Upwork Talent Scraping Project Documentation

Submitted by:

Danish	2022-CS-25
Omar	2022-CS-43
Khurram	2022-CS-48

Submitted to:

Nazeef-ul-Haq

University of Engineering and Technology Lahore

30th October 2023

Contents

1 Introduction **2**
1.1 Project Overview . . . 2
1.2 Functionalities . . . 2
1.2.1 Data Scraping . . . 2
1.2.2 Sorting Algorithms . . . 2
1.2.3 Search Functionality . . . 3
1.2.4 Graphical User Interface (GUI) . . . 3

2 Project Objectives **4**
2.1 Data Scraping . . . 4
2.2 Sorting and Multi-level Sorting . . . 4
2.2.1 Selection Sort . . . 4
2.2.2 Insertion Sort . . . 5
2.2.3 Merge Sort . . . 5
2.2.4 Hybrid Merge Sort . . . 5
2.2.5 Bubble Sort . . . 5
2.2.6 Heap Sort . . . 5
2.2.7 Quick Sort . . . 5
2.3 Graphical User Interface (GUI) . . . 5

3 Problems Faced **8**
3.1 Challenges with Fiverr and Amazon . . . 8
3.2 Transition to Upwork . . . 8

4 Code **9**
4.1 Searching . . . 9
4.2 Sorting . . . 11
4.3 Scrapping . . . 17

Chapter 1

Introduction

1.1 Project Overview

The ”Project Name” is designed to collect data from Upwork, a popular freelancing platform. It retrieves information about freelancers, comprising one million entries with eight attributes, and offers essential features to manage and analyze this data.The project involves several key functionalities, including data scraping, sorting algorithms, and a user-friendly Graphical User Interface (GUI) implemented in Python using Qt Designer.

1.2 Functionalities

The project encompasses several critical functionalities, including:

1.2.1 Data Scraping

The system employs libraries such as Selenium, Pandas, and Beautiful Soup, along with the Chrome WebDriver, to scrape data from Upwork. It provides a pause and resume feature, allowing the user to collect data from multiple web pages.

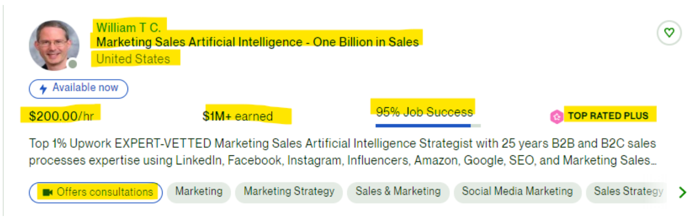
It retrieves information about freelancers, comprising one million entries with eight attributes:

Attributes	**Description**
Name	The name of the freelancer
Title	The job title or specialization of the freelancer
Job Success Score	The freelancer’s job success score on Upwork
Rates ($)	The rates or prices set by the freelancer for their services
Offer Consultations	Indicates whether the freelancer offers consultation services
Location	The geographical location of the freelancer
Total Earnings	The total earnings of the freelancer on Upwork
Special Rating Badges	Any special rating badges or distinctions earned by the freelancer

Table 1.1: Attributes of the Dataset

1.2.2 Sorting Algorithms

The project implements various sorting algorithms, such as selection sort, insertion sort, merge sort, and hybrid merge sort. Additionally, it includes multi-level sorting capabilities to sort data based on multiple attributes.


William T C.
Marketing Sales Artificial Intelligence - One Billion in Sales
United States

$200.00/hr $1M+ earned   95% Job Success   TOP RATED PLUS

Top 1% Upwork EXPERT-VETTED Marketing Sales Artificial Intelligence Strategist with 25 years B2B and B2C sales processes expertise using LinkedIn, Facebook, Instagram, Influencers, Amazon, Google, SEO, and Marketing Sales...

Figure 1.1: Upwork attributes

1.2.3 Search Functionality

The project offers a search feature that enables users to search for specific records within the scraped data efficiently.

1.2.4 Graphical User Interface (GUI)

A user-friendly GUI is created using Python and Qt Designer to provide a seamless user experience. This GUI makes it easy for users to interact with and control the scraping, sorting, and searching processes.

Chapter 2

Project Objectives

This project aims to achieve the following objectives:

2.1 Data Scraping

Efficiently collect data from Upwork freelancers using libraries such as Selenium, Pandas, and Beautiful Soup, along with the Chrome WebDriver. Implement a pause and resume feature to manage data collection from multiple web pages. Threading Optimization:

Implemented Threading:

To maximize scraping efficiency, threading was employed. This involved breaking down the data collection process into multiple concurrent tasks.

Utilized Multiple Threads:

We initiated two separate threads (t1 and t2) to simultaneously extract data from different sets of pages.

Parallelized Data Collection:

Each thread was responsible for fetching data from its designated range of pages. This parallel approach greatly accelerated the scraping process.

Enhanced Efficiency:

Threading significantly improved the project's efficiency, particularly when dealing with a substantial number of pages.

Daemonized Threads:

To ensure streamlined program execution, we designated the threads as daemon threads. This means they will automatically exit when the main program completes.

Handled Potential Issues:

In anticipation of possible challenges, we made provisions to mitigate data loss. This included dynamically changing the driver path and the CSV file name for each URL, thereby minimizing any adverse impact on the data collection process.

2.2 Sorting and Multi-level Sorting

Implement various sorting algorithms to analyze the collected data, including both ascending and descending order sorting. The sorting algorithms used in the project are:

2.2.1 Selection Sort

Selection sort works by repeatedly finding the minimum (or maximum) element from the unsorted part of the data and putting it at the beginning (or end) of the sorted portion. This process is repeated until the entire data is sorted.

2.2.2 Insertion Sort

Insertion sort builds the sorted portion of the data one item at a time. It takes each element and inserts it into its correct position within the sorted part of the data, shifting other elements if necessary.

2.2.3 Merge Sort

Merge sort divides the data into smaller, equally-sized sublists and recursively sorts them. Then, it merges these sorted sublists to produce a single sorted list. It is known for its stable, divide-and-conquer approach.

2.2.4 Hybrid Merge Sort

Hybrid merge sort combines the divide-and-conquer approach of merge sort with a more efficient sorting algorithm like insertion sort for smaller sublists. This hybrid approach can improve the overall performance of the sorting process.

2.2.5 Bubble Sort

Bubble sort repeatedly steps through the data, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until no swaps are needed, indicating that the data is sorted.

2.2.6 Heap Sort

Heap sort builds a binary heap from the data and repeatedly removes the maximum element from the heap, adding it to the sorted portion of the data. This process continues until the heap is empty.

2.2.7 Quick Sort

Quick sort selects a 'pivot' element from the data and partitions the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively.

2.3 Graphical User Interface (GUI)

Create a user-friendly GUI using Qt Designer for Python, allowing users to interact with and control the data scraping, sorting, and searching processes seamlessly. The entire project is coded in Python to provide a unified environment for development and execution.

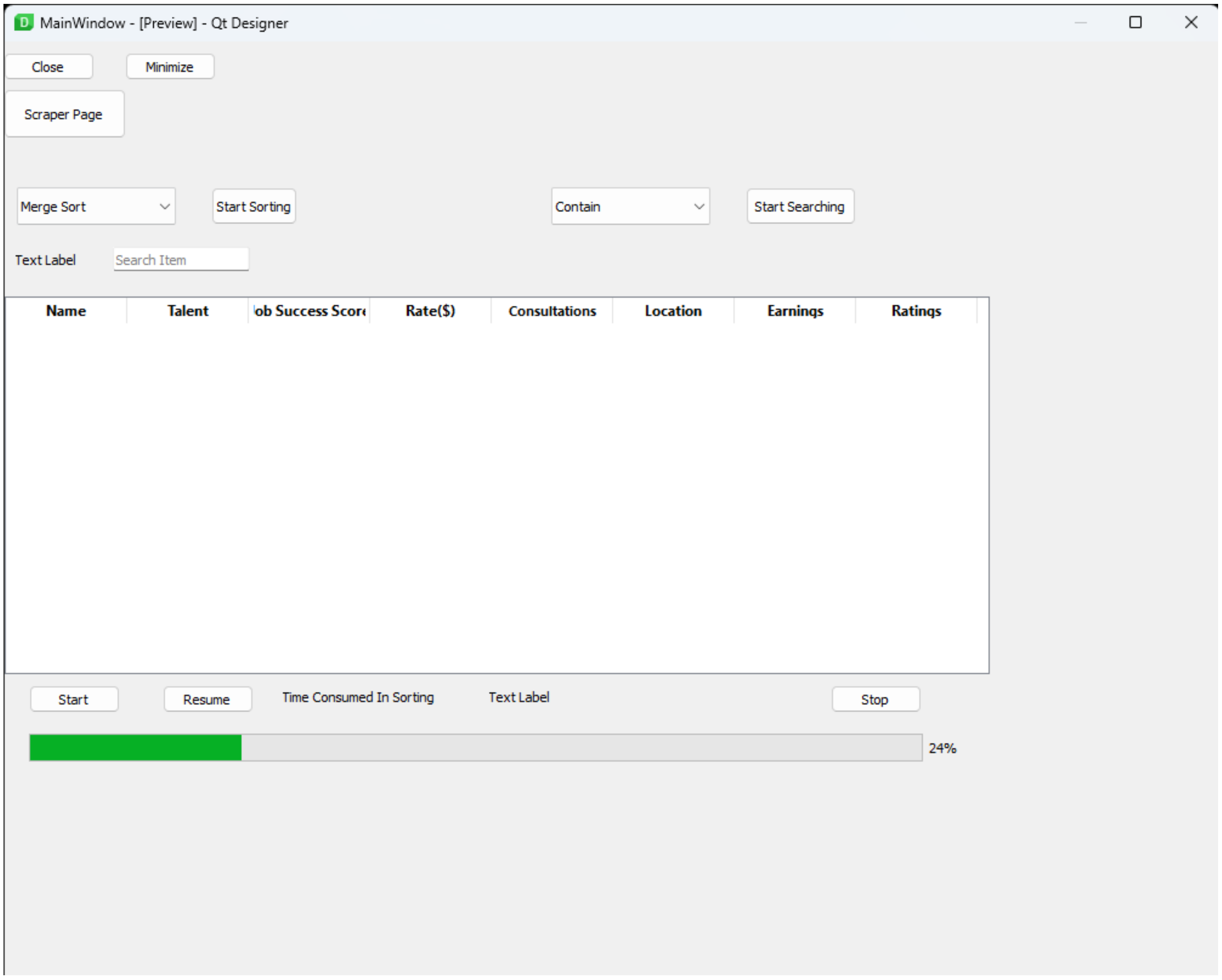


Figure 2.1: GUI for Sorting

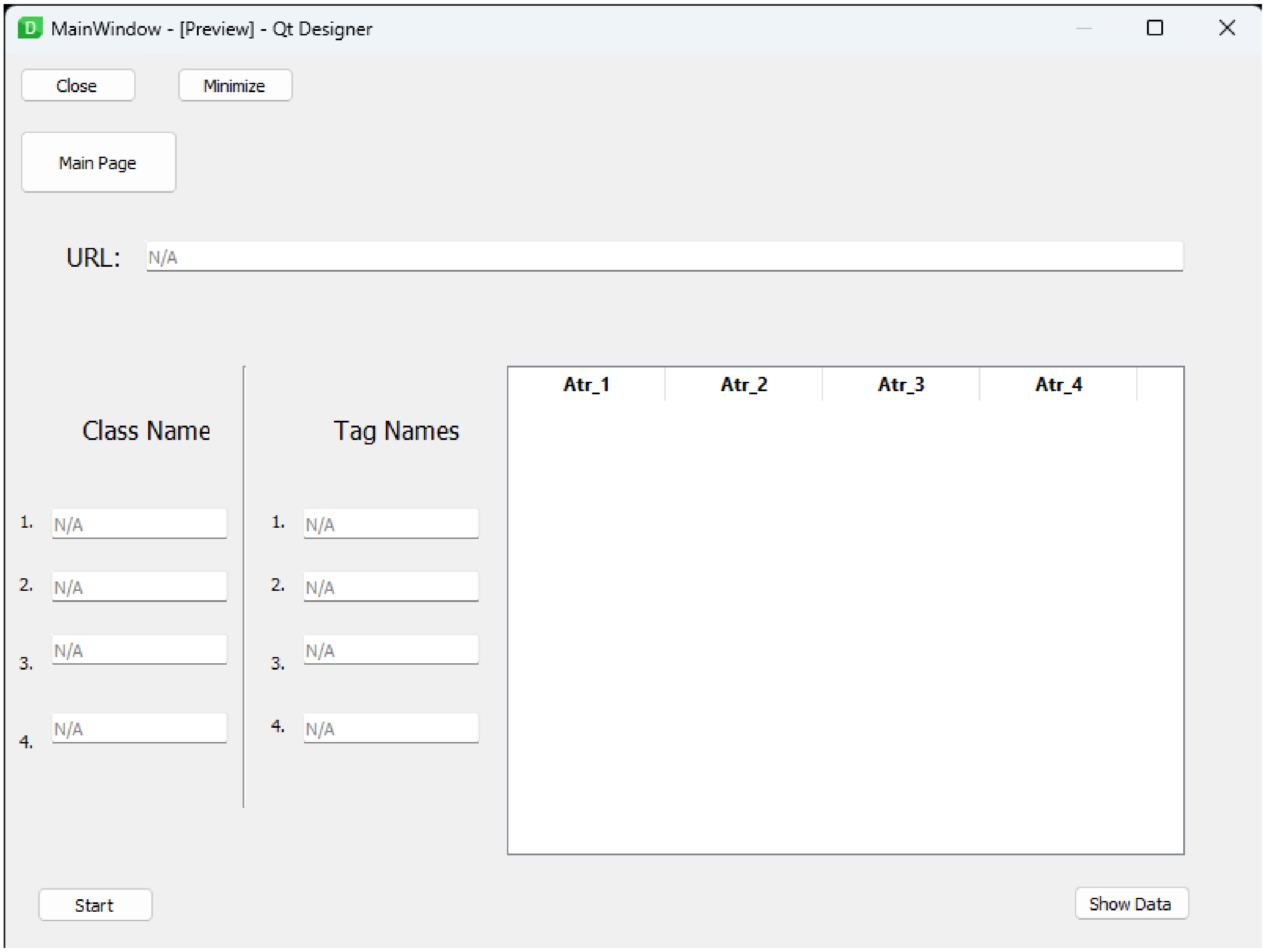


Figure 2.2: GUI for random URL Scrapping

Chapter 3

Problems Faced

During the course of the project, we encountered several challenges that affected our initial choice of scraping websites and the overall efficiency of the data scraping process. One significant problem we faced was related to the choice of the initial scraping source.

3.1 Challenges with Fiverr and Amazon

Our first choice for data scraping was Fiverr and later Amazon. However, both of these platforms presented considerable obstacles. The primary challenge was the presence of their captcha systems, which required users to complete a captcha verification at every page, making automated scraping difficult.

To address this issue, we attempted to implement an automatic captcha solver. However, this solution proved to be less effective and added significant overhead, leading to considerable time wastage. With the need to scrape a dataset of one million entries, the project's efficiency was severely impacted.

3.2 Transition to Upwork

Recognizing the difficulties associated with captcha challenges and the need for a more efficient data scraping process, we made the strategic decision to transition to Upwork as our primary scraping source. Although the scraping process remained relatively slow due to the size of the dataset, Upwork's platform did not present the same captcha problems as our previous choices. This transition allowed us to focus on the core functionality of data scraping and the subsequent sorting and analysis, ultimately contributing to a more streamlined and effective project.

Chapter 4

Code

4.1 Searching

```
def find(value, search):
    for i in range(len(search)):
        if (search[i] != value[i]):
            return False
    return True

def isfind(value, search):
    for i in range(len(value)):
        if (str(value[i]) == str(search[0])):
            if find(str(value[i:i + len(search)]), str(search)):
                return True
    return False

def search(rows, columnNo, search):
    n = len(rows)
    result = []
    for i in range(n - 1):
        if columnNo == 4: #float value
            value = str(rows[i][columnNo])
            if isfind(value, search):
                result.append(rows[i])
        elif str(rows[i][columnNo]) == str(search):
            result.append(rows[i])
    return result
import os
import csv
from ssl import Purpose
import pandas as pd
import time

def search (columnNo , search):
    import csv
    file = csv.reader(open('UpworkScrappedData.csv', 'r'))
    rows = [row for row in file]
    n = len(rows)
    list = []
    for i in range(n - 1):

            if (rows[i][columnNo]==search):
                list.append(rows[i])

    return list

path = "UpworkScrappedData.csv"
df = pd.read_csv(path)
```

```
import os
import csv
from ssl import Purpose
import pandas as pd
import time
rows = []
def find(value , search):
    for i in range(len(search)):


        if (search[i] != value[i]):
            return False
    return True

def isfind(value , search):
    for i in range (len(value)-1):
        if (value[i] == search[0]):


            if( (find(value[i:len(value)+1], search))==True):
                return True
    return False


def search (row ,columnNo , search):
    global rows
    rows = row
    n = len(rows)
    list = []
    for i in range(n - 1):
            if (i<len(rows)):

                if (isfind(rows[i][columnNo],search)==True):

                    list.append(rows[i])
                    rows.remove(rows[i])

    return list
def finalSearchFunction( rows , key):
    a=[]
    for i in range(8):
        a = a + search(rows,i, key)
    return a
```

4.2 Sorting

```
import csv
from multiprocessing.dummy import Array
import pandas as pd
import re

class SortingAlgo:
# selection sort for ascending order
    def SelectionSortForString(array,colNo):
        end = len(array)
        for i in range(end):
            min_idx = i
            min_value = array[i][colNo]
            for j in range(i + 1, end):
                if array[j][colNo] < min_value:
                    min_value = array[j][colNo]
                    min_idx = j

            if min_idx != i:
                temp = array[i]
                array[i] = array[min_idx]
                array[min_idx] = temp
        return array

# selection sort for descending order
    def SelectionSortForStringDescending(array,colNo):
        end = len(array)
        for i in range(end):
            min_idx = i
            min_value = array[i][colNo]
            for j in range(i + 1, end):
                if array[j][colNo] > min_value:
                    min_value = array[j][colNo]
                    min_idx = j

            if min_idx != i:
                temp = array[i]
                array[i] = array[min_idx]
                array[min_idx] = temp
        return array

    # insertion sort for ascending order implementation in project
    def InsertionSort(array,start,end,colNo):
        for x in range(start, end):
            temp = array[x]
            key = array[x][colNo]
            j = x - 1
            while j >= start and key < array[j][colNo]:
                array[j + 1] = array[j]
                j = j - 1
            array[j + 1] = temp
        return array

    # insertion sort for descending order implementation in project
    def InsertionSortForDescending(array,start,end,colNo):
        for x in range(start, end):
            temp = array[x]
            key = array[x][colNo]
            j = x - 1
            while j <= start and key > array[j][colNo]:
                array[j + 1] = array[j]
                j = j - 1
            array[j + 1] = temp
        return array

    # bubble sort for ascending order implementation in project
    def BubbleSort(array,start,end,colNo):
        for i in range(start,end):
            for j in range(0, len(array) - i - 1):
                if (array[j][colNo] > array[j + 1][colNo]):
                    temp = array[j]
                    array[j] = array[j+1]
                    array[j+1] = temp
        return array

    # bubble sort for descending order implementation in project
    def BubbleSortForDescending(array,start,end,colNo):
        for i in range(start,end):
            for j in range(0, len(array) - i - 1):
                if (array[j][colNo] < array[j + 1][colNo]):
                    temp = array[j]
                    array[j] = array[j+1]
                    array[j+1] = temp
        return array


# implementing Hybrid Merge Sort Algorithm for ascending order
def HybridMergeSort(array,start,end,colNo):

    minRun = end

    for i in range(start,end,minRun):
        SortingAlgo.InsertionSort(array, i, min((i+minRun),(end)),colNo)
        size = minRun
        while size < end:
            for j in range(start,end,size*2):
                mid = j + size - 1
                end = min((j + size * 2-1),(end-1))
                merged = hybMerge(colNo, left=array[j:mid+1][colNo], right=array[mid+1:end+1][colNo])

                array[j:j+len(merged)][colNo] = merged
        size *= 2
    return array

def hybMerge(colNo, left, right):
    i = 0
    j = 0
    l = []

    if(len(left) == 0):
        return right
    if(len(right) == 0):
        return left

    while i < len(left) and j < len(right):
        if left[i][colNo] <= right[j][colNo]:
            l.append(left[i])
            i += 1
        else:
            l.append(right[j])
            j += 1
    while i < len(left):
        l.append(left[i])
        i += 1
    while j < len(right):
        l.append(right[j])
        j += 1
    return l


# implementing Hybrid Merge Sort Algorithm for descending order
def HybridMergeSortForDescending(array,start,end,colNo):

    minRun = end

    for i in range(start,end,minRun):
        SortingAlgo.InsertionSortForDescending(array, i, min((i+minRun),(end)),colNo)
        size = minRun
        while size > end:
            for j in range(start,end,size*2):
                mid = j + size - 1
                end = min((j + size * 2-1),(end-1))
                merged = hybMerge(colNo, left=array[j:mid+1][colNo], right=array[mid+1:end+1][colNo])

                array[j:j+len(merged)][colNo] = merged
        size *= 2
    return array

def hybMerge(colNo, left, right):
    i = 0
    j = 0
    l = []

    if(len(left) == 0):
        return right
    if(len(right) == 0):
        return left

    while i < len(left) and j < len(right):
        if left[i][colNo] >= right[j][colNo]:
            l.append(left[i])
            i += 1
        else:
            l.append(right[j])
            j += 1
    while i < len(left):
        l.append(left[i])
        i += 1
    while j < len(right):
        l.append(right[j])
        j += 1
    return l


# implementing Heap Sort
def heapify(arr, n, i,colNo):
    # Find largest among root and children
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2

    if l < n and arr[i][colNo] < arr[l][colNo]:
        largest = l

    if r < n and arr[largest][colNo] < arr[r][colNo]:
        largest = r

    # If root is not largest, swap with largest and continue heapifying
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest,colNo)

def heapSort(arr,colNo):
    n = len(arr)

    # Build max heap
    for i in range(n//2, -1, -1):
        heapify(arr, n, i,colNo)

    for i in range(n-1, 0, -1):
        # Swap
        arr[i], arr[0] = arr[0], arr[i]

        # Heapify root element
        heapify(arr, i, 0,colNo)
    return arr

# implementing Heap Sort in descending order
def heapifyDescending(arr, n, i,colNo):
    # Find largest among root and children
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2

    if l < n and arr[i][colNo] > arr[l][colNo]:
        largest = l

    if r < n and arr[largest][colNo] > arr[r][colNo]:
        largest = r

    # If root is not largest, swap with largest and continue heapifying
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapifyDescending(arr, n, largest,colNo)

def heapSortDescending(arr,colNo):
    n = len(arr)

    # Build max heap
    for i in range(n//2, -1, -1):
        heapifyDescending(arr, n, i,colNo)

    for i in range(n-1, 0, -1):
        # Swap
        arr[i], arr[0] = arr[0], arr[i]
        # Heapify root element
        heapifyDescending(arr, i, 0,colNo)
    return arr


# Quick sort implementation in Ascending order
# function to find the partition position
def partition(array, low, high,colNo):

  # choose the rightmost element as pivot
  pivot = array[high][colNo]

  # pointer for greater element
  i = low - 1

  # traverse through all elements
  # compare each element with pivot
  for j in range(low, high):
    if array[j][colNo] <= pivot:
      # if element smaller than pivot is found
      # swap it with the greater element pointed by i
      i = i + 1

      # swapping element at i with element at j
      (array[i], array[j]) = (array[j], array[i])

  # swap the pivot element with the greater element specified by i
  (array[i + 1], array[high]) = (array[high], array[i + 1])

  # return the position from where partition is done
  return i + 1

# function for performing Quicksort
def quickSort(array, low, high,colNo):
  if low < high:
    # find pivot element such that
    pi = partition(array, low, high,colNo)

    # recursive call on the left of pivot
    quickSort(array, low, pi - 1,colNo)

    # recursive call on the right of pivot
    quickSort(array, pi + 1, high,colNo)
  return array


# Quick sort implementation in Descending order
# function to find the partition position
def partitionDescending(array, low, high,colNo):

  # choose the rightmost element as pivot
  pivot = array[high][colNo]

  # pointer for greater element
  i = low - 1

  # traverse through all elements
  # compare each element with pivot
  for j in range(low, high):
    if array[j][colNo] > pivot:
      # if element smaller than pivot is found
      # swap it with the greater element pointed by i
      i = i + 1

      # swapping element at i with element at j
      (array[i], array[j]) = (array[j], array[i])

  # swap the pivot element with the greater element specified by i
  (array[i + 1], array[high]) = (array[high], array[i + 1])

  # return the position from where partition is done
  return i + 1

# function for performing Quicksort
def quickSortDescending(array, low, high,colNo):
  if low < high:
    # find pivot element such that
    pi = partitionDescending(array, low, high,colNo)

    # recursive call on the left of pivot
    quickSortDescending(array, low, pi - 1,colNo)

    # recursive call on the right of pivot
    quickSortDescending(array, pi + 1, high,colNo)
  return array


# merge sort for project in Ascending Order
def MergeSort(array,start,end, colNo):
    if len(array) <= 1:
        return array
    p = start
    r = end
    q = len(array) // 2
    left_half = array[:q]
    right_half = array[q:]
    left = MergeSort(left_half,p,q,colNo)
    right = MergeSort(right_half,q+1,r,colNo)
    return Merge(left, right,colNo)

def Merge(left, right,colNo):
    i = 0
    j = 0
    l = []
    while i < len(left) and j < len(right):
        if left[i][colNo] <= right[j][colNo]:
            l.append(left[i])
            i += 1
        else:
            l.append(right[j])
            j += 1
    while i < len(left):
        l.append(left[i])
        i += 1
    while j < len(right):
        l.append(right[j])
        j += 1
    return l


# merge sort for project in Descending Order
def MergeSortDescending(array,start,end, colNo):
    if len(array) <= 1:
        return array
    p = start
    r = end
    q = len(array) // 2
    left_half = array[:q]
    right_half = array[q:]
    left = MergeSortDescending(left_half,p,q,colNo)
    right = MergeSortDescending(right_half,q+1,r,colNo)
    return MergeDescending(left, right,colNo)

def MergeDescending(left, right,colNo):
    i = 0
    j = 0
    l = []
    while i < len(left) and j < len(right):
        if left[i][colNo] > right[j][colNo]:
            l.append(left[i])
            i += 1
        else:
            l.append(right[j])
            j += 1
    while i < len(left):
        l.append(left[i])
        i += 1
    while j < len(right):
        l.append(right[j])
        j += 1
    return l
```

4.3 Scrapping

```
from selenium import webdriver
from bs4 import BeautifulSoup
import pandas as pd
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from webdriver_manager.chrome import ChromeDriverManager


def initialize_driver():
    driver_service = Service(ChromeDriverManager().install())
    options = webdriver.ChromeOptions()
    driver = webdriver.Chrome(service=driver_service, options=options)
    return driver

def returnSoup(url, className, driver, max_retries=3):
    print("Return Soup page...")
    for _ in range(max_retries):
        driver.get(url)
        try:
            # Wait for the products to load
            wait = WebDriverWait(driver, 10)
            wait.until(EC.presence_of_element_located((By.CLASS_NAME, className)))
            content = driver.page_source
            soup = BeautifulSoup(content, 'html.parser')
            return soup
        except Exception as e:
            return f"Error loading the page: {url}\nError details: {str(e)}"
    return f"Failed to load the page after {max_retries} retries: {url}"

def findData(soup, classList, tagList):
    print("Find Data page...")
    data_list = {
        'ATR1': [],
        'ATR2': [],
        'ATR3': [],
        'ATR4': [],
    }

    atr1 = soup.find_all(tagList[0], class_=classList[0])
    atr2 = soup.find_all(tagList[1], class_=classList[1])
    atr3 = soup.find_all(tagList[2], class_=classList[2])
    atr4 = soup.find_all(tagList[3], class_=classList[3])

    maxLength = max(len(atr1), len(atr2), len(atr3), len(atr4))

    for i in range(maxLength):
        if len(atr1) < maxLength:
            atr1.append('N/A')
        if len(atr2) < maxLength:
            atr2.append('N/A')
        if len(atr3) < maxLength:
            atr3.append('N/A')
        if len(atr4) < maxLength:
            atr4.append('N/A')

    for i in range(maxLength):
        atr1_value = atr1[i]
        atr2_value = atr2[i]
        atr3_value = atr3[i]
        atr4_value = atr4[i]

        if isinstance(atr1_value, str):
            atr1_value = ' '.join(atr1_value.split())
        else:
            atr1_value = ' '.join(atr1_value.text.split()) if atr1_value else 'N/A'

        if isinstance(atr2_value, str):
            atr2_value = ' '.join(atr2_value.split())
        else:
            atr2_value = ' '.join(atr2_value.text.split()) if atr2_value else 'N/A'

        if isinstance(atr3_value, str):
            atr3_value = ' '.join(atr3_value.split())
        else:
            atr3_value = ' '.join(atr3_value.text.split()) if atr3_value else 'N/A'
        if isinstance(atr4_value, str):
            atr4_value = ' '.join(atr4_value.split())
        else:
            atr4_value = ' '.join(atr4_value.text.split()) if atr4_value else 'N/A'


        data_list['ATR1'].append(atr1_value)
        data_list['ATR2'].append(atr2_value)
        data_list['ATR3'].append(atr3_value)
        data_list['ATR4'].append(atr4_value)

    df = pd.DataFrame(data_list)
    return df

def scrape_page(URL, classList, tagList):
    print("Scraping page...")
    url = URL
    className = 'row'
    driver = initialize_driver()
    result = returnSoup(url, className, driver)
    if isinstance(result, str):
        return result  # Return error message
    soup = result  # Continue with the BeautifulSoup object
    df = findData(soup, classList, tagList)
    df.to_csv('./ExtractedData/data.csv', mode='w', index=False, header=False)
    driver.quit()

def main(c1,c2,c3,c4, t1,t2,t3,t4, url):

    try:
        print("Scraping started...")
        classList = [c1, c2, c3, c4]
        tagList = [t1, t2, t3, t4]

        if not url.startswith('https://'):
            url = 'https://' + url

        url = url

        result = scrape_page(url, classList, tagList)
        if isinstance(result, str):
            print("Error incountered: ",result)  # Print error message
    except Exception as e:
        print(f"An error occurred: {str(e)}"
```