```
import time
from bs4 import BeautifulSoup
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"
    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
    base_url_google = "https://www.google.com/search"
    search_url_google = general_check(
        lambda: create_user_search_parameters(user_keywords, user_location,
base_url_google, query),
        "Unable to generate search...")
    soup = general_check(lambda: get_html_code(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(soup, jobs_list, 'tNxQIb PUpOsf', 0, 'div'),
```

```
            "Unable to initialize job search...")
    general_check(lambda: find_job_data(soup, jobs_list, 'wHYlTd MKCbgd a3jPc', 1, 'div'),
            "Unable to initialize job search...")
    general_check(lambda: find_job_data(soup, jobs_list, 'wHYlTd FqK3wc MKCbgd', 2, 'div'),
            "Unable to initialize job search...")
    general_check(lambda: find_job_data(soup, jobs_list, 'gmxZue', 3, 'span'), "Unable to
initialize job search...")

    convert_to_csv(jobs_list, 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+Engineer, 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, 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(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:", search_url)
    return search_url


def get_html_code(search_url):
```

```
    driver = general_check(lambda: Driver(browser="Chrome", 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 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(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 convert_to_csv(jobs_list, data_frame):
    data_frame = pd.DataFrame(jobs_list[1:], columns=['Job Title', 'Company', 'Location &
via.', 'Date Posted'])
    data_frame.to_csv('google_jobs_listings.csv', index=False)
    read_data_frame = pd.read_csv("google_jobs_listings.csv",
                          usecols=["Company", "Job Title", "Location & via.", "Date Posted"],
                          index_col="Job Title", na_values=0)
    print(read_data_frame.to_string())


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