```
import time
from bs4 import BeautifulSoup
from seleniumbase import Driver


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(get_user_input(), "Unable to grab search
parameters...")

    # Google
    base_url_google = "https://www.google.com/search"
    search_url_google = general_check(create_user_search_parameters(user_keywords,
user_location, base_url_google, query), "Unable to generate search...")
    soup = general_check(get_html_code(search_url_google), "Unable to load soup...")

    jobs_list = jobs_list_create_helper(soup, 'tNxQIb PUpOsf')
    general_check(find_job_data(soup, jobs_list, 'tNxQIb PUpOsf', 0), "Unable to initialize job
search...")
    general_check(find_job_data(soup, jobs_list, 'wHYlTd MKCbgd a3jPc', 1), "Unable to
initialize job search...")
    general_check(find_job_data(soup, jobs_list, 'wHYlTd FqK3wc MKCbgd', 2), "Unable to
initialize job search...")
```

```
  general_check(find_job_data(soup, jobs_list, 'Yf9oye', 3), "Unable to initialize job search...")
  general_check(find_job_data(soup, jobs_list, 'nNzjpf-cS4Vcb-PvZLI-Ueh9jd-LgbsSe-Jyewjb-tlSJBe', 4), "Unable to initialize job search...")
  print(jobs_list)

# Make a function that validates strings
def input_valid_str(input_check):
  check = False
  char_check = ''
  for char in input_check:
    try:
      char_check = int(char)
      check = True
      break
    except:
      pass
  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):
  try:
    statement
    return statement
  except:
    print(err_statement)


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(Driver(browser="Chrome", headless=False), "Unable to load
driver...")
  general_check(driver.get(search_url), "Unable to load webpage...")
  bottom_height = driver.execute_script("return document.body.scrollHeight")
  while True:
    driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
    time.sleep(.5)
```

```
        new_height = driver.execute_script("return document.body.scrollHeight")
        if new_height == bottom_height:
            soup = general_check(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 = soup.find_all('div', class_=f'{class_name}')
    rows, cols = (len(job_cards), 5)
    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):
    job_cards = soup.find_all('div', class_=f'{class_name}')
    counter = 0
    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):
    return



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