instruction

Installation

You need to install the following libraries and dependencies to use this project:

- [Selenium](https://selenium-python.readthedocs.io/): Used for web automation.
- [Pandas](https://pandas.pydata.org/): Used for data manipulation.
- [IPython Widgets](https://ipywidgets.readthedocs.io/): Used for interactive widgets in IPython.
- [Tkinter](https://docs.python.org/3/library/tkinter.html): Used for creating GUI applications.
- [PyQt5](https://pypi.org/project/PyQt5/): Used for creating GUI applications.
- [Requests](https://docs.python-requests.org/en/master/): Used for making HTTP requests.
- [Beautiful Soup](https://www.crummy.com/software/BeautifulSoup/): Used for web scraping.

To use these libraries, first install it using pip:

```
(.venv) $ pip install selenium pandas ipywidgets requests beautifulsoup4
```

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.common.by import By
import pandas as pd
from selenium.webdriver.chrome.options import Options
import os
from selenium.webdriver import ActionChains
from selenium.common.exceptions import NoSuchElementException
from IPython.display import display, clear_output
import ipywidgets as widgets
import tkinter as tk
from tkinter import filedialog
from PyQt5.QtWidgets import QApplication, QMainWindow, QPushButton
import sys
import re
import zipfile #extract zip file
import requests
from bs4 import BeautifulSoup
```

You may also need to install specific drivers or additional software depending on your use case. For example, to use Selenium, you'll need to download and configure ChromeDriver (https://sites.google.com/chromium.org/driver/).

Setup downloading

The following part is the setup for the chromeDriver and downloading path:

- setting up the directory path for activating the browser

```
PATH = "/Users/frrookie521/Desktop/civil/cee495/cee495/Google\ Chrome\ for\ Testing.
```

- setting up the downloading path:

```
download_path = "/Users/frrookie521/Desktop/civil/cee495/automationtesting/"
```

Setting up the chromedriver environment

- `--disable-dev-shm-usage` is using for solving shared memory issue
- `download.default_directory` is setting up the downloading path
- `download.default_directory: False` is automated downloading files without user configuration
- `download.directory_upgrade: True` is ensures that Chrome will always use the downloading path
- `safebrowsing.enabled: True` is safe browsing

```
chrome_options = Options()
chrome_options.add_argument('--disable-dev-shm-usage')
prefs = {
    'download.default_directory': download_path,
    'download.prompt_for_download': False,
    'download.directory_upgrade': True,
    'safebrowsing.enabled': True
}
chrome_options.add_experimental_option('prefs', prefs)
```

- importing the excels that showing the lists of variables and their urls

```
excel_file = pd.read_excel('VariableSources_2.xlsx', header=None)
```

- initiate the google chrome as driver with the option set up from chrome_options:
- `driver.set_window_size`: set up the window size for the driver

```
driver = webdriver.Chrome(options=chrome_options)
driver.set_window_size(1500, 1000)
```

- `excel_file.iloc[1:, 2].tolist()`: extracting urls from the excel file
- `unique`: extracting the unique urls

```
urls = excel_file.iloc[1:, 2].tolist()
unique = set(url for url in urls if isinstance(url, str))
zipfiles = ['file1.zip', 'file2.zip']
```

Converting_TXT_file_to_CSV

text_to_csv(*input_filename*, *output_filename*)

Return a csv file that contains data from the txt file

Input_filename::	txt file that is opened by driver
Output_filename::	csv file that is created by the code itself

- using `headlines` to read all the lines from txt file and create a list. Each element will be the string the contain all the data in the line.
- After open the `output_filename` with write mode, the loop keep iterating for each line from the line list.
- `line.strip().split()` remove any leading whitespace for each line, and then splitting them into a list of individual string.
- `','.join(fields)` takes a list of individual and combine them into a single string as csv file format.

```
def text_to_csv(input_filename, output_filename):
    with open(input_filename, 'r') as txt: #open the file
        lines = txt.readlines()
    with open(output_filename, 'w') as csv_file:
        for line in lines:
            fields = line.strip().split()
            csv_line = ','.join(fields)
            csv_file.write(csv_line + '\n')
```

Interacting with URLs and filtering the data

In this codebase, we have designed a modular approach to interact with various URLs, each corresponding to a different data source. This modularity allows us to tailor our interactions to the specific requirements of each data source. For clarity, we've created distinct functions, each designed to handle a unique URL. These functions encompass the necessary HTML interactions, data extraction procedures, and any specific steps needed to retrieve data from each URL.

Common code for interacting button with each function:

- `driver.get()` is asking the driver to open the webpage base on the unique url
- `find_element` is the main part of the automation. It using for interacting with different kind of buttons.
- `time.sleep()` is using for delaying the following command. Sometimes, the webpage is not been fully loaded due to the network environment
- `EC.visibility_of_element_located()` is another kind of approach to dealing with slow loaded webpage. It will start the interaction until html is visible

The following are the functions name:

- **census_geographic**(*driver*, *unique*)

Open and download the table S0701 from the US Census.

  - After downloading the file, the code will extract the zip files. By setting up the maximum attempts, the code will extract once for each attempts until the files have been successfully extracted. The reason is that sometimes, the zip file may not be downloaded properly due to the internet.
  - The csv file name will be `f'ACSST5Y{year}.S0701-Data.csv'`, where the data is from 2017 to 2021. After doing the filtering, we save the data as `f'{year}.csv'`.

- **census_demographics**(*driver*, *unique*)

Open and download the table DP05 from the US Census.

  - The process is same as previous

- **tamu_umr_report**(*driver*, *unique*)

Open and download the 2021 Urban Mobility Report and Appendices.

  - The downloaded file will be excel table. The table is labeled inconsistently. Therefore the following work is cleaning and reorganizing the data.
  - we keep the necessary columns in the table and convert it to CSV file `'umr.csv'` and delete the previous excel spreadsheet.

- **census_data**(*driver*, *unique*)

Open and download the business applications by County.

- The downloaded data is excel spreadsheet. Convert it to CSV file version

- **census_survey**(*driver*, *unique*)

Open and download the County business patterns.

- download and extracting the data with the same approach as previous did.
- Using `'text_to_CSV'` function to convert the txt file into CSV

- **cnbc_business**(*driver*, *unique*)

Open and download the business ranking.

- downloading the business ranking data from the CNBC website. However, different with previous method, we let the code to search for the exacted year data's url. By doing this way, it will reduce the complication of interaction process. For the year 2009 and 2013, table's URL is different comparing to other years.
- For other years, we created URL format `f"https://www.cnbc.com/{year}/{month}/{day}/americas-top-states-for-business.html"`. Because these reports are been published between August and October, from 10th to 24th. Therefore, We set up the year is `year >= 2014`. Month is `in range(6,8)`, and the day is `in range(10,24)`. By trying different month and day, if the url is existing, then the `response.status_code` will be 200.
- After successfully login to the webpage, we try to extract the data from original html file and convert it to CSV file

- **realtor_research**(*driver*, *unique*)

Open and download the Residential data.

- After successfully downloading the CSV table, we clean and filter the data and save as `realtor.csv`.

- **taxfoundation**(*driver*, *unique*)

Open and download the State Business Tax Climate Index.

- we extract the data from the official website and convert it to the CSV file.

- **census_construction**(*driver*, *unique*)

Open and download the Building Permits Survey (BPS).

- **zillow**(*driver*, *unique*)

Open and download the Housing data.

- **climate_hazard**(*driver*, *unique*)

Open and download the National Risk Index.

- The process is same as working on census_demographics

- **census_DP02**(*driver*, *unique*)

Open and download the table DP02.

- The process is same as working on census_demographics

- **census_S2301**(*driver*, *unique*)

Open and download the table S2031.

  - The process is same as working on census_demographics

- **census_B25104**(*driver*, *unique*)

Open and download the table B25104.

  - The process is same as working on census_demographics

- **census_S0802**(*driver*, *unique*)

Open and download the table S0802.

  - The process is same as working on census_demographics

- **census_B14004**(*driver*, *unique*)

Open and download the table B14004.

  - The process is same as working on census_demographics

- **library**(*driver*, *unique*)

Open and download the binary data.

  - After `driver.get(url)`, we access the online source. Since the html is dynamically changing, we need to scroll down the webpage to find the html that we want by using `window.scrollBy(0, 1000);`
  - After downloading, we extract the zip file to get CSV file. However, after the extracting there will be a new folder that contains two different CSV files. What we do is trying to move these CSV files out from the new folder to our downloading path folder.

- **interstate**(*driver*, *unique*)

Open and download the interstate highway data.

- we setup a key for each html

```
years = range(2015, 2022)
download_xpath_years = {
   2015: '//*[@id="fullpage"]/table[3]/tbody/tr[39]/td[3]/a',
   2016: '//*[@id="fullpage"]/table[4]/tbody/tr[39]/td[3]/a',
   2017: '//*[@id="fullpage"]/table[3]/tbody/tr[39]/td[3]/a',
   2018: '//*[@id="fullpage"]/table[3]/tbody/tr[39]/td[3]/a',
   2019: '//*[@id="fullpage"]/table[3]/tbody/tr[39]/td[3]/a',
   2020: '//*[@id="fullpage"]/table[3]/tbody/tr[39]/td[3]/a',
   2021: '//*[@id="fullpage"]/table[3]/tbody/tr[33]/td[3]/a'
    }
```

- Based on each year URL and html, we will download xls files. Renaming the file as `f"{year}_interstate.xls`

- After downloading, we clean and filter the data, converting to `f'{year}_interstate.csv`

Target

- we setup notation for each unique URLs.

Check_target

we save the notation for each target. These notation names will be the function name which will be operated in previous code.

```
target_formats = list(target.keys())
```

Then, checking whether each url from the excel sheet `VariableSources_2.xlsx` matched with our target urls. If matched, the output will return True

```
return any(url.startswith(format) for format in target_formats)
```

We convert each key to an interaction function that can be operated automatically by the code.

```
for url in unique:
    if check_target(unique):
        interaction_function = target[url]
        interaction_function(driver, url)
```