Actividad Evaluable 2

Descripción

MÓDULO	**Máster en Gestión de la Ciberseguridad**
ASIGNATURA	**Data Driven Security**
Fecha Límite de Entrega	20 de enero de 2025, a las 23:59
Puntos	25% de la Nota Total. **Mínimo se requiere un 5 para poder ir al examen final.**
Carácter	Grupo (max 2 personas)

Enunciado:

En esta actividad se planteará una serie de preguntas relacionadas con los temas Datos Elegantes y Gráficos y Bots Crawlers y Scrapping

El estudiante debe responder a tales preguntas antes de la fecha límite de entrega. Se considerará tanto la corrección de las soluciones como su presentación.

La entrega debe incluir el documento RMarkdown (.Rmd) y el documento renderizado en formato HTML (opcional incluir el PDF). Para la entrega, se espera que el contenido este disponible a través de un repositorio de código público creado en Github específicamente para esta actividad, antes de la fecha límite de entrega.

Se considerará tanto la corrección de las soluciones como su presentación y el código utilizado para la obtención de los resultados.

Parte de esta actividad implica ejecutar código R. Éste debe poderse ejecutar directamente sobre un terminal nuevo en R o en RStudio. El código es imprescindible para la corrección del ejercicio.

Las entregas tardías serán marcadas como "tarde", y pueden NO ser evaluadas.

1. Datos Elegantes + Análisis de Datos con Web Scrapping

Pregunta 1:

Queremos programar un programa de tipo web scrapping con el que podamos obtener una página web, mediante su URL, y poder analizar su contenido HTML con tal de extraer datos e información específica.

Nuestro programa ha de ser capaz de cumplir con los siguientes pasos:

1. **Descargar la página web de la URL indicada, y almacenarlo en un formato de R apto para ser tratado.**
   El primer paso para realizar tareas de *crawling* y *scraping* es poder descargar los datos de la web. Para esto usaremos la capacidad de R y de sus librerías (httr y XML) para descargar webs y almacenarlas en variables que podamos convertir en un formato fácil de analizar (p.e. de HTML a XML).

   Consejos:
   - *Podemos descargar páginas y contenido web con las funciones del paquete HTTR usando la función* `GET()`.
   - *Como la página va a estar en HTML, podemos usar las funciones del paquete XML. Por ejemplo, para pasar de HTML a XML podemos usar la función* `htmlParse()`.
   - *Si no tenéis instaladas las librerías, podéis usar la función* `install.packages()` *para ello. En Ubuntu o MacOS necesitaréis instalar previamente "libxml2-dev".*

2. **Analizar el contenido de la web, buscando el título de la página (que en HTML se etiqueta como "**title**").**
   En las cabeceras web encontramos información como el título, los ficheros de estilo visual, y meta-información como el nombre del autor de la página, una descripción de esta, el tipo de codificación de esta, o palabras clave que indican qué tipo de información contiene la página. Una vez descargada la página, y convertida a un formato analizable (como XML), buscaremos los elementos de tipo "title". P.e. "`<title>Titulo de Página</title>`".

   Pistas:
   - Ahora podemos usar las funciones xpathSApply() para buscar los valores de la página que deseemos.

3. **Analizar el contenido de la web, buscando todos los enlaces (que en HTML se etiquetan como "a"), buscando el texto del enlace, así como la URL.**
   Vamos a extraer, usando las funciones de búsqueda XML, todos los enlaces que salen de esta página con tal de listarlos y poder descargarlas más tarde. Sabemos que estos son elementos de tipo "`<a>`", que tienen el atributo "`href`" para indicar la URL del enlace. P.e. "`<a href = 'enlace'>Texto del Enlace</a>`". Del enlace nos quedaremos con la URL de destino y con el valor del enlace (texto del enlace).

   Pistas:
   - *Otra vez, usando* `xpathSApply()` *podemos buscar los enlaces, en los que nos interesa el texto (valor del enlace) y la URL a la que apunta (atributo "href").*
   - *La función* `xpathSApply()` *actua como* `sapply()`*, simplificando los resultados. Si el resultado es un vector, retornará un vector, pero si el resultado contiene valores "*`NULL`*", en vez de crear un vector creará una lista, para que no se pierdan los valores NULL (en un vector los NULL desaparecen). Comprobad el retorno de* `xpathSApply`*, y si hay NULLs tratadlos (p.e. con la función* `is.null()` *para encontrarlos:*
     "`valores_nulos <- sapply(lista, is.null)`*", y asignarles un valor por defecto como* `NA`*: "*`lista[valores_nulos] <- NA`*" y convertid la lista en un vector con la función* `unlist()`*.*

4. **Generar una tabla con cada enlace encontrado, indicando el texto que acompaña el enlace, y el número de veces que aparece un enlace con ese mismo objetivo.**
   En este paso nos interesa reunir los datos obtenidos en el anterior paso. Tendremos que comprobar, para cada enlace, cuantas veces aparece.

   Pista:
   - *Usando los enlaces encontrados, y gracias a la función* `table()`*, podemos hacer recuentos de elementos en vectores, matrices y data.frames.*

5. **Para cada enlace, seguirlo e indicar si está activo (podemos usar el código de status HTTP al hacer una petición a esa URL).**
   En este paso podemos usar la función HEAD de la librería "`httr`", que en vez de descargarse la página como haría GET, solo consultamos los atributos de la página o fichero destino.
   HEAD nos retorna una lista de atributos, y de entre estos hay uno llamado "`header`" que contiene más atributos sobre la página buscada. Si seguimos podemos encontrar el "`status_code`" en "`resultado$status_code`". El "`status_code`" nos indica el resultado de la petición de página o fichero. Este código puede indicar que la petición ha sido correcta (200), que no se ha encontrado (404), que el acceso está restringido (403), etc.

- Tened en cuenta que hay enlaces con la URL relativa, con forma "/xxxxxx/xxxxx/a.html". En este caso, podemos indicarle como "handle" el dominio de la página que estamos tratando, o añadirle el dominio a la URL con la función "paste".
- Tened en cuenta que puede haber enlaces externos con la URL absoluta, con forma "http://xxxxxx/xxxx/a.html" (o https), que los trataremos directamente.
- Tened en cuenta que puede haber enlaces que apunten a subdominios distintos, con forma "//subdominio/xxxx/xxxx/a.html". En este caso podemos adjuntarle el prefijo "https:" delante, convirtiendo la URL en absoluta.
- Tened en cuenta URLS internas con tags, como por ejemplo "#search-p". Estos apuntan a la misma página en la que estamos, pero diferente altura de página. Equivale a acceder a la URL relativa de la misma página en la que estamos.

Es recomendado poner un tiempo de espera entre petición y petición de pocos segundos (comando "`Sys.sleep`"), para evitar ser "baneados" por el servidor. Para poder examinar las URLs podemos usar expresiones regulares, funciones como "grep", o mirar si en los primeros caracteres de la URL encontramos "//" o "http". Para tratar las URLs podemos usar la ayuda de la función "paste", para manipular cadenas de caracteres y poder añadir prefijos a las URLs si fuera necesario.

Pistas:

- *Aquí se tendrá que iterar sobre cada enlace encontrado, comprobando los atributos obtenidos por la función head(). Sabemos que el retorno de HEAD es una lista/objeto con el atributo "status_code"*
- *Según la dirección del enlace es absoluta o relativa, habrá que usar un "handle" o no a la hora de hacer HEAD. Podemos usar grep() para buscar si existe la cadena de texto "http" al principio de la cadena de caracteres (u otras funciones o métodos para buscar subcadenas de texto)*
- *Al final de cada iteración, usaremos Sys.sleep() de unos pocos segundos, para no torpedear el servidor que aloja la página web.*
- *Finalmente agruparemos los datos en un data.frame: texto del enlace, url del enlace, veces que se ha visto cada enlace, y el código de retorno.*
- *Si hemos mantenido en nuestro vector de URLs los elementos NA, no es de extrañar que hacer head() sobre estos, obtengamos códigos "404" o de error web.*

Por ejemplo, una página web con:

```
<html>
	<head><title>HOLA</title></head>
	<body>
		<p>Esto es una frase <a href = “URL1”> ENLACE</a></p>
		<p>Esto es otra frase <a href = “URL2”> OTRO ENLACE</a></p>
		<p>Otro párrafo</p>
		<a href = “URL1”>ENLACE FINAL</a>
	</body>
</html>
```

… queremos que nos devuelva algo como:

Cabecera:	“HOLA”

…y esto, suponiendo que URL1 y URL2 retornan un estado “200 OK”:

Enlace	Texto	Visto	Estado
URL1	ENLACE	2	200
URL2	OTRO ENLACE	1	200
URL1	ENLACE FINAL	2	200

Recomendaciones:

- Cuando tratéis datos y obtengáis resultados, guardadlos en variables en vez de imprimirlos por pantalla solamente. De esta forma podéis reaprovechar los resultados luego si fuera necesario.
- Si tenéis que iterar sobre vuestros datos, **aprovechad las funciones de vectorización** de R. Por ejemplo, para sumar un vector no hagáis “for(i in 1:10) s <- s + v[i]”, sino “s <- sum(v)”. Aprovechad también las diferentes funciones de la familia apply(): (> ?apply).
- Si tenéis que **filtrar filas o columnas de un data.frame**, en vez de iterar con un bucle, usad las **funciones de subset**: “[]” o $”.

- **La página de ejemplo a analizar es: https://www.mediawiki.org/wiki/MediaWiki**

- **Entenderemos que el dominio base es https://www.mediawiki.org.**

Pregunta 2:

Elaborad, usando las librerías de gráficos base y qplot (ggplot2), una infografía sobre los datos obtenidos. Tal infografía será una reunión de gráficos donde se muestren los siguientes detalles:

1. **Un histograma con la frecuencia de aparición de los enlaces, pero separado por URLs absolutas (con "http…") y URLs relativas.**

2. **Un gráfico de barras indicando la suma de enlaces que apuntan a otros dominios o servicios (distinto a https://www.mediawiki.org en el caso de ejemplo) vs. la suma de los otros enlaces.**
   Aquí queremos distinguir enlaces que apuntan a mediawiki versus el resto. Sabemos que las URLs relativas ya apuntan dentro, por lo tanto hay que analizar las URLs absolutas y comprobar que apunten a https://www.mediawiki.org.

   Pista:
   - Cuando usamos la función sum() o similares sobre un vector con valores NA, nos encontramos que el resultado es NA. Esto se da porque no se puede conocer una suma que incluye un elemento "desconocido". R permite incluir un parámetro a estas funciones "na.rm = TRUE" para indicar que se ignoren los elementos NA. Si para cuando convertimos NULLs a NA, no los filtramos para nuestro data.frame, nos será útil indicar que se ignoren los NA para estas operaciones.
   -

3. **Un gráfico de tarta (pie chart) indicando los porcentajes de Status de nuestro análisis.**
   Por ejemplo, si hay 6 enlaces con status "200" y 4 enlaces con status "404", la tarta mostrará un 60% con la etiqueta "200" y un 40% con la etiqueta "404". Este gráfico lo uniremos a los anteriores. El objetivo final es obtener una imagen que recopile los gráficos generados.

Usad la capacidad de R y ggplot2 para componer gráficos en una sola figura. Si tales gráficos están compuestos directamente desde R (y no en el documento memoria), se puntuará mejor.

Indicar a continuación la solución, justificando cada detalle y decisión tomada, e incluyendo los gráficos.