Web Scraping and Data Management Exam 1

Introduction:

Objective:

This document provides a detailed guide for web scraping and cleaning course data from the Essex Summer School website. The R script extracts course titles, URLs and course descriptions from the website and processes the course descriptions to remove unwanted characters and whitespace.

Prerequisites:

Software Requirements

- R version 4.4.1 or higher
- Required R packages `rvest`, `tidyverse`, `stringr`, `pacman` and `purrr`.

To install the required packages, use the following command: `install.packages(c("rvest", "tidyverse", "stringr", "pacman", "purrr"))`.

Data Requirements

- No specific data requirements, since this script directly scrapes the data from the URL below.

Procedure Steps:

Step 1: Load Required Libraries

- Use the `p_load` function from the `pacman` package to load multiple libraries simultaneously. This includes:
  - `rvest` for web scraping.
  - `tidyverse` for comprehensive data manipulation.
  - `stringr` for string operations.
  - `purrr` for functional programming tools.

```
pacman::p_load(rvest, tidyverse, stringr, purrr)
```

Step 2: Scrape Course Data

- Read and extract the course titles and URLs from the Essex Summer School course list page:

```
# URL of the Essex Summer School courses page
main_url <- "https://essexsummerschool.com/summer-school-facts/courses/ess-2024-course-list/"
```

```r
# Read the HTML content from the URL
main_page <- read_html(main_url)

# Extract course titles and URLs
courses <- main_page |> html_elements(".article a")

course_data <- tibble(
  Title = courses |> html_text(trim = TRUE),
  URL = courses |> html_attr("href")
)
```

1. **Assign the URL:** Assign the URL of the Essex Summer School 2024 course list page to the object `main_url`. This URL will be used for web scraping the course information.
2. **Fetch HTML Content:** Use the `read_html` function from the `rvest` package to read the HTML content of the webpage stored in `main_url`. This function retrieves the web page's content required for parsing and analysis. Store the result in the `main_page` object.
3. **Select Course Elements:** Use the `html_elements(".article a")` function to select all HTML elements that match the CSS selector `.article a`. This selector targets links (anchor tags `<a>`) within a section marked by the class `article`, which contains the course titles and their respective URLs.
4. **Create a Structured Data Table:** Construct a `tibble` called `course_data` to store the extracted information in a structured format:
   - **Title Column:** Populate this column by extracting the text content of the course links using the `html_text(trim = TRUE)` function. The `trim = TRUE` argument removes any leading or trailing whitespaces from the course titles.
   - **URL Column:** Populate this column by extracting the URLs from the `href` attributes of the anchor tags using the `html_attr("href")` function.

This `tibble` provides a clean and organised structure for further analysis or processing of the course information.

Step 3: Filter Unwanted URLs

- Filter out the unwanted URLs

1. **Define Unwanted Patterns:** Start by creating a vector of unwanted URL patterns using the `c()` function in Base R. Assign this vector to the object `unwanted_urls`. These patterns include email links (starting with `mailto:`) and a specific mailing list signup page URL.
2. **Filter Out Unwanted URLs:** Use the `filter`, `str_detect`, and `paste` functions to remove any rows from `course_data` where the URL matches the unwanted patterns.
   - `course_data |>`: The pipe operator `|>` directs the `course_data` tibble into the `filter()` function from the `tidyverse` package.
   - `str_detect(URL, ...)`: The `str_detect()` function from the `stringr` package checks each `URL` in `course_data` for a match against the unwanted patterns. It returns `TRUE` for matches and `FALSE` otherwise.

- `paste(unwanted_urls, collapse = "|")`: The `paste()` function combines all patterns in the `unwanted_urls` vector into a single string, separated by the `|` character. This creates a regular expression that will match any URL containing either `"^mailto:"` or `"https://essexsummerschool.com/mailing-list-signup/"`.

- `filter(!...)`: The `filter()` function retains rows where the condition within it is `TRUE`. By using the `!` operator, you negate the condition, so only rows where the `URL` does **not** match any of the unwanted patterns are kept.

3. **Update `course_data`:** The output is an updated `course_data` tibble that excludes any rows where the URL matches the patterns in `unwanted_urls`. This ensures that the dataset contains only valid course URLs, filtering out unwanted email links and the mailing list signup page.

```
# Filter out mailto link and mailing list
unwanted_urls <- c("^mailto:", "https://essexsummerschool.com/mailing-list-signup/")
course_data <- course_data |>
  filter(!str_detect(URL, paste(unwanted_urls, collapse = "|")))
```

- `^mailto:`: This is the regular expression pattern that matches any URL starting with `mailto:`
- `https://essexsummerschool.com/mailing-list-signup/`: This is a specific URL to exclude which is the mailing list signup page.

Step 4: Extract Course Descriptions

1. **Define the Extraction Function:** Start by creating a function called `get_description()` that retrieves course descriptions from each URL in the `course_data` object.

   - `get_description(URL)`: This function takes a URL as its input and returns the course description from the corresponding web page.

   - `read_html(URL)`: This function from the `rvest` package reads and downloads the HTML content from the specified `URL`, making it ready for parsing and analysis.

   - `html_elements("#mds-course-description")|>`: This function from the `rvest` package and selects and extracts all HTML elements from `course_page` that have the specified id. The |> pipe operator passes the course_page object (which contains the HTML content of a webpage) into the html_elements() function.

   - `html_text(trim = TRUE)`: This function extracts the text content from the selected HTML elements and trims any leading or trailing whitespaces, ensuring clean text output.

   - `return(description)`: Finally, this function from `Base R` returns the extracted course description.

2. **Apply the Function to `course_data`:** Update the `course_data` tibble by adding a new column called `Description`, which stores the course descriptions extracted from each URL.

   - `mutate()` function: This function, from the `tidyverse` package, adds or modifies columns in a tibble. Here, it's used to create a new `Description` column in `course_data`.

   - `map_chr(URL, get_description)`: The `map_chr()` function from the `purrr` package applies the `get_description()` function to each URL in the `course_data` tibble. It returns a character vector, which is appropriate for storing the course descriptions as text.

3. **Final Output:** The `course_data` tibble now has a `Description` column, where each row contains the course description fetched from its corresponding URL.

```
# Function to extract course descriptions from each URL
get_description <- function(URL) {
  course_page <- read_html(URL)
  description <- course_page |>
    html_elements("#mds-course-description") |>
    html_text(trim = TRUE)
  return(description)
}

# Add a column for descriptions by applying the get_description function
course_data <- course_data |>
  mutate(Description = map_chr(URL, get_description))
```

- `#mds-course-description`: This CSS selector targets the element with the `id` `mds-course-description`, where the course description is located.
- `map_chr()`: Efficiently applies the `get_description()` function to each URL and stores the resulting text in the `Description` column.

Step 5: Clean Up Course Descriptions

- Refine course descriptions by addressing encoding issues, replacing unwanted characters, and removing extra whitespace.

1. **Define the Cleaning Function:** Define the `clean_description` function to process course descriptions. This function will:
   - **Fix Encoding Issues:** Convert text encoding to UTF-8.
   - **Replace Unwanted Characters:** Substitute incorrect or unwanted characters with the correct ones.
   - **Remove Extra Whitespace:** Trim and clean up extra spaces in the descriptions.
2. **Apply the Cleaning Function:** To clean course descriptions in the `course_data` tibble:
   - **Mutate Operation:** Use the `mutate()` function from the `tidyverse` package to apply `clean_description()` to the `Description` column.

```
# Function to clean up the description
clean_description <- function(desc) {
  # Convert encoding if needed (UTF-8 is standard)
  desc <- iconv(desc, from = "latin1", to = "UTF-8", sub = "")

  # Replace unwanted characters
  desc <- str_replace_all(desc, c(
    "â€™" = "’",    # Replace with the correct apostrophe
    "â€“" = "-",    # Replace with the correct dash
    "â€œ" = "“",    # Replace with the correct opening double quotation mark
    "â€" = "”",      # Replace with the correct closing double quotation mark
    "â€˜" = "‘",     # Replace with the correct opening single quotation mark
    "[*]" = ""       # Remove asterisks
  ))

  # Remove extra whitespace
  desc <- str_squish(desc)  # str_squish() from stringr handles multiple spaces and trims

  return(desc)
}

# Apply the clean_description function to the Description column
course_data <- course_data %>%
  mutate(Description = clean_description(Description))
```

Function Breakdown:

- `iconv()` : Converts text encoding from `latin1` to `UTF-8` .
    - `from = "latin1"` : Specifies the source encoding.
    - `to = "UTF-8"` : Specifies the target encoding.
    - `sub = ""` : Replaces unconvertible characters with an empty string.
- `str_replace_all()` : Replaces specified patterns in the text.
    - `"â€™"` is replaced with `’` .
    - `"â€“"` is replaced with `-` .
    - `"â€œ"` is replaced with `“`.
    - `"â€"` is replaced with `”`.
    - `"â€˜"` is replaced with `‘`.
    - `"[*]"` is removed.
- `str_squish()` : Removes extra spaces and trims leading/trailing whitespace.
- `mutate()` : Updates the `Description` column with cleaned data.

Step 6: View and Save the Data

- To view the cleaned data use the `View()` function and to save the tibble `course_data` to a `.csv` file, use the `write_csv()` function from the `tidyverse` package.

```
# View the updated data
View(course_data)

# Optional: Save the data to a CSV file
write_csv(course_data, "essex_summer_school_courses.csv")
```

Conclusion:

This detailed guide provides instructions for web scraping and cleaning course data from the Essex Summer School website. This R script performs the following tasks: 1) Extraction: Retrieves course titles, URLs and descriptions from the website, and 2) Processing: Cleans course descriptions to remove unwanted characters and whitespace. The resulting cleaned data is available for review in the `course_data.csv` file.