Web Scraping MY

Handover Notes

Draft 1

Last Updated 15/7/2024

# Project Introduction

**Web scraping** is the process of automatically extracting data from a web page.

Every week, we scrape product data from the 14 biggest insurers (and several banks) in Malaysia’s websites. This allows us to be informed when new products are launched, and old products are taken down. The company’s goal is to be kept up to date with insurance product market trends.

Data points we scrape/keep track of in our master database (*all\_products.csv*):

1. *company*
2. *product\_type*
3. *product\_name*
4. *insurance\_type*

**(Life Insurance/General Insurance/Family Takaful/General Takaful)**

Depends on what the product’s company is registered as under [BNM](https://www.bnm.gov.my/regulations/fsp-directory).

Generally, company’s name will be shown on website. [Example](https://www.zurich.com.my/en/insurance-products/protection/for-my-health/zurich-icare-essential):

*A screenshot of a computer

Description automatically generated*

*A screenshot of a computer

Description automatically generated*

Hence, **Zurich iCare Essential** is a **Life Insurance** product.

1. *discovered\_place –* Whether we find product on bank or insurer website
2. *date\_discovered –* Date of the first time we encounter/scrape a product
3. *date\_retired –* Date we realise a product is missing (not on website anymore)

There are 3 tasks to be handed over:

1. Running the code weekly to update master database
2. Filling up product type form weekly
3. Extending the scraping to bank websites

Timeline:

1. The code is to be ran on **every Friday**.
2. If new products are discovered, the product type form is to be filled in with the info of those new products on the **following Monday to Wednesday**.
3. To be decided

# 1. Running the code weekly to update the master database

## 1.1 Code Structure

The main technologies used for this project are the R language and the [rvest](https://rvest.tidyverse.org/articles/rvest.html) library.

The code is written such that each website has its own *.R* file.

A screenshot of a computer

Description automatically generated

The 2 code files to be run weekly are *run.R* and *compare.R*, in that order.

*run.R* automatically runs the .R files in the code folder and compiles the results.

*compare.R* compares this weeks’ results with last week's results, detects new and retired products, and updates the database accordingly.

## 1.2 Running *run.R*

In RStudio or VSCode, open *Web Scraping* as home directory.

Open *run.R* and solve the Zurich captcha per the 2nd comment.



Then, run *run.R* (CTRL+SHIFT+ENTER)

If all goes well, a folder *Results\_MONTH\_YEAR\_WEEK* will be created in *Web Scraping,* containing csv files with results of all companies scraped and a compilation csv.

## 1.3 Troubleshooting and maintaining *run.R*

Sometimes, one or more of the code files will not scrape properly and *run.R* gives you an error message. (FWD has been having issues like this recently)

Most of the time, this can be resolved by manually re-running the failed code file and placing its results csv into the *Results\_MONTH\_YEAR\_WEEK* folder.

Note: avoid re-running code that has already scraped successfully. This wastes time and may cause you to be flagged by the website’s anti-bot measures.

In cases where this doesn’t resolve the issue, most likely the website has changed, and the code needs to be adjusted accordingly. Inform Dyson if you need more time to fix the code.

It’s normal for *run.R* to take up to 15 minutes to finish running. Though if it’s stuck on a single website for more than 5 minutes, likely something is wrong.

Can ask Daniel if need help with code.

## 1.4 Running *compare.R*

Once compilation.csv has been produced for this week, you should run *compare.R*

In RStudio or VSCode, open *Web Scraping* as home directory. Then, run *compare.R* (CTRL+SHIFT+ENTER)

compare.R has a few outputs:

1. Produces a backup of the all\_products.csv in the Backup folder. If there’s any error in the results and updating the master database, you can always restore a backup.
2. Produces a report of new and retired products in the Reports folder. This lets you see if the changes made are correct.
3. Updates the master database with new and retired products.

It’s also a good idea to check if the changes are actually accurate, e.g. google retired products to see if they can still be found.

## 1.5 Manual updates to master database

Check the report to see if any manual adjustments need to be made to the master database. Here are some common things that need to be adjusted (not exhaustive):

1. **Incomplete or inaccurate *product\_type***. For some websites the code cannot accurately determine *product\_type*. So, I put placeholders like ‘TBD’, ‘Savings/Investment’ or ‘Medical/CI/H&P’. Example:



Need to check the website or products brochures for these products to determine whether they are ‘Savings’ or ‘Investment’, then update their entry in *all\_product.csv* accordingly.

1. **Renamed products.** Occasionally websites change the way the product is presented. For example:
   1. ‘GREAT SmartVantage’ to ‘Great SmartVantage’
   2. ‘FWD Max Wealth’ to ‘Max Wealth’
   3. ‘Maxi-Wealth’ to ‘Maxi Wealth’

Code would treat ‘GREAT SmartVantage’ as a retired product and ‘‘Great SmartVantage’ as a new product, even though we can tell they are likely the same product.

Whenever you see 2 similar-sounding product names, one in retired products and another in new products, it’s a good idea to double check if they are the same product. One way to do this is to see if this week’s and last week’s product\_description are the same.

Once you confirm that the product is renamed, remove the ‘new product’ in *all\_products.csv* and change the status of the ‘old product’ from ‘Retired’ to ‘Existing’.

## 1.6 Update tracking sheet

Once the master database has been updated for the week, proceed to updating the tracking sheet at *Nicholas Actuarial Solutions Sdn Bhd\Intern - General\Product\Web Scraper Tracking.xlsx*

Inform Dyson of any new/retired products

# 2 Filling up product type form weekly

For all new products found each week, download its brochure into the Brochures&Flyers folder.

The product type form is found at

***Intern - General\Alistair\Web Scraping\Product Type\Product Type MY.xlsx***

Details of any new products are to be filled into this form.

The **ReadMe** tab of the spreadsheet contains comprehensive instructions on how to fill it in, which I won’t repeat in this document.

Additional Notes/Tips:

1. All relevant product information should be filled in. If there is no suitable row to put it in, you may create a new one.
2. Fill in the details in a succinct way. Avoid lengthy paragraphs if they are not necessary. Avoid copy-pasting from brochure, paraphrasing to shorten where necessary (without reducing any important details).
3. Make sure the info is filled up neatly. E.g. consistent spacing, font formatting, line borders etc.

All of the above improves readability and enables comparisons to be made more easily.

You may refer to ***Product Type Sample.xlsx***, or any prior filled-in products for reference.

Seek out Dyson if you need any clarification while filling up the form.

Once the product type form has been filled in for the week, proceed to updating the tracking sheet at *Nicholas Actuarial Solutions Sdn Bhd\Intern - General\Product\Web Scraper Tracking.xlsx*

# 3 Extending scraping to bank websites

There is a list of banks that we are interested in scraping from.

A screenshot of a computer

Description automatically generated

***Intern - General\Product\Malaysia Bank List.xlsx***

Need to write code for remaining websites.

Should produce results with same format and data points as other banks, see cimb.csv or publicbank.csv

Integrate new code into *run.R* and *compare.R*

You may refer to the following for reference:

1. Scraping code of existing insurer/bank companies. Similar patterns are techniques tend to be used across multiple websites.
2. Documentation PowerPoint file – can help understand how existing companies are being scraped (not updated for banks)
3. [rvest website](https://rvest.tidyverse.org/articles/rvest.html) and [rvest documentation](https://cran.r-project.org/web/packages/rvest/rvest.pdf) – to learn rvest
4. ChatGPT – helpful if there’s something specific you need to do in base R

Web Scraping Tips:

1. Use read\_html\_live() rather than read\_html() for dynamically generated websites