```
import re
import time
from bs4 import BeautifulSoup
from selenium.common import NoSuchElementException
from selenium.webdriver.common.by import By
from selenium.webdriver.common.action_chains import ActionChains
from seleniumbase import Driver
import pandas as pd


def main():
    user_keywords = "ERR"
    user_location = "ERR"
    job_title = "ERR"
    job_location = "ERR"
    company_name = "ERR"
    query = "ERR"
    csv_file_path = "ERR"
    base_url = "ERR"
    apply_link = "ERR"
    search_url = "ERR"
    jobs_html = []
    date_posted = "ERR"
    jobs_list = []
    soup = "ERR"
    data_frame = []

    # Grab user search parameters
    user_keywords, user_location = general_check(lambda: get_user_input(), "Unable to grab
search parameters...")

    # Google
    do_google_helper(user_keywords, user_location, query, data_frame)

    # Indeed
    do_indeed_helper(user_keywords, user_location, query, data_frame)

    # Ziprecruiter
    do_career_builder_helper(user_keywords, user_location, query, data_frame)

    # Final CSV
    read_google = pd.read_csv('google_jobs_listings.csv', index_col="Job Title", na_values=0)
    read_indeed = pd.read_csv('indeed_job_listings.csv', index_col="Job Title", na_values=0)
    read_career_builder = pd.read_csv('career_builder_job_listings.csv', index_col="Job Title",
na_values=0)
    final_csv = pd.concat([read_career_builder, read_indeed,
read_google]).drop_duplicates(keep='first')
    final_csv_dropped_duplicates = final_csv[~final_csv.index.duplicated(keep='first')]
    final_csv_dropped_duplicates.to_csv('final_csv', index=False)

    print(final_csv_dropped_duplicates.to_string())



def do_career_builder_helper(user_keywords, user_location, query, data_frame):
    base_url_zip_recruiter =
'https://www.careerbuilder.com/jobs?company_request=false&company_name=&compa
ny_id=&keywords='
    search_url_career_builder = general_check(
        lambda: create_user_search_parameters_career_builder(user_keywords,
user_location, base_url_zip_recruiter,
                                                    query),
        "Unable to generate search...")
    soup = general_check(lambda: get_html_code_indeed(search_url_career_builder),
"Unable to load soup...")
    jobs_list = general_check(lambda: jobs_list_create_helper(soup, 'data-results-title dark-
blue-text b'),
                              "Unable to generate jobs list...")
    general_check(lambda: find_job_data_career_builder(soup, jobs_list, 'data-results-title
dark-blue-text b', 0, 'div'),
        "Unable to initialize job search...")
    general_check(lambda: find_job_data_career_builder(soup, jobs_list, 'data-details', 1,
'div'),
        "Unable to initialize job search...")
    general_check(lambda: find_job_data_career_builder(soup, jobs_list, 'data-details', 2,
'div'),
        "Unable to initialize job search...")
```

```
  general_check(lambda: find_job_data_career_builder(soup, jobs_list, 'data-results-
publish-time', 3, 'div'),
      "Unable to initialize job search...")

  csv_file_path = 'career_builder_job_listings.csv'
  convert_to_csv(jobs_list, csv_file_path, data_frame)



def do_indeed_helper(user_keywords, user_location, query, data_frame):
  base_url_indeed = "https://www.indeed.com/jobs"
  search_url_indeed = general_check(
      lambda: create_user_search_parameters_indeed(user_keywords, user_location,
base_url_indeed, query),
      "Unable to generate search...")
  soup = general_check(lambda: get_html_code_indeed(search_url_indeed), "Unable to
load soup...")
  jobs_list = general_check(lambda: jobs_list_create_helper(soup, 'css-pt3vth e37uo190'),
                    "Unable to generate jobs list...")
  general_check(lambda: find_job_data_indeed(soup, jobs_list, 'css-pt3vth e37uo190', 0,
'div'),
            "Unable to initialize job search...")
  general_check(lambda: find_job_data_indeed(soup, jobs_list, 'css-1h7lukg eu4oa1w0', 1,
'span'),
            "Unable to initialize job search...")
  general_check(lambda: find_job_data_indeed(soup, jobs_list, 'css-1restlb eu4oa1w0', 2,
'div'),
            "Unable to initialize job search...")
  general_check(lambda: find_job_data_indeed(soup, jobs_list, 'css-1yxm164 eu4oa1w0',
3, 'span'),
            "Unable to initialize job search...")


  csv_file_path = 'indeed_job_listings.csv'
  convert_to_csv(jobs_list, csv_file_path, data_frame)


def do_google_helper(user_keywords, user_location, query, data_frame):
  base_url_google = "https://www.google.com/search"
  search_url_google = general_check(
      lambda: create_user_search_parameters_google(user_keywords, user_location,
base_url_google, query),
      "Unable to generate search...")
  soup = general_check(lambda: get_html_code_google(search_url_google), "Unable to
load soup...")

  jobs_list = general_check(lambda: jobs_list_create_helper(soup, 'tNxQIb PUpOsf'),
"Unable to generate jobs list...")
  general_check(lambda: find_job_data_google(soup, jobs_list, 'tNxQIb PUpOsf', 0, 'div'),
            "Unable to initialize job search...")
  general_check(lambda: find_job_data_google(soup, jobs_list, 'wHYlTd MKCbgd a3jPc', 1,
'div'),
            "Unable to initialize job search...")
  general_check(lambda: find_job_data_google(soup, jobs_list, 'wHYlTd FqK3wc MKCbgd',
2, 'div'),
            "Unable to initialize job search...")
  general_check(lambda: find_job_data_google(soup, jobs_list, 'gmxZue', 3, 'span'),
            "Unable to initialize job search...")

  csv_file_path = 'google_jobs_listings.csv'
  convert_to_csv(jobs_list, csv_file_path, data_frame)


# Make a function that validates strings
def input_valid_str(input_check):
  check = False
  for char in input_check:
      if char.isalpha() or char == '+':
          pass
      else:
          check = True
  if check:
      return False
  else:
      return True
```

```
# Make a function that checks if a statement executes properly, throws specified error
statement otherwise
def general_check(statement, err_statement):
   bool_check = True
   try:
      check = statement()
   except:
      print(err_statement)
      bool_check = False

   if bool_check:
      return check



def get_user_input():
   print("Please enter any keyword that you would like with the spaces being replaced by +")
   print("\tExample: Data+Scientist, Computer+Science, etc..")
   user_keywords = input("\tEnter: ")

   while True:
      check = input_valid_str(user_keywords)
      if check:
         break
      else:
         user_keywords = input("\tInvalid Input...\n\tEnter: ")

   print("\nPlease enter any location that you would like with the spaces being replaced by
+")
   print("\tExample: Kearney+Nebraska, Manhattan+New+York, etc...")
   user_location = input("\tEnter: ")

   while True:
      check = input_valid_str(user_location)
      if check:
         break
      else:
         user_location = input("\tInvalid Input...\n\tEnter: ")
```

```
    return user_keywords, user_location


def create_user_search_parameters_google(user_keywords, user_location,
base_url_google, query):
    query = f"?q={user_keywords}+jobs+in+{user_location}&ibp=htl;jobs"
    search_url = base_url_google + query

    print("Search URL Google:", search_url)
    return search_url


def create_user_search_parameters_indeed(user_keywords, user_location,
base_url_indeed, query):
    query =
f"?q={user_keywords}+&l=+{user_location}&radius=100&from=searchOnHP&vjk=95f6d1a0
3732205d"
    search_url = base_url_indeed + query

    print("\nSearch URL Indeed:", search_url)
    return search_url


def create_user_search_parameters_career_builder(user_keywords, user_location,
base_url_zip_recruiter, query):
    query =
f"{user_keywords}+&location=+{user_location}+%2C+&pay=&emp=&cb_veterans=false&c
b_workhome=all&sort=date_desc"
    search_url = base_url_zip_recruiter + query

    print("\nSearch URL Zip Recruiter:", search_url)
    return search_url


def get_html_code_google(search_url):
    driver = general_check(lambda: Driver(browser="Chrome", uc=True, headless=False),
"Unable to load driver...")
```

```
    general_check(lambda: driver.get(search_url), "Unable to load webpage...")
    bottom_height = general_check(lambda: driver.execute_script("return
document.body.scrollHeight"),
                              "Unable to execute script...")
    while True:
        general_check(lambda: driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);"),
                      "Unable to execute script...")
        time.sleep(.5)
        new_height = general_check(lambda: driver.execute_script("return
document.body.scrollHeight"),
                                   "Unable to execute script...")
        if new_height == bottom_height:
            soup = general_check(lambda: BeautifulSoup(driver.page_source, 'html.parser'),
"Unable to parse webpage...")
            return soup
        bottom_height = new_height


def get_html_code_indeed(search_url):
    driver = general_check(lambda: Driver(browser="Chrome", uc=True, headless=False),
"Unable to load driver...")
    general_check(lambda: driver.get(search_url), "Unable to load webpage...")
    time.sleep(2)
    soup = general_check(lambda: BeautifulSoup(driver.page_source, 'html.parser'),
"Unable to parse webpage...")
    return soup


def get_html_code_career_builder(search_url):
    driver = general_check(lambda: Driver(browser="Chrome", uc=True, headless=False),
"Unable to load driver...")
    general_check(lambda: driver.get(search_url), "Unable to load webpage...")
    soup = general_check(lambda: BeautifulSoup(driver.page_source, 'html.parser'),
"Unable to parse webpage...")
    return soup
```

```
def jobs_list_create_helper(soup, class_name):
  job_cards = general_check(lambda: soup.find_all('div', class_=f'{class_name}'), "Unable
to find class name...")
  rows, cols = (len(job_cards), 4)
  jobs_list = [[0 for i in range(cols)] for j in range(rows)]
  return jobs_list


def find_job_data_google(soup, jobs_list, class_name, index, header):
  job_cards = general_check(lambda: soup.find_all(f'{header}', class_=f'{class_name}'),
               "Unable to find class name...")
  counter = 0
  if class_name == 'gmxZue':
    for every in job_cards:
      results = every.text.replace("ShareFacebookWhatsAppXEmailClick to copy linkShare
linkLink copied", "")
      temp = results.split('via')
      last_two = temp[1].split("      ")
      stripped = last_two[1].split('ago', 1)[0]
      if 'days' in last_two[1]:
        jobs_list[counter][index] = stripped + 'ago'
      counter += 1
    return jobs_list
  for equipment_type in job_cards:
    jobs_list[counter][index] = equipment_type.text
    counter += 1
  return jobs_list


def find_job_data_indeed(soup, jobs_list, class_name, index, header):
  job_cards = general_check(lambda: soup.find_all(f'{header}', class_=f'{class_name}'),
               "Unable to find class name...")
  counter = 0
  for equipment_type in job_cards:
    jobs_list[counter][index] = equipment_type.text
    if class_name == 'css-1restlb eu4oa1w0':
      new_text = equipment_type.text + " via. Indeed"
      jobs_list[counter][index] = new_text
```

```
        if 'Employer' in equipment_type.text:
            new_text = equipment_type.text.replace('Employer', '')
            jobs_list[counter][index] = new_text
        counter += 1
    return jobs_list


def find_job_data_career_builder(soup, jobs_list, class_name, index, header):
    job_cards = general_check(lambda: soup.find_all(f'{header}', class_=f'{class_name}'),
                              "Unable to find class name...")
    counter = 0
    for equipment_type in job_cards:
        if class_name == 'data-results-title dark-blue-text b':
            jobs_list[counter][index] = equipment_type.text
            counter += 1
        if class_name == 'data-results-publish-time':
            jobs_list[counter][index] = equipment_type.text
            counter += 1
        if 'Full-Time' in equipment_type.text:
            data_list = re.split('\n', equipment_type.text)
            jobs_list[counter][index] = data_list[index]
            counter += 1
        if 'Part-Time' in equipment_type.text:
            data_list = re.split('\n', equipment_type.text)
            jobs_list[counter][index] = data_list[index]
            counter += 1
        if 'Intern' in equipment_type.text:
            data_list = re.split('\n', equipment_type.text)
            jobs_list[counter][index] = data_list[index]
            counter += 1
    return jobs_list


def convert_to_csv(jobs_list, csv_file_path, data_frame):
    data_frame = pd.DataFrame(jobs_list, columns=['Job Title', 'Company', 'Location & via.',
'Date Posted'])
    data_frame.to_csv(f'{csv_file_path}', index=False)
    read_data_frame = pd.read_csv(f"{csv_file_path}",
                  usecols=["Company", "Job Title", "Location & via.", "Date Posted"],
                  index_col="Job Title", na_values=0)


if __name__ == '__main__':
    main()
```