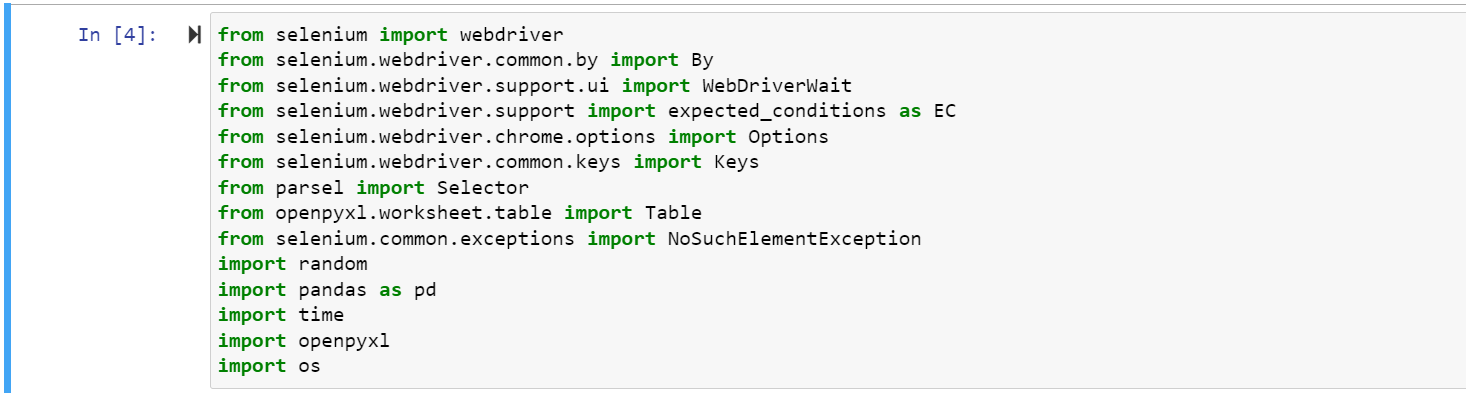
**LinkedIn Web Scraping**

The main purpose of this task was to use the Selenium library to automate the computer to open LinkedIn, iterate through every recommended job that is available, fetch the details of every job such as their name, location and workplace type and then store the details in an Excel Sheet. This is called Web Scraping.

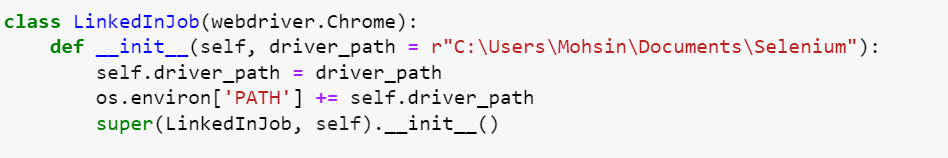
**Cell Block #1:**



These are the necessary libraries required for the automation and scraping to take place. The first library imported allows us to navigate through web pages, click on buttons, and perform website scraping. The **selenium.webdriver import By** library is used to locate HTML elements on a web page. The **WebDriverWait** library is used to wait for some time before a specific condition is fulfilled before execution of your code. The **import expected\_conditions as EC** is used with the **WebDriverWait** library and we use the shortform **EC**(expected condition) to wait for a specific condition before execution of our code. The **import Keys** library is used to send special keys or combinations to elements on a web page, such as input fields. Here we used the send keys library to enter username and password in the sign in page. The **from parsel import selector** library is used to extract data from HTML documents using XPATH and CSS selector methods.

The **import table** library allows us to work with tables in the Excel Worksheet. They provide various functionalities such as filtering, sorting, and formatting of sheets. The **NoSuchElementException** libraryis used to handle errors in situations where a specific element is not available during automation of that web page. The **import random** library provides us with various functionalities such as making random choices, we use it to stop the execution during the iteration of jobs so that all the data gets fetched and we don’t miss any scraping of data. We use the **pandas** library as well which is used for manipulation of data such as conversion of data into a DataFrame so that we can use them in an excel file. The **time** library is used as well. The **openpyxl** library is used to handle excel files such as opening them. We also use the **os** library to perform tasks such as file and directory operations.

**Cell Block #2:**

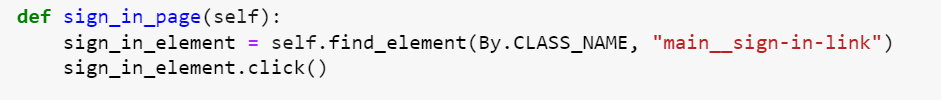


Here we start with the automation process first by creating a class first named **LinkedInJob** and passes **webdriver.Chrome()** as a parameter**.** It is used to launch the Chrome Browser. We then create a constructor and name it **\_\_init\_\_** and pass two parameters **self** and a driver path where our folder is created. The next line assigns the value of the driver path to the instance variable **self.driver\_path** This variable will then be accessible throughout the class.

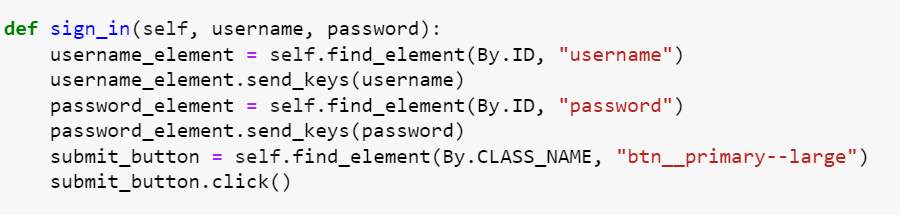
Then we modify the path by using the **os.environ** method and appending the driver path to it. It is done to locate the folder where the Chrome Webdriver is located. We then make a function named **super** which calls the parent class **LinkedInJob** and allows it to inherit the functionalities of the **Webdriver.Chrome()** parameter.



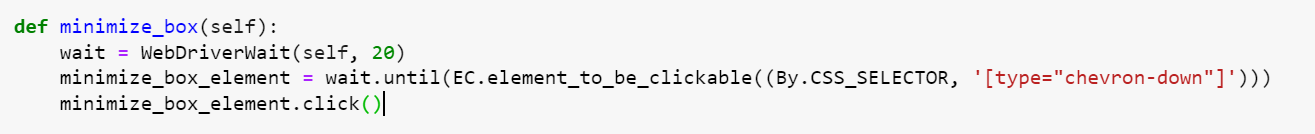
Next, we create a new function called **home\_page()** and pass the **self** as a parameter. We use the get function and pass the URL of the main page of LinkedIn webpage so that when we run this code it automatically opens the main page. This function is used to load the Home page of LinkedIn.



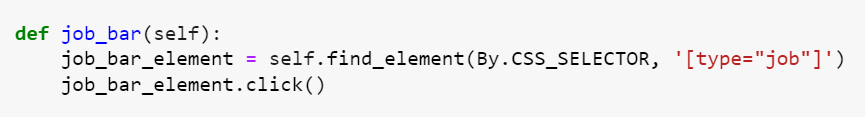
Next, we must click on the sign in option on the home page. For that we create a new function called **sign\_in\_in()**, we use the **self** parameter and the **find\_element** function to locate the HTML element which contains the parent class of the Sign In button. Once it is located, we use the **.click()** function so that it clicks on the Sign In option once it is located.



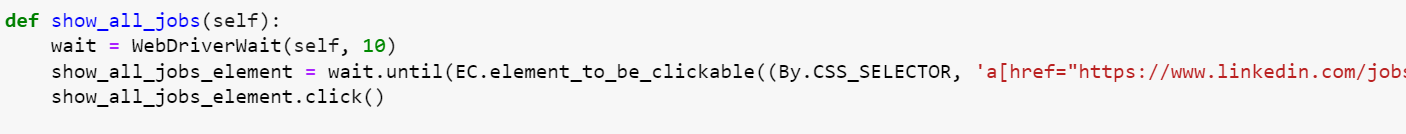
Now that we are on the Sign in page, we now must enter our account details so that we can login. For that we create a new function named **sign\_in()** and pass the parameters called **self**, **username**, and **password**. To get the username we create a variable and use the **.find\_element()** method to locate the HTML element of the username block. We locate it using the ID method since ID for every element is unique. Next, we modify the variable by using the **send\_keys** function that will allow us to enter our login details when we call the function. We apply the same process in the next two lines of code as well to get the password. Once the username and password are entered, we must click on the submit option. For that we use the **.find\_element()** method as well to locate the HTML element of the block. Once it is located, we use the **.click()** button and if the login details are correct, it redirects us to another page which contains our home page feed.



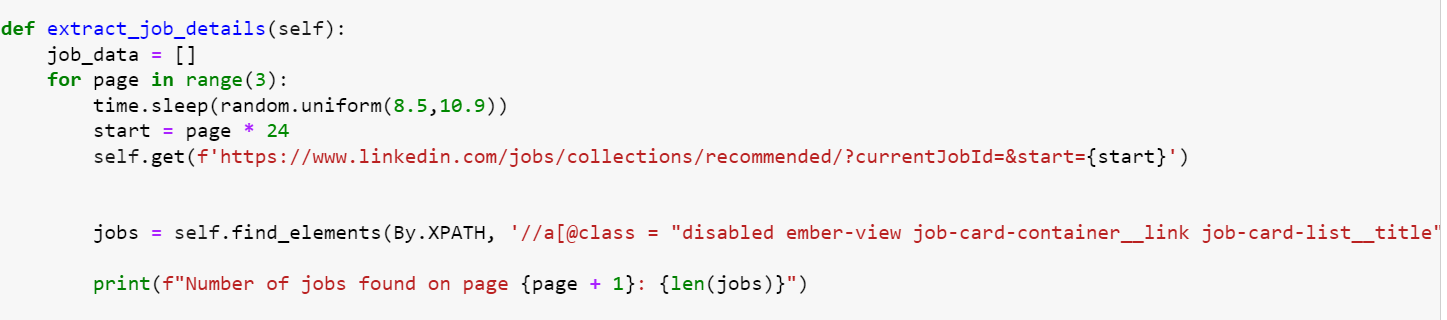
Now that we have logged in to our LinkedIn account, a message box appears on the right-hand side of our page. So, to close it so that the home page seems to be clearer to us we make a new function named as **minimize\_box()** and pass the same parameter. Here, we make a variable named as **wait** and we assign a function to it called **WebDriverWait(self, 20).** This means that when it redirects us to a new page, we will wait for a few seconds before executing any function because it depends on the network connection as to how long it will take for the page to be loaded fully. So, for it to not go through a timeout, we allow it to wait for a while and assign it to **wait.until(EC.element\_to\_be\_clickable()** function, where it waits until the page is fully loaded, then it starts the execution. Now it finds the HTML element in the block of chat head where we can minimize the chat head. Once, it is located we use the same **.click()** function to click on it and then it minimizes.



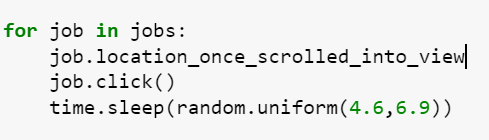
Now, we must go to the jobs section and the jobs section is right next to the navigation bar. We make a new function and pass a parameter. Then we find the HTML element of the block and we then use the same **.click()** function to click on it once it is located during automation.



After clicking on the Jobs section, it redirects us to a new page which contains a job of every kind. It contains a section which has jobs based on your profile(those are the recommended jobs), some are based on your location etc. Our task is to go to the recommended jobs section. For that we make a **wait** variable and assign it a function that waits for 10 seconds before performing any task. Then we make another variable called **show\_all\_jobs\_element()** and pass the previous variable in it along with the **.until(EC.element\_to\_be\_clickable()** function so that it waits until the page has been loaded and then we perform any execution. The show all recommended jobs page redirects you to a new page so to go to that page we use the **href** link from its HTML structure. In the end we use **.click()** method as usual to click on it once its located.



Now, the main task comes where we must iterate through every job available on the recommended jobs section page, then go to the next page if its available and iterate through every job as well. For that we make a new function called **extract\_job\_details()**. Next, we make a variable called **job\_data** and assign it to a list. This is where we will store data of every job we will iterate. Now we start a for loop that will iterate through every page available, and we write 3 in the range which means it will iterate through the first three pages only. The **time.sleep(random.uniform())** function is used to so that the loop takes a break in between when it is iterating through the pages so that the details of every job are saved, and no data is missed or lost. Next, we make a variable named start and assign it a multiplier of 24 because 24 jobs are available on every page and the link contains the word **page**, so we multiply the page by 24 so that it moves on to the next page when it has iterated over the entire job container. The we use **.get()** function to get the URL of the page of the recommended jobs and pass the **start** parameter so that it loads the first page. Then, we make a variable named jobs and here we store the HTML element of the entire container that contains all jobs. We find its parent class by using the **XPATH** method. Then, we use the print statement to print the total number of jobs available on every page.

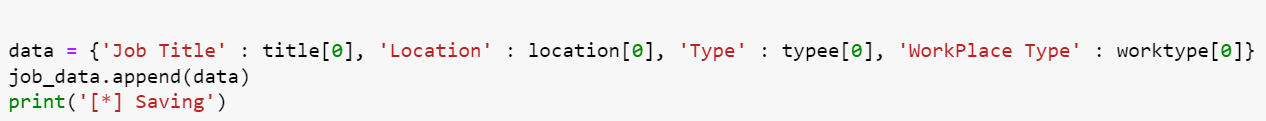


We make another for loop that creates a new variable job and iterates through the jobs variable which we have created before. The next line considers every job available in the job container and clicks on every job whenever they appear during iteration. During each iteration we take a small break of **(4.6,6.9)** seconds by using the **time.sleep()** function so that data doesn’t get mixed up or we don’t lose to iterate over any job that was available at that time.

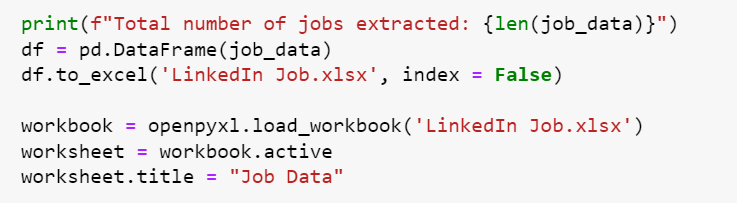


Now comes the extraction part where we must extract the details of every job that is available. For scraping of data, we use the **try** function where we scrape the name of the company that put forward the job. For that we use the XPATH method to locate the HTML element parent class. At the end of the line, we use the .text function so that we get the name of the company in text form. The **title = job\_title.split('\n')** line splits the **job\_title** string using the newline character **('\n')** as the separator and stores the resulting substrings in a list named title. This is done to extract individual components from the **job\_title** string if it contains newline-separated values.

The **except** function is used to raise an exception if the necessary HTML element is not available during the scraping of data of every job. We raise an error and store the null string in the variable during this case. This is repeated 3 more times to extract the location, type, and workplace type of every job available. This is called Web Scraping of data.

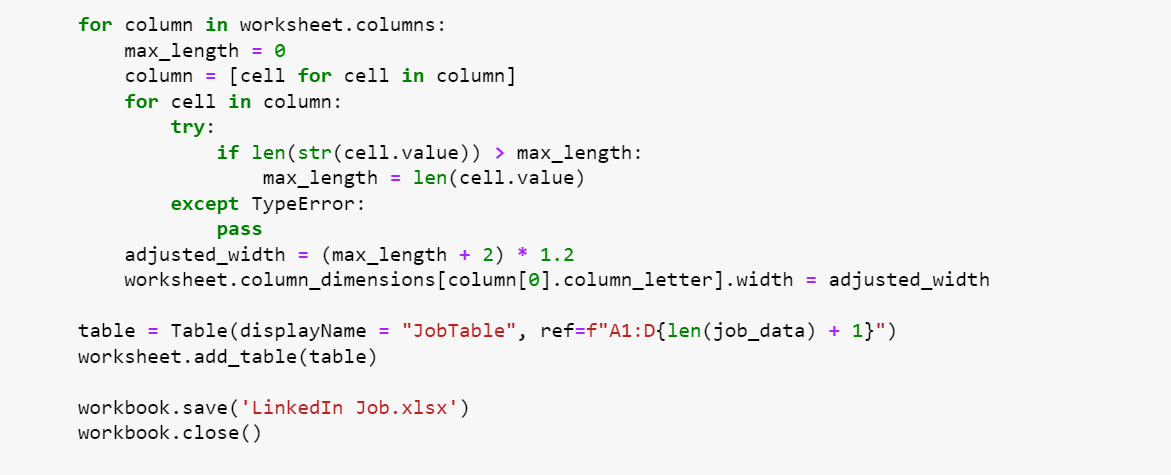


We then create a list of dictionaries called **data** where we store the variables and name their columns which we want them to be in the Excel Sheet. Then we append this dictionary into the **job\_data** list we created so that all the scraped data is stored in a list. To ensure that data is being saved one by one we write a print statement that shows us that data is being saved one by one.



Next, we use a print statement to print the total number of jobs extracted at the end of each page. Then we convert the **job\_data** variable into a DataFrame and store it in a new variable called **df**. Now we convert the DataFrame into an Excel Sheet using the **to\_excel()** method. We set the **index** as **False** as we don’t want to include the index number of each cell.

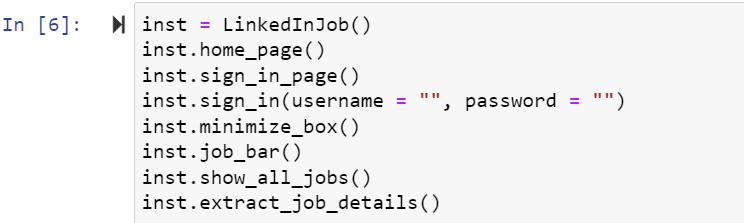
After that open the **workbook** using the **load\_workbook** function from the **openpyxl** library and set the name of the sheet we are working on. We set the name of the sheet as **“Job Data”**.



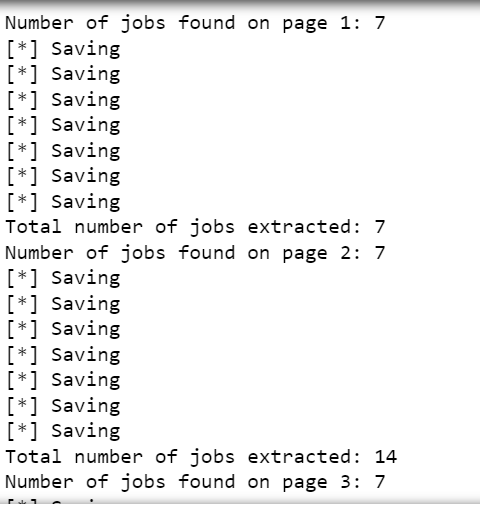
We now set the width of each column in the Excel File based on the maximum length of a particular string which we have scraped from the job details section. We use a for loop and pass the parameter as **worksheet.columns** which we want to perform the setting of width on. We first declare the maximum length of each column and for that we use the variable **max\_length** and set it to 0. After that we use a **try** function and use an **if** statement that if the length of the extracted data(by converting it into a string first) is greater than the **max\_length** of the columns, the **max\_length** variable is modified with the compared value. If it is not available, then we just use the **pass** function to get out of the loop. We make a new variable named as the **adjusted\_width** and pass the **max\_length** variable in it along with increasing the size of the character by +2 and multiplying it by 1.2 just to be on the safe side and it doesn’t fall short. We then set the dimensions of the columns and store it in the **adjusted\_width** variable.

After that we make a table in the Excel File for better presentation of data and the data looks easily readable to the reader. After that we save the **workbook** and close it using the **workbook.close()** method.

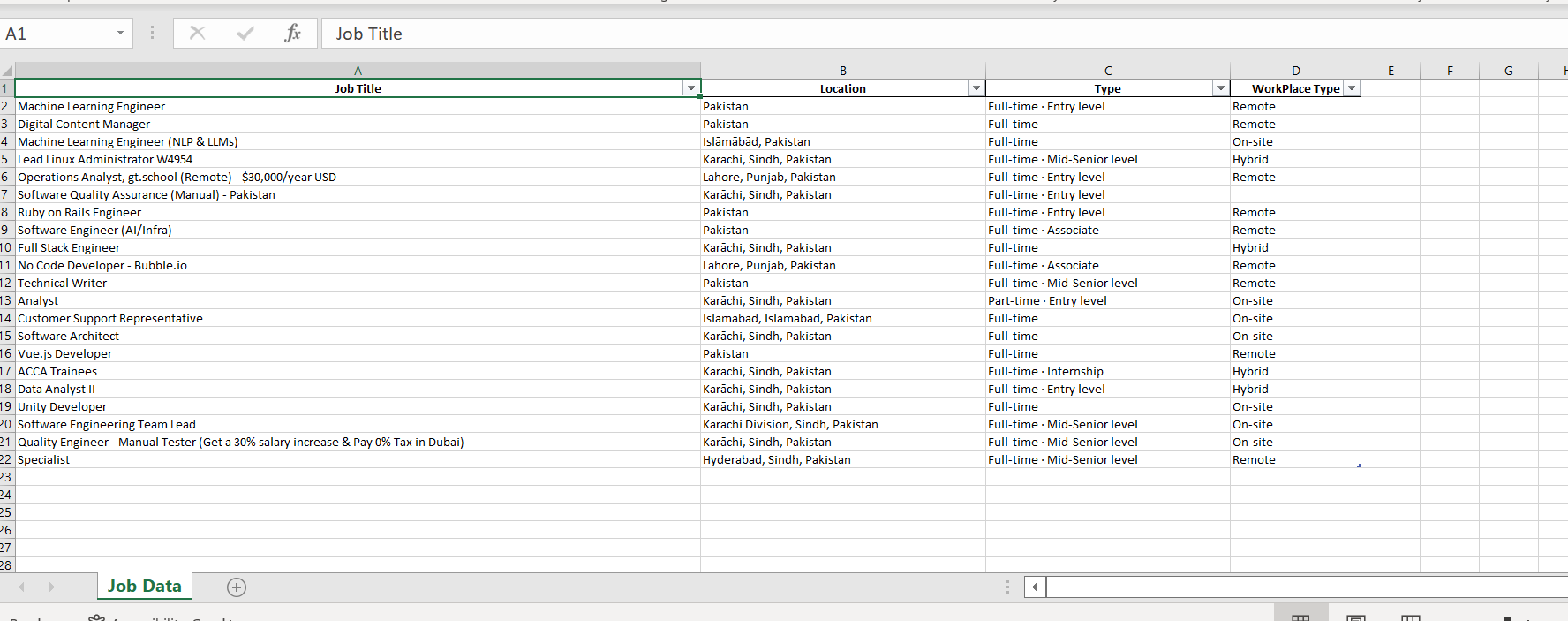
**Cell Block #3:**



In the final cell block we first make an instance of the Parent class and then call each of the functions one by one for the automation to take place.



This is what the terminal shows when it is iterating over each job available on every job. However, LinkedIn restricts us from scraping each job available at that time that’s why it only scrapes over 7 jobs per page.



This is how the Excel Sheet appears to be at the end of the automation process. It saves the data of each job available on the recommended page of the job section.