd(), './Outcomes/ag_legislation_1975_2021_0004

# bbox_inches="tight" keeps the vis from getting cut off at the edges in the saved
# dip is "dots per inch" and controls image quality.  Many scientific journals have
# https://stackoverflow.com/questions/16183462/saving-images-in-python-at-a-very-hi

plt.show()
```

Skip to main content

Chapter 5 Sentiment analysis

sentiment analysis

Chapter 6 Conclusion

Chapter 7 Appendix