Descarga y procesamiento de los datos usados en los modelos de frecuencia mixta para el nowcast de PIB

Manual del Desarrollador - Automatización de procesos para el nowcast de PIB agregado y sectorial

Dirección General de Política Macroeconómica - Ministerio de Hacienda y Crédito Público

2024-05-27

El presente manual es una descripción detallada y completa del proceso de descarga y procesamiento de datos usados en los modelos de frecuencia mixta para el nowcast de PIB. Para ello, se explicará sucintamente las tecnologías necesarias para la automatización de la descarga y el procesamiento de datos, las funciones construidas y los paquetes requeridos. El documento sirve como una guía para comprender y familiarizarse con todo el proceso de descarga y procesamiento de los datos que se requierén en los modelos de frecuencia mixta, y también sirve como guía para facilitar el mantentimiento y las modificaciones al código. Contiene incialmente una sección preliminar donde se dan unas indicaciones generales del documento, luego una introducción donde se presenta en que contextos se utilizaría el códgo y las metodologías acá descritas, posteriormente se describen en detalle cada una de las etapas requeridas para realizar la actualización de la base de datos usada en los modelos de frecuencia mixta que se emplean en el nowcast de PIB las cuales son: 1) la descarga y obtención de los datos y 2) el procesamiento de los datos y la construcción de la base de datos usada en los modelos de frecuencia mixta, finalmente se encuentran los apéndices al documento.

Tabla de contenidos

Listado de Figuras

Listado de Tablas

1 Preliminares

Siglas

- **Terminología económica y estadística:**
  - **Nowcast** Pronóstico a muy corto plazo. Hace referencia a pronósticos de uno o máximo dos trimestres en el futuro.
- **Indicadores económicos:**
  - **ISE** Índice de Seguimiento a la Economía
  - **PIB** Producto Interno Bruto
- **Instituciones:**
  - **MHCP** Ministerio de Hacienda y Crédito Público
  - **DGPM** Dirección General de Política Macroeconómica
  - **Minenergía** Ministerio de Minas y Energía
  - **DANE** Departamento Administrativo Nacional de Estadística
  - **DIAN** Dirección de Impuestos y Aduanas Nacionales
  - **FNC** Federación Nacional de Cafeteros
  - **Aerocivil** Aeronautica Civil
  - **Banrep** Banco de la República
  - **Camacol** Cámara Colombiana de la Construcción

Conocimientos técnicos útiles que aprenderá en el documento

- Tecnologias web y web scraping mediante las librerias `rvest` en `R`
- Procesamiento avanzado de datos en Excel mediante las librerías `tidyxl` y `unpivotr`
- Manejo de *expresiones regulares* mediante la librería `stringr`
- Manejo de excepciones mediante mediante la sentencia `try and catch` de R
- Programación funcional básica en R
- Desestacionalización de series de tiempo en R
- Estimación de un Filtro de Regresión en Tres Pasos en R
- Manejo de series de tiempo en R mediante librerias especializadas como lo son `xts` y `fable`

2 Introducción

En la actualidad, existen diferentes indicadores macroeconómicos que en conjunto pueden dar un diagnóstico de la situación económica del país, lo que resulta muy útil para la toma de decisiones de los agentes y la formulación de política pública por parte de entidades como el Ministerio de Hacienda y Crédito Público. Dichos indicadores, pueden ser encontrados en las diferentes páginas web de diversos tipos de organizaciones y agencias como lo son las entidades gubernamentales, las agencias de estadísticas, los gremios y en general el sector privado.

El objetivo del presente documento, consiste primordialmente en explicar de manera detallada el proceso de actualización de la base de datos que usan los modelos de frecuencias mixtas para el nowcast de PIB. Dicho proceso, se compone de dos etapas: 1) la descarga y obtención de datos de diferentes fuentes de información que se encuentran en páginas web de entidades gubernamentales, agencias estadísticas, gremios y el sector privado y 2) el procesamiento de los datos y la actualización de la base de datos que que usan los modelosde frecuencias mixtas para el nowcast de PIB.

2.1 Descripción general del proceso completo para actualizar la base de datos que usan los modelos de frecuencias mixtas para el nowcast de PIB

El proceso completo para actualizar la base de datos que usan los modelos de frecuencias mixtas:

1. Descarga y obtención de los datos
   - Descarga automática de bases de datos de páginas web
     - Vía `rvest en R`
   - Obtención manual de variables, cuándo los datos se generán y obtienen de manera interna dentro del ministerio.
2. Procesamiento de las bases de datos para actualizar de manera automática la información dentro de la base de datos usada por los modelos de frecuencias mixtas para el nowcast de PIB.

2.2 Manejo de directorios y bases de datos

Subdirectorios principales

Hay dos subdirectorios principales los cuáles son:

- **bases_datos:** Contiene todas las bases de datos que funcionan como entrada (*input*) para los programas que descargan los datos de la web y los procesan. Adicionalmente, contiene la base de datos actualizada que es usada por los modelos de frecuencias mixtas para el nowcast de PIB y que es el resultado final (*output*) de los programas.
- **scripts:** Contiene los scripts necesarios para la descarga y el procesamiento de los datos que se emplean para actualizar la base de datos que usan los modelos de frecuencia mixta para el nowcast de PIB.
- **readme:** Contiene la documentación relacionada con el funcionamiento de los programas, incluyendo el presente manual.

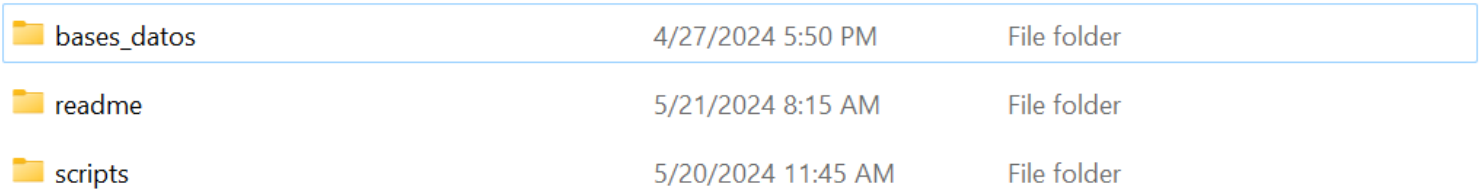


Figura 1: Subdirectorios principales

2.2.1 Subdirectorio "bases_datos"

Subdirectorio "bases_datos"

El subdirectorio *bases_datos* contiene las siguientes carpetas:

- **input_web_scraping:** Contiene las bases de datos que sirven como entrada (input) para los programas de descarga y procesamiento de datos .
- **output_web_scraping:** Contiene las bases de datos que resultan de procesos realizados por los programas, ya sea de descarga o actualización de la base de datos final para los procesos de
- **adicionales_info:** Contiene todo aquel material que sirva para complementar y facilitar el endentimiento del proceso de actualización de la base de datos que usan los modelos de frecuencias mixtas.

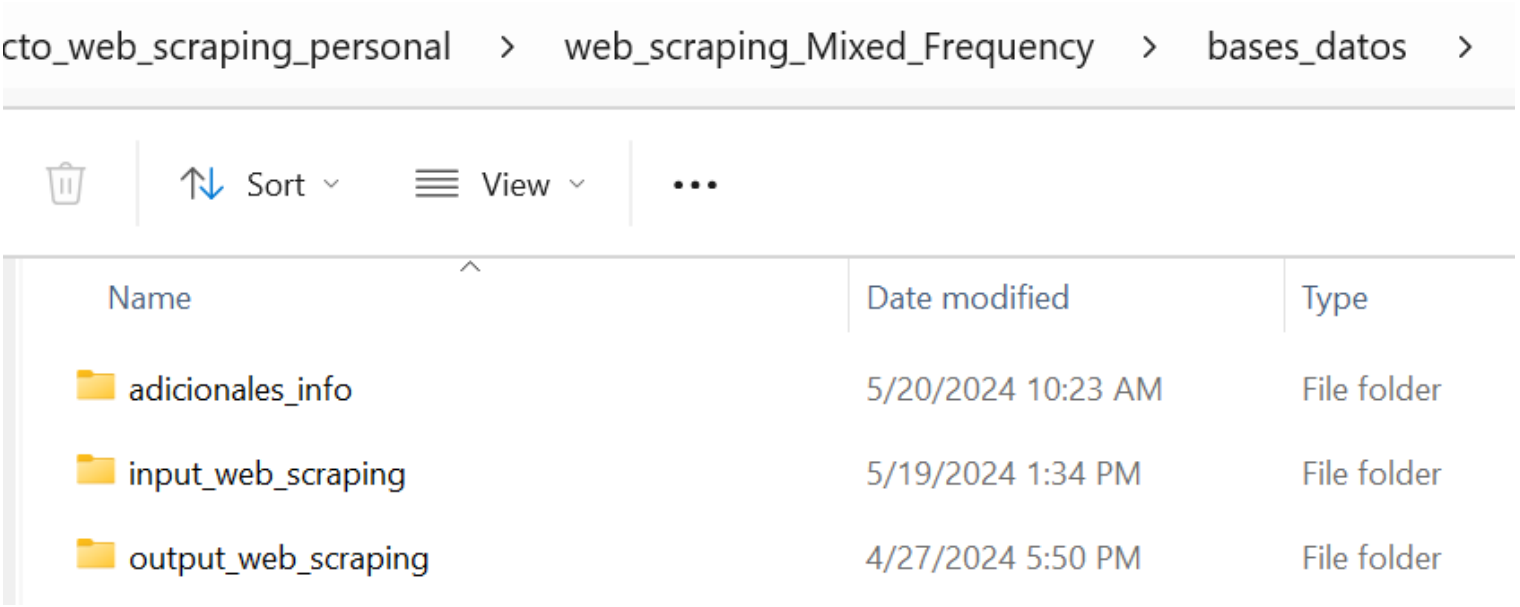


Figura 2: Subdirectorio “bases_datos”

2.2.1.1 Subdirectorio “input_web_scraping”

Dentro del subdirectorio *input_web_scraping* se encuentran las bases de datos que sirven como input para los programas de descarga y procesamiento de datos que sirven para la actualización de la base de datos de los modelos de frecuencias mixtas que se usan en el nowcast de PIB.

! Bases de datos que sirven de entrada (input) para la actualización de la base de datos que usan los modelos de frecuencias mixtas para el nowcast de PIB

Las bases de datos de entrada (input) para la actualización de la base de datos de los modelos de frecuencias mixtas son las siguientes:

- `base_final_pre_actualizacion.xlsx`: Consiste en la base de datos final usada para los modelos de frecuencia mixta para el nowcast de PIB , previa a la actualización de los datos. Se sugiere usar la última versión disponible pre-actualización de datos siempre que se vaya a hacer una nueva actualización a dicha base de datos final.
- `input_web_scraping_MF.xlsx`: Contiene toda la *metadata* necesaria para la descarga de las bases de datos y el procesamiento de las mismas, que permite actualizar la base de datos final que es usada por los modelos frecuencias mixtas para el nowcast de PIB.

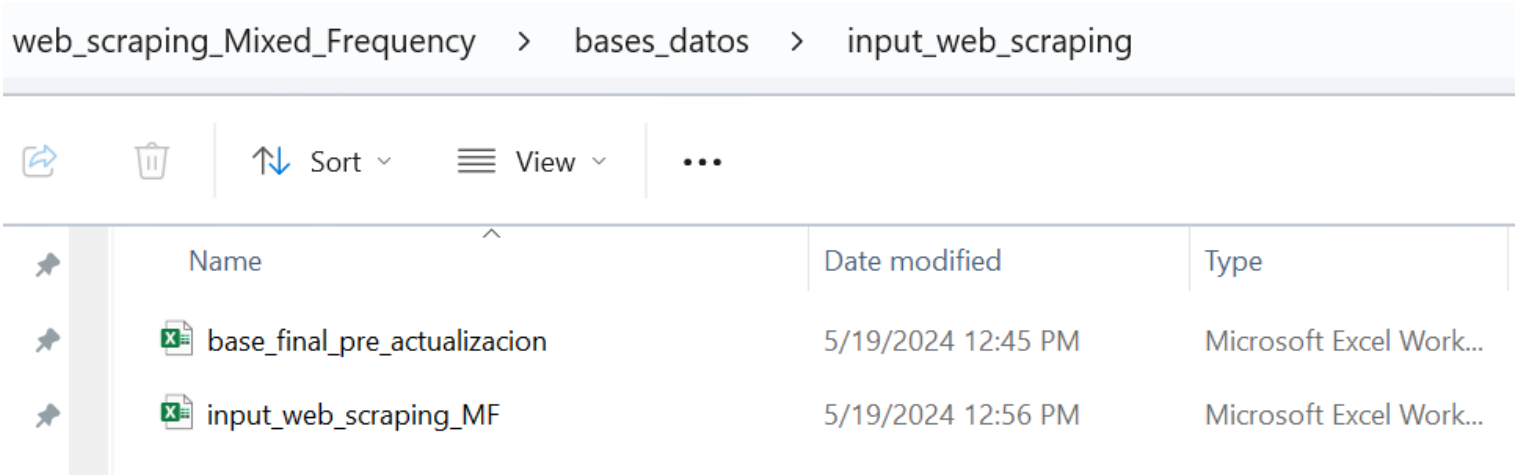


Figura 3: Subdirectorio “input_web_scraping”

En posteriores secciones del documento, se realizará una explicación más detallada de cada una de las bases mencionadas.

2.2.1.2 Subdirectorio "output_web_scraping"

⚠ Subdirectorio output_web_scraping

El subdirectorio *output_web_scraping* contiene las siguientes carpetas:

- **bases_descargadas:** Contiene todas las bases de datos en formato `.xls` o `.xlsx` que fuerón descargadas mediante el proceso de *web scraping* usando el paquete `rvest` de R.
- **bases_finales:** Contiene las bases de datos finales usadas por los modelos de frecuencias mixtas para el nowcast de PIB, luego de haber procesado las bases de datos descargadas por el proceso de *web scraping* y haber actualizado dichas bases finales usando los nuevos datos. Es decir, acá se encuentran las bases de datos finales listas para ser empleadas en los modelos de frecuencias mixtas para el nowcast de PIB.

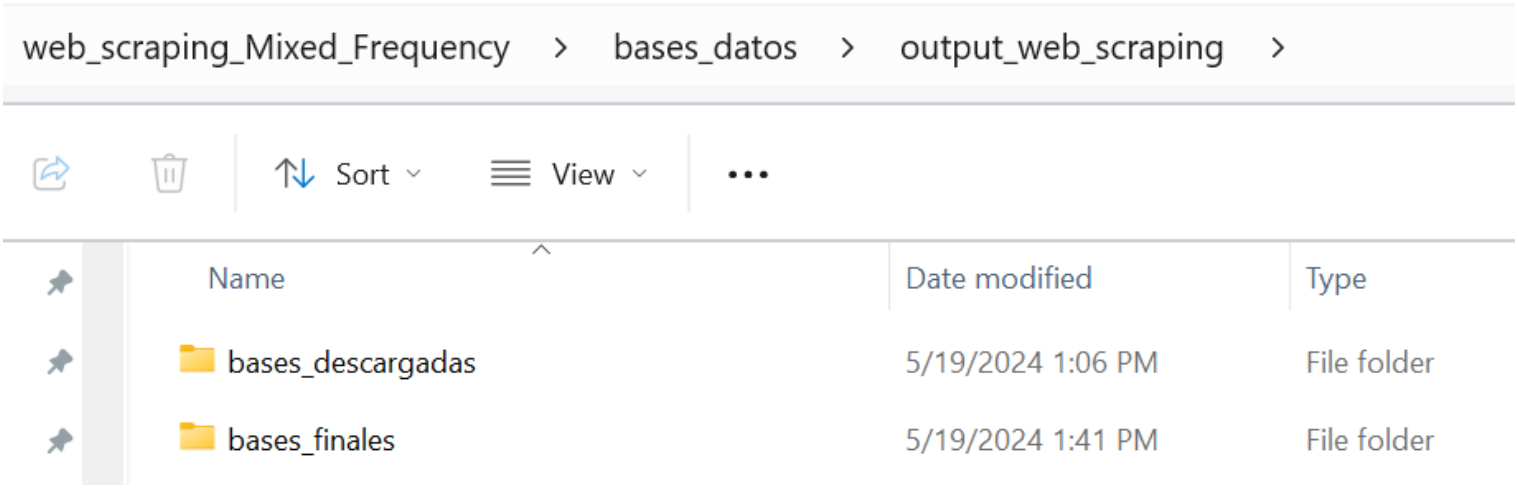


Figura 4: Subdirectorio "output_web_scraping"

2.2.1.3 Subdirectorio "adicionales_info"

⚠ Subdirectorio adicionales_info

El subdirectorio *adicionales_info* contiene bases de datos adicionales que no hacen parte del proceso de descarga y procesamiento de datos, pero que incluyen información adicional que puede ser relevante para el operario:

- `Códigos_Sectores_mixed_frequency,xlsx`**:** Contiene cada uno de los códigos de las variables tanto de alta como de baja frecuencia que se usan para identificar a cada una de las variables que son usadas por los modelos de frecuencias mixtas para realizar el nowcast de PIB a nivel sectorial.

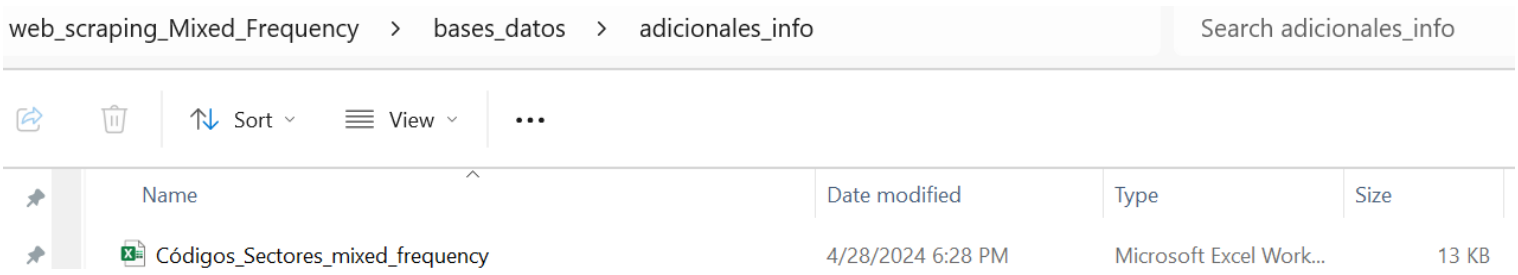


Figura 5: Subdirectorio "adicionales_info"

2.2.2 Subdirectorio “scripts”

Dentro del subdirectorio *scripts* se encuentran los scripts que sirven para descargar y procesar los datos que se emplean para actualizar la base de datos final que usan los modelos de frecuencias mixtas para el nowcast de PIB.

! Subdirectorio “scripts”

Scripts que hacen parte del directorio, y son los que permiten realizar la actualización de la base de datos final que usan los modelos de frecuencias mixtas para el nowcast de PIB:

- `web_scraping_R_funciones_descarga.R`: Contiene todas las funciones necesarias para descargar los base de datos que se encuentra en las diferentes fuentes de información de las paginas web. El operario no lo ejecuta directamente, sino indirectamente a partir del script `web_scraping_R_full_process_operario`.
- `web_scraping_R_funciones_procesamiento_bases.R`: Contiene todas las funciones necesarias para el procesamiento de las bases de datos descargadas anteriormente de las diferentes fuentes de información de las páginas web y adicionalmente las funciones para actualizar labase de datos que usan los modelos de frecuencias mixtas para el nowcast de PIB. El operario no lo ejecuta directamente, sino indirectamente a partir del script `web_scraping_R_full_process_operario`.
- `web_scraping_R_full_process_operario.R`: Script que es ejecutado directamente por el operario y que permite de manera automática descargar y procesar los datos que sirven para actualizar la base de datos final que usan los modelos de frecuencias mixtas para el nowcast de PIB.

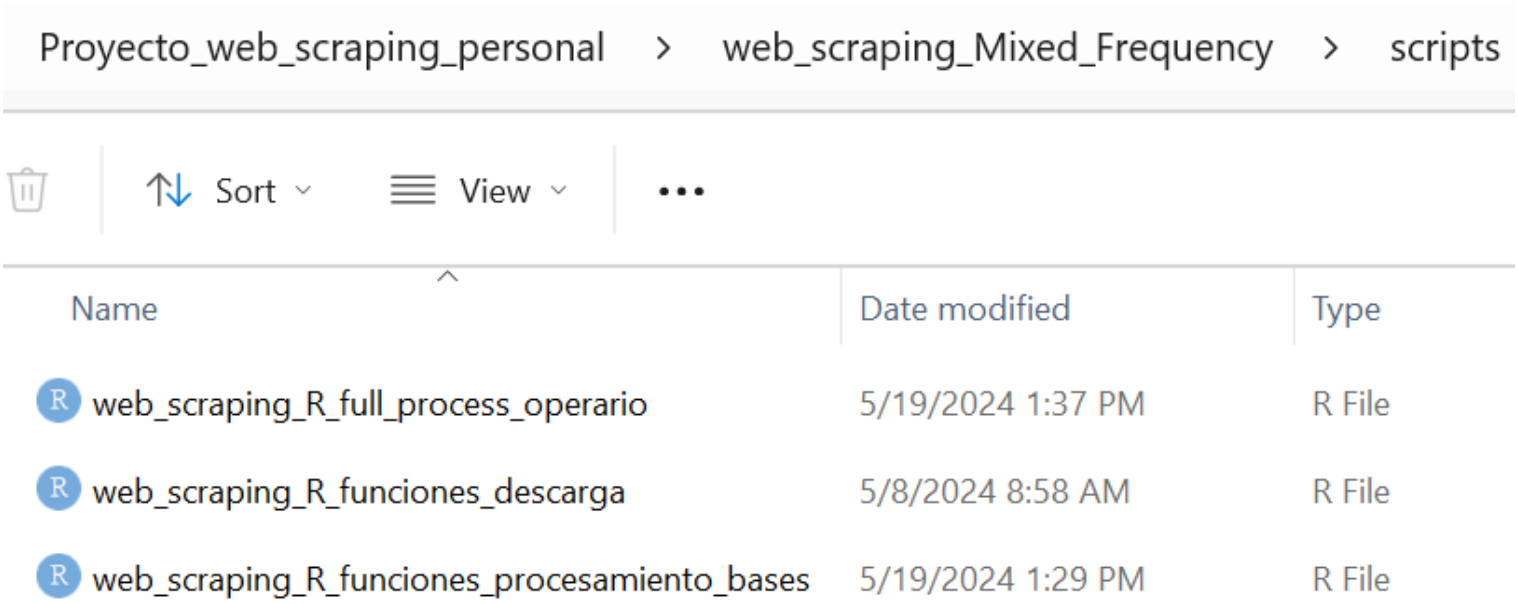


Figura 6: Subdirectorio “scripts”

3 Descarga y obtención de los datos para los modelos de frecuencias mixtas

3.1 Principales tecnologías que se usan en la programación web: HTML, CSS y JavaScript

3.1.1 Tecnología princial: HTML

3.1.1.1 Aspectos generales de HTML

HyperText Markup Language (**HTML**) es el lenguaje estándar de tipo **markup** para el diseño de documentos que van a ser renderizados en un navegador web, es decir, toda página web tiene asociada un documento `HTML` que le da su estructura. Al no ser posible renderizar en un navegador una página web sin un documento `HTML` asociado a ésta, el lenguaje `HTML`es la tecnología más importante que se debe conocer a la hora de realizar web scraping dado que **siempre** se va a tener que interactuar con un documento `HTML` antes de hacer cualquier tipo descarga en la web vía web scraping. La extensión de un archivo `HTML` es `.html` y dichos documentos pueden abrirse directamente en un navegador sin necesidad de un tratamiento adicional más allá de crear el documento con la extensión `.html`.

Todo documento `HTML` tiene estructura de árbol con un nodo raíz o principal, del cuál se desprenden nodos hijos. A los diferentes nodos dentro de un documento `HTML`también se les conoce como `tags` y dependiendo del tipo de `tag` este tendrá `atributos` diferentes. Un atributo es una característica o propiedad específica del `tag` al que corresponde, es decir, diferentes tipos de `tags` tendrán diferentes atributos asociados.

> Descripción de algunos de los tags básicos que conforman el lenguaje HTML

A continuación, se da una lista no exhaustiva de algunos de los `tags` básicos que conformán el lenguaje `HTML`:

- `<html> </html>`: Nodo raíz del que se desprenden todos los documentos
- `<head> </head>`: Nodo que se deprende inmediatamente del nodo raíz <html>. Contiene toda la metadata relacionada con la página web.
- `<body> </body>`: Nodo que se deprende inmediatamente del nodo raíz <html>. Contiene todo los tags que conforman el documento `HTML` de la página web, es decir, toda la estructura de la página web se encuentra dentro de este tag.
- `<div> </div>`: Tags usados para dividir un documento en diferentes partes (*divisions*). Tags de propósito general.
- `<h1> </h1> --> <h6> </h6>`: Tags que representan encabezados (Headers) que van desde tipo 1 (`<h1> </h1>`) hasta tipo 6 (`<h6> </h6>`) (el orden representa el tamaño del fondo).
- `<p> </p>`: Tags tipo párrafo.
- `<a> </a>`: Tags que contiene links dentro de ellos. El atributo `href` es el que generalmente contiene el link, como lo puede ser un link descargable a un archio `.xlsx` que se encuentre en la web.
- `<ul> </ul>`: Tags que representan una lista no ordenada.
- `<li> </li>`: Tags que representan una lista ordenada.

Para mayor información acerca de los tags que conforman el lenguaje y de sus atributos el lector se puede remitir al Stanford HTML Cheatsheet, donde encontrará una lista más exhaustiva y mayor profundidad sobre cada uno de los tags que hacen parte del lenguaje.

Todo `tag` tiene la misma sintaxis, empezando por `<tag>` y finalizando por `</tag>,` y todo `<tag>` que se encuentre dentro de otro `<tag>` se considera hijo (child) del `tag` que lo contiene. La Figura 7 muestra un ejemplo muy simple de un documento HTML que ilustra dicha idea.

Figura 7: Ejemplo de documento HTML simple. Tomado de DataCamp, curso Web Scraping in R.

Al lado izquierda de la Figura 7 se encuentra el ejemplo de un documento HTML sencillo y al lado derecho se encuentra como se vería dicho documento renderizado en un navegador. Se observa un nodo raíz en `<html> </html>` y un nodo hijo directo en `<body> </body>`. Dentro del tag `<body> </body>` se encuentra los tags `<h2> </h2>` y `<p> </p>` que hacen referencia al encabezado y los dos parágrafos que se observan al lado derecho de la figura.

```
<p>
  Here follows a
  <a href="https://google.com">link</a>.
</p>
```

Figura 8: Ejemplo de tag `<a> </a>` que contiene un link asociado en su atributo `href`.Tomado de DataCamp, curso Web Scraping en R.

Un ejemplo de un `tag` que contiene asociado un link asociado se presenta en la Figura 8.

En la Figura 8 se observa que dentro del tag `<a> </a>` se encuentra el atributo `href` que contiene un link directo a *google.com*.

La Figura 9 muestra una representación adicional de un documento `HTML`. Al lado izquierdo se encuentra el código HTML del documento, mientras que al lado derecha se encuentra una representación esquemática de dicho código.

HTML is like a tree

```
<html>
  <body>
    <div>
      <p>The first paragraph.</p>
    </div>
    <div>
      Not an actual paragraph,
      but with a <a href="#">link</a>.
    </div>
    <p>A paragraph without an
      enclosing div.</p>
  </body>
</html>
```

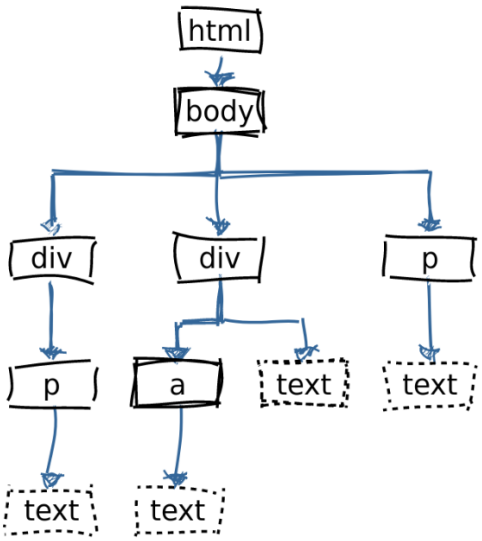


Figura 9: Ejemplo de la estructura de árbol de un código HTML. Tomado de DataCamp, curso Web Scraping en R.

De la Figura 9 se observa la estructurar jerarquíca de árbol que tiene el código `HTML`. Inicialmente hay un nodo raíz `<html> </html>` del que se desprende un nodo body y luego se desprenden dos nodos `<div> </div>` y un nodo `<p> </p>` dandole la estructura al documento `HTML` y por ende a la página web.

3.1.1.2 Selección de un Tag específico en un documento HTML desde el navegador

Para seleccionar un `tag` específico dentro de un documento `HTML` desde el navegador lo primer que hay que hacer es abrir la `ventana del desarrollador` del navegador oprimiendo la tecla `f12`. Eso va a abrir una ventana en el navegador que contiene todas las herramientas que permiten realizar programación web dentro del navegador. Dentro de la pestaña elementos del la ventana del desarrollador se encuentra el documento `HTML` completo de la página web.

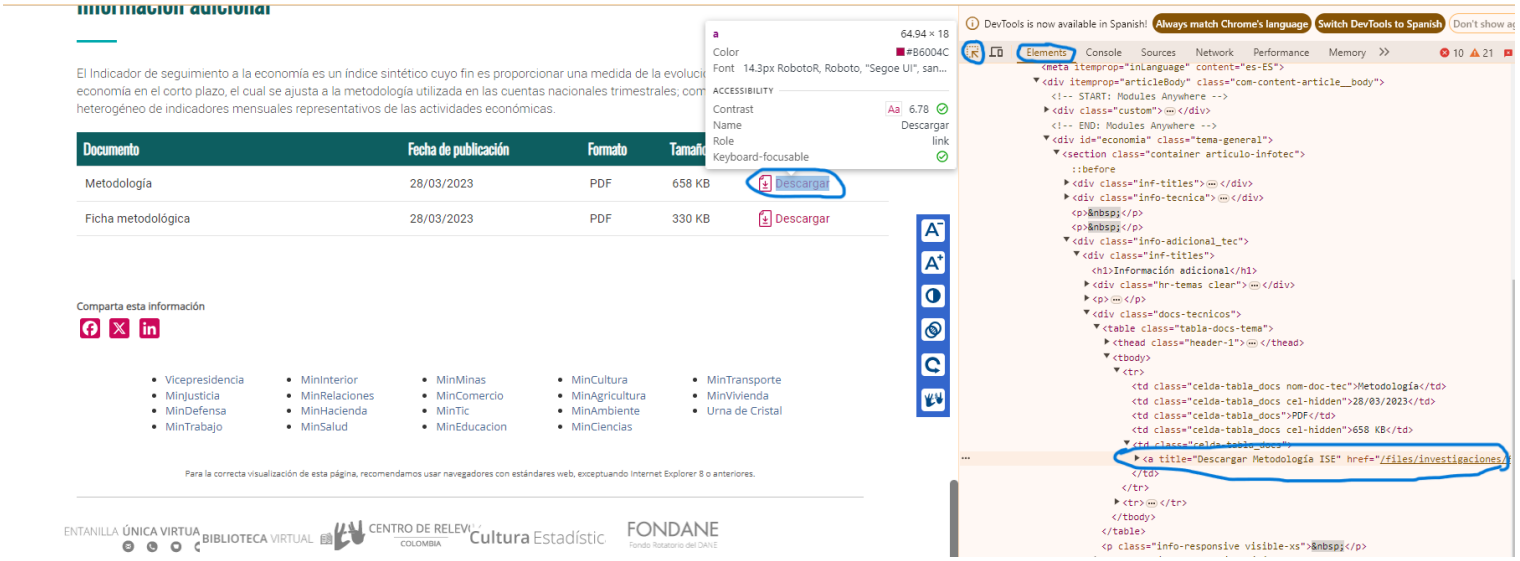


Figura 10: Elementos necesarios para identificar un tag específicado de un documento HTML de una página web

Adicionalmente, en la esquina superior izquierna izquierda se encuentra una flecha diagonal (inclinación 45 grados) que apunta hacía el noroccidente de la pantalla. Dandole click a dicha flecha u oprimiento la combinación de teclas `ctrl + shift + c` se activa la posibilidad de navegadra a través de la página web y al pasar con el mouse por los diferentes elementos de la página web en azul se van a ir resaltando cada uno de los elementos de la página al que el mouse está apuntando. Al darle click en dicho elemento de la página web resaltado en azul, la ventana del desarrollador automáticamente selecciona el `tag` correspondiente dentro del documento `HTML` al elemento de la página sobre el cual se le ha hecho click.

La Figura 10 muestra el proceso descrito anteriormente para seleccionar el tag que contiene la guía metodológica del ISE. Al realizar los pasos descritos anteriormente y darle click en el elemento de la página web que dice `Descargar` y se encuentra sombreado en azul se selecciona de manera automática en la ventana del desarrollador un `tag` tipo `<a> </a>`. En dicho `tag`, se encuentra el atributo `href` que contiene el link por el cuál se puede descargar el PDF de dicha guía metodológica.

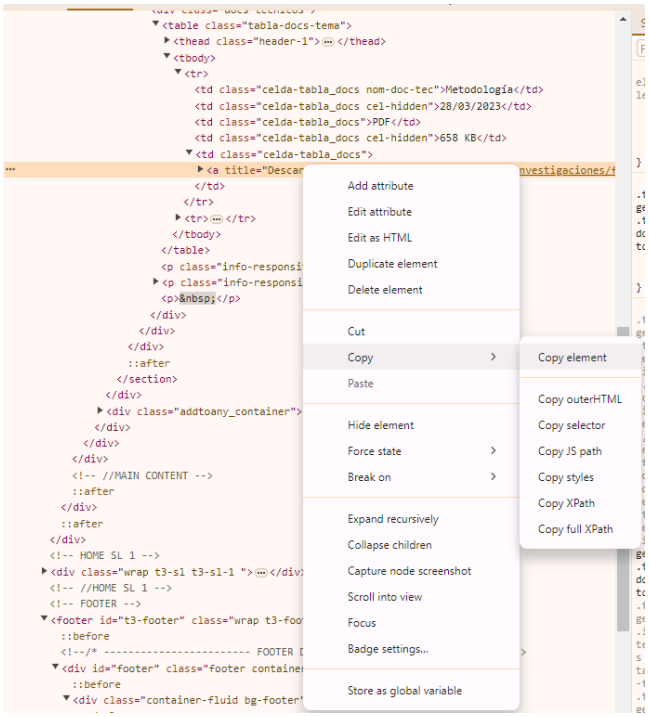


Figura 11: Copiar el CSS Selector o el XPath asociado al nodo o tag seleccionado en la ventana del desarrollador

La figura Figura 11 muestra que luego de identificar el `tag` o nodo de interés del documento `HTML` dando click derecho en dicho `tag` y luego específicando la opción Copy aparecen las opciones `Copy selector` para copiar el `CSS Selector` asociado a dicho `tag` o la opción `XPath` para

copiar el xpath asociado a dicho `tag`. Ambos selectores son igualmente validos y funcionales, y el uso del uno u el otro queda a gusto del desarrollador y eloperario. Son simplemente, maneras diferentes de identificar el mismo nodo o tag. Después de seleccionar `CSS Selector` o `XPath` ya es posible identificar el nodo que tiene la información o base de datos que se va a descagar y emplear técnicas de `web scraping` para descargar dicha información.

3.1.2 Tecnología adicionales: CSS y JavaScript

Cascading Style Sheets CSS es la segunda tecnología web más importante para el desarrollo de páginas web después de `HTML`. Es un lenguaje cuyo proposito consiste en darle diseño y estilo a un documento escrito en lenguaje tipo `markup` como lo pueden ser `HTML` o `XML`. Por ende, se usa para darle elementos de diseño adicionales a los documentos web que contienen un archivo `HTML` asociado. Básicamente, `CSS` establece reglas de diseño que le indican al navegador como debería renderizar una página web. Por tanto, en el desarrollo el lenguaje `CSS` se puede ver como un lenguaje comoplemtario a `HTML`. El lenguaje `CSS` utiliza **Selectores CSS** (`CSS Selectors`) que lo que permiten es identificar nodos o `tags` específicos del document `HTML` para darles cierto estilo. En lo que respecta al `web scraping`, la sintáxis de dichos `Selectores CSS` también puede ser usada para identificar los nodos o `tags` de interés a la hora de hacer la descarga de información en la página web que se encuentre en un nodo específico.

JavaScript, a menudo abreviado como `JS`, es un lenguaje de programación y tecnología central de la web, junto con `HTML` y `CSS`. La mayor parte de las páginas web hoy en día utilizan `JavaScript` en el lado del cliente (`Front-end`) para manejar el comportamiento de ésta, lo que lo hace el lenguaje de programación más porpular del mundo. `JavaScript` fue diseñado por Brendan Eich, el mismo desarrollador de los navegadores de código abierto `Mozilla Firefox` y `Brave`. En lo que respect al `web scraping`, como se explícará más adelante, las técnicas y herramientas para extraer información de la web de manera automática diferirá dependiendo de si la página es activamente renderizada por `JS` o no. Las páginas web estáticas, que no requiere el uso activo de un navegador que corra o ejectue `JavaScript`, emplean técnicas de web scraping relativamente sencillas y la mayoría de las librerías diseñadas para web scraping en `R` y `python` tienen las herramientas para realizar dicha descarga de datos. Por su lado, las páginas web dinámicas, que sí requiere el uso activo de un navegador que corra o ejectue `JavaScript` detrás para poder renderizar la página web, requieren de técnicas de `web scraping` que emulen lo que haría una persona en una navegador, y para ello requieren el uso activo de un navegador que ejecture `JavaScript` en tiempo real mientras se hace la descarga de datos.

3.2 `Web scraping`

3.2.1 ¿Qué es el `Web scraping`?

Es un conjunto de técnicas y métodos que permite extraer y descargar información y datos de páginas web de manera automática. A nivel práctico, el `web scraping` consiste en la realización de un programa capaz de acceder a sitios web y recuperar información dentro de éstos. Para ello, el programa mediante una librería especializada accede al archivo `HTML` de la página web de la cuál se quiere obtener información e interactua con dicho archivo `HTML` para obtener los datos requeridos.

Detrás de dicho proceso, cuándo una libería interactúa con una página web, lo que está haciendo es enviar y recibir solicitudes `HTTP` que es la forma en la que los diferentes servidores que conforman la web se comunican, es decir, cada solicitud que se haga desde una librería, como lo puede ser por ejemplo descargar una base de datos de una página web, lo que está haciendo es enviar una solicitud `HTTP` al servidor que contiene la página web para que se haga la descarga de la base.

El `web scraping` es un procedimiento muy conveniente para automatizar tareas repetitivas, como lo puede ser la descarga de datos, o cuándo haya que descargar un volumen muy grande de información de la web y sea incoveniente realizar dicha tarea manualmente. Por ello, las técnicas de web scraping son muy convenientes para automatizar procesos dentro de la DGPM del ministerio, y en particular para automatizar la obtención de los datos requeridos para la estimación del nowcast de PIB utilizando modelos de frecuencias mixtas.

Uno de los beneficios del web scraping es que reduce errores de digitalización que se pueden dar cuándo un operario copia y pega manualmente de una base de datos a otra y reduce sustancialmente el tiempo de construcción de bases de datos, por ejemplo, la base de datos final que se usa en los modelos de frecuencias mixtas para los nowcast de PIB sectorial se pueden hacer construir y actualizar en menos de 10 minutos utilizando las técnicas de web scraping.

3.2.2 ¿Cómo realizar `Web scraping`?

Para realizar web scraping comunmente se emplea una librería especializada para dicha tarea. En el caso de `python` existen librerías como `Scrapy` o `BeautifulSoup` y en el caso de R `rvest` y `selenider`. Adicionalmente, en el caso en que se necesite realizar web scraping más avanzados o testeo webs que requieran el uso directo de un navegador, se emplea la librería selenium, disponible en `Java`, `python` y `R`, que ofrecer una variedad de herramientas para trabajar directamente con un navegador web.

Ahora bien, dependiendo de si una página es renderizada dinámicamente utilizando `JavaScript` o no las técnias de web scraping pueden cambiar, a continuación se da una explicación más detallada de las diferencias entre el web scraping en páginas estáticas vs el web scraping en páginas dinámicas.

3.2.2.1 Web scraping con páginas web estáticas

Una página web se considera *estática* cuando no requiere el uso activo de un navegador que corra o ejectue `JavaScript` detrás para poder renderizar la página, es decir, con solo el archivo plano `HTML` es posible renderizar completamente la página web.

Páginas web de instituciones como el DANE, Camacol, Aerocivl, FNC y DIAN, tienen esa característica, por lo que el proceso de web scraping para descargar información de dichas páginas es más fácil de realizar, es más véloz e inmediato.

Librerías como `rvest` a la hora de realizar web scraping en una página web estática simplemente envia un requerimiento `HTTP` y con saber cuál es el nodo o `tag` que contiene la información de interés es posible descargar dicha información.

3.2.2.2 Web scraping con páginas web dinámicas

Una página web se considera *dinámica* cuando requiere el uso activo de un navegador que corra o ejectue `JavaScript` detrás para poder renderizar la página web, es decir, el navegador activamente utiliza `JavaScript` para que interactúe con los diferentes nodos o `tags` del archivo `HTML` y de esta forma pueda darle el dinamismo en tiempo real a la página web. Por tanto, si el navegador web no está ejectutando `JavaScript` sobre la página web, ésta no va a poder ser renderizada por parte del navegador.

Páginas web como lo puede ser la plataforma `totoro` del Banrep son páginas dinámicas porque si el navegador no ejecuta JavaScript la página no va a ser renderizada en lo absoluto.

El procedimiento de web scarping para una página web dinámica difiere del procedimiento que hay que llevar a cabo para una página web estática. Mientras que en una página web estática simplemente con conocer el nodo específico que contiene la información de interés es posible descargar los datos, en el caso de una página web dinámica generalmente hay que emular lo que haría una persona en una navegador, es decir, un procedimiento de web scraping en una página dinámica generalmente tiene que replicar acciones como dar clicks, ingresar texto en un `entry` entre otras cosas que haría un usuario dentro de un navegador web. Para el caso de python, la librería por excelencia para realizar web scraping en una página web dinámica es `selenium`, mientras que en el caso de R la librería `selenider` permite realizar dicho tipo de web scraping en páginas web dinámcias.

3.3 Tablas de las variables que se encuentran en la base final usada por los modelos de frecuencia mixta para el nowcast de PIB

Actualmente, los **modelos de frecuencia mixta** emplean en su estimación un total de 28 variables de baja frecuencia (variables trimestrales) y 27 variables de alta frecuencia (variables mensualesa) provenientes de diferentes fuentes de datos como lo son instituciones gubernamentales, agencias de estadísticas, gremios y diversas entidades privadas. De esas variables, las 28 variables de baja frecuencia se descargan de manera automática y 21 de las 27 variables de alta frecuencia se descargan de manera automática, las otras 6 variables se obtiene de manera manual al ser variables que se producen de manera interna dentro del ministerio.

La Figura 12 muestra los códigos de las variables que son usadas por los modelos de frecuencias mixtas para los nowcast de PIB correspondiente a los sectores de la economía vistos por el lado de la oferta. En verde se encuentran las variables que no son descargadas por el código, dado que son variables que se producen de manera interna por parte de la DGPM del MHCP.

La Figura 13 muestra los códigos de las variables que son usadas por los modelos de frecuencias mixtas para los nowcast de PIB correspondiente a los sectores de la economía vistos por el lado de la demanda.

3.3.1 Tabla variables de baja frecuencia (trimestrales)

Las 28 variables de baja frecuencia que son descargadas y procesadas de manera automática están dadas por la tabla Tabla 1 .

Tabla 1: Tabla de las variables de baja frecuencia que son usadas por los modelos de frecuencias mixtas cuya obtención ya fue automatizada vía *web scraping*

No.	Nombre de la variable	Fuente	Modo de obtención
1	Agrícultura y ganadería - Cultivos (y_cultivos)	DANE	Automática - Web Scraping (rvest en R)
2	Agrícultura y ganadería - Café (y_cafe)	DANE	Automática - Web Scraping (rvest en R)
3	Agrícultura y ganadería - Pecuario (y_pecuario)	DANE	Automática - Web Scraping (rvest en R)
4	Explotación de minas y canteras - Carbón (y_carbon)	DANE	Automática - Web Scraping (rvest en R)
5	Explotación de minas y canteras - Petróleo (y_petroleo)	DANE	Automática - Web Scraping (rvest en R)
6	Explotación de minas y canteras - Minerales Metalíferos (y_metaliferos)	DANE	Automática - Web Scraping (rvest en R)
7	Industrías manufactureras (y_industria)	DANE	Automática - Web Scraping (rvest en R)
8	Electricidad, gas y agua (EGA) (y_ega)	DANE	Automática - Web Scraping (rvest en R)
9	Construcción - Edificaciones (y_edificaciones)	DANE	Automática - Web Scraping (rvest en R)
10	Construcción - Obras Civiles (y_obrasciviles)	DANE	Automática - Web Scraping (rvest en R)
11	Comercio - Gran rama de comercio (y_comercio)	DANE	Automática - Web Scraping (rvest en R)
12	Comercio - Comercio minorista (y_minorista)	DANE	Automática - Web Scraping (rvest en R)
13	Comercio - Transporte (y_transporte)	DANE	Automática - Web Scraping (rvest en R)
14	Comercio - Alojamiento (y_alojamiento)	DANE	Automática - Web Scraping (rvest en R)

No.	Nombre de la variable	Fuente	Modo de obtención
15	Información y comunicaciones (y_infocom)	DANE	Automática - Web Scraping (rvest en R)
16	Actividades financieras (y_financieras)	DANE	Automática - Web Scraping (rvest en R)
17	Actividades inmobiliarias (y_inmobiliarias)	DANE	Automática - Web Scraping (rvest en R)
18	Actividades profesionales (y_profesionales)	DANE	Automática - Web Scraping (rvest en R)
19	Administración pública, educación y salud (y_admin_publica)	DANE	Automática - Web Scraping (rvest en R)
20	Actividades artísticas y de entretenimiento (y_artisticas)	DANE	Automática - Web Scraping (rvest en R)
21	Impuestos indirectos (y_imp_indirectos)	DANE	Automática - Web Scraping (rvest en R)
22	Consumo privado (y_cons_priv)	DANE	Automática - Web Scraping (rvest en R)
23	Consumo público (y_admin_publica)	DANE	Automática - Web Scraping (rvest en R)
24	Formación bruta de capital fijo - Vivienda (y_vivienda)	DANE	Automática - Web Scraping (rvest en R)
25	Formación bruta de capital fijo - Otros edificios y estructuras (y_otrosedificios)	DANE	Automática - Web Scraping (rvest en R)
26	Formación bruta de capital fijo - Maquinaría y equipo (y_maquinaria)	DANE	Automática - Web Scraping (rvest en R)
27	Exportaciones (y_expos)	DANE	Automática - Web Scraping (rvest en R)
28	Importaciones (y_impo_cn)	DANE	Automática - Web Scraping (rvest en R)

3.3.2 Tabla variables de alta frecuencia (mensuales)

Las 27 variables de alta frecuencia que son descargadas y procesadas de manera automática están dadas por la tabla Tabla 2 .

Tabla 2: Tabla de las variables de alta frecuencia que son usadas por los modelos de frecuencias mixtas

No.	Nombre de la variable	Fuente	Modo de obtención
1	Producción de cultivos (cultivos)	MHCP	Variable interna MinHacienda
2	Producción de café (cafe)	FNC	Automática - Web Scraping (rvest en R)
3	Producción pecuaria (pecuario)	MHCP	Variable interna MinHacienda
4	Extracción de hulla - Carbón de piedra (ipi_carbon)	DANE	Automática - Web Scraping (rvest en R)
5	Producción de petróleo (prod_petro)	MHCP	Variable interna MinHacienda
6	Exportaciones de oro (expo_oro)	MHCP	Variable interna MinHacienda
7	Índice de producción real - Industría (ipr_industria)	DANE	Automática - Web Scraping (rvest en R)
8	Demanda de energía (dem_energía)	MHCP	Variable interna MinHacienda
9	ISE - EGA (ise_ega)	DANE	Automática - Web Scraping (rvest en R)
10	Inicialización de viviendas (ini_viv)	Camacol	Automática - Web Scraping (rvest en R)
11	Despacho concreto edificaciones (desp_conc_edif)	DANE	Automática - Web Scraping (rvest en R)
12	Despacho concreto obras civiles (desp_conc_obras_c)	DANE	Automática - Web Scraping (rvest en R)
13	ISE - Comercio (ise_comercio)	DANE	Automática - Web Scraping (rvest en R)
14	Índice de ventas reales - Comercio minorista (ivr_minorista)	DANE	Automática - Web Scraping (rvest en R)
15	PCA - Variables transporte (pca_trans_var)	MHCP	Variable interna MinHacienda
16	Ingresos reales - Hoteles (ingresos_hoteles)	DANE	Automática - Web Scraping (rvest en R)
17	ISE - Información y comunicaciones (ise_infocom)	DANE	Automática - Web Scraping (rvest en R)
18	ISE - Actividades financieras (ise_financieras)	DANE	Automática - Web Scraping (rvest en R)
19	ISE - Actividades inmobiliarias (ise_inmobiliarias)	DANE	Automática - Web Scraping (rvest en R)
20	ISE - Actividades profesionales (ise_profesionales)	DANE	Automática - Web Scraping (rvest en R)
21	ISE - Administración pública, educación y salud (ise_admin_publica)	DANE	Automática - Web Scraping (rvest en R)
22	ISE - Actividades artísticas y entretenimiento (ise_artisticas)	DANE	Automática - Web Scraping (rvest en R)
23	IVA - Interno (imp_indirectos)	DIAN	Automática - Web Scraping (rvest en R)
24	ISE - Actividades terciarias (ise_ter)	DANE	Automática - Web Scraping (rvest en R)
25	Importaciones - Bienes de capital (impo_capital_real)	DANE	Automática - Web Scraping (rvest en R)
26	Exportaciones - Totales (expom)	DANE	Automática - Web Scraping (rvest en R)
27	Importaciones - Totales (impo_bienes_reales)	DANE	Automática - Web Scraping (rvest en R)

3.4 Base de datos necesaria para la descarga de las bases de datos vía Web scraping

Como se mencionó en secciones anteriores, dentro del directorio de trabajo `input_web_scraping` se encuentra el archivo `input_web_scraping_MF.xls`
cuya hoja `descarga` contiene la información necesaria para realizar el proceso de descarga en R vía web scraping de las diferentes bases de datos que proveen las variables que usan los modelos de frecuencia mixta para el nowcast de PIB. El contenido de dicha base de datos se encuentra en la Figura 14 .

Códigos por sectores Equipo Real				
Sector	**Subsector**	**Código Baja frecuencia**	**Código Alta frecuencia (estructurada)**	**Códigos Alta frecuencia (no estructurada)**
Agricultura y ganadería	Cultivos	y_cultivos	cultivos	
	Café	y_cafe	cafe	
	Pecuario	y_pecuario	pecuario	
Explotación de minas y canteras	Carbón	y_carbon	ipi_carbon	
	Petróleo	y_petroleo	prod_petro	
	Minerales Metalíferos	y_metaliferos	expo_oro	
Industrias manufactureras		y_industria	ipr_industria	maleta
Electricidad, gas y agua (EGA)		y_ega	dem_energia	
			ise_ega	
Construcción	Edificaciones	y_edificaciones	ini_viv	
			desp_conc_edif	
	Obras Civiles	y_obrasciviles	desp_conc_obras_c	
Comercio	Gran rama de comercio	y_comercio	ise_comercio	computador | chevrolet
	Comercio minorista	y_minorista	ivr_minorista	alkosto
	Transporte	y_transporte	pca_trans_var	
			ise_comercio	servientrega
	Alojamiento	y_alojamiento	ingresos_hoteles	booking
Información y comunicaciones		y_infocom	ise_infocom	cinecolombia
Actividades financieras		y_financieras	ise_financieras	
Actividades inmobiliarias		y_inmobiliarias	ise_inmobiliarias	
Actividades profesionales		y_profesionales	ise_profesionales	secop
Administación pública, educación y salud		y_admin_publica	ise_admin_publica	
Actividades artísticas y de entretenimiento		y_artisticas	ise_artisticas	zoologico
Impuestos indirectos		y_imp_indirectos	imp_indirectos	

Figura 12: Códigos de los sectores por el lado de la oferta para pronósitico de nowcast usando modelos de frecuencias mixtas.

Componente	**Subcomponente**	**Código Baja frecuencia**	**Código Alta frecuencia (estructurada)**	**Códigos Alta frecuencia (no estructurada)**
Consumo privado		y_cons_priv	ise_ter	computador
Consumo público		y_admin_publica		
Formación bruta de capital fijo	Vivienda	y_vivienda	y_edificaciones	
	Otros edificios y estructuras	y_otrosedificios	y_obrasciviles	
	Maquinaria y equipo	y_maquinaria	impo_capital_real	indus_maqui_equip
Exportaciones		y_expos	expom	
Importaciones		y_impo_cn	impo_bienes_reales	

Figura 13: Códigos de los sectores por el lado de la demanda para pronóstico de nowcast usando modelos de frecuencias mixtas.

global_url	enlace_variable	tipo_selector	tag_selector_descarga	Fuente	nombre_base_descarga
https://www.dane.g	https://www.dane.gov	css_selector	#economia > section > div.info-	DANE	DANE_ISE_9_actividades.xlsx
https://www.dane.g	https://www.dane.gov	css_selector	#economia > section > div.info-	DANE	DANE_ISE_12_actividades.xlsx
https://www.dane.g	https://www.dane.gov	css_selector	#economia > section > div.info-	DANE	DANE_PIB_oferta.xlsx
https://www.dane.g	https://www.dane.gov	css_selector	#economia > section > div.info-	DANE	DANE_PIB_demanda.xlsx
https://camacol.co	https://camacol.co/nu	xpath	//*[@id="block-descargabletab	CAMACOL	CAMACOL_Tablas_coyuntura.xlsx
https://www.dane.g	https://www.dane.gov	css_selector	#cnpv > div > div > table:nth-chi	DANE	DANE_EMMET_territorial.xlsx
https://www.dane.g	https://www.dane.gov	css_selector	#cnpv > div > div > table:nth-chi	DANE	DANE_EMC.xlsx
https://www.dane.g	https://www.dane.gov	css_selector	#t3-content > div > div.com-cor	DANE	DANE_Importaciones.xls
https://federacionc	https://federaciondec	xpath	//*[@id="3891"]/div/div/div/div	FNC	FNC_Estadisticas.xlsx
https://www.dane.g	https://www.dane.gov	css_selector	#t3-content > div > div.com-cor	DANE	DANE_Exportaciones.xlsx
https://www.dane.g	https://www.dane.gov	css_selector	#t3-content > div > div.com-cor	DANE	DANE_ECG.xlsx
https://www.dane.g	https://www.dane.gov	css_selector	#t3-content > div > div.com-cor	DANE	DANE_EMA.xlsx
https://www.dian.g	https://www.dian.gov.	css_selector	#WebPartWPQ6 > div.ms-rtesta	DIAN	Recaudo_impuestos_DIAN_colombia
https://www.dane.g	https://www.dane.gov	css_selector	#rlta-panel-empleo-y-desocupa	DANE	DANE_Mercado_laboral.xlsx
https://www.dane.g	https://www.dane.gov	css_selector	#t3-content > div > div.com-cor	DANE	DANE_IPI.xlsx
https://www.dane.g	https://www.dane.gov	css_selector	#t3-content > div > div.com-cor	DANE	DANE_EC.xlsx

Figura 14: Base de datos necesaria par la descarga vía web scraping de las bases de datos que se encuentran en las páginas web.

Variables principales de la base de datos que específica la información necesaria para hacer la descarga de los datos

Las variables principales de la base de datos que se encuentra en la hoja `descarga` del archivo `input_web_scraping_MF.xlsx` son:

- `global_url`: Tiene la URL *global* de la página web específica para la entidad gubernamental, la agencia de estadística, el gremio o la entidad del sector privado. E.g. en el caso del DANE la URL correspondería a `https://www.dane.gov.co/`
- `enlace_variable:` Tiene la URL *específica* donde se encuentra la base de datos que se quiere descargar. E.g. en el caso de la bases de datos que contienen el ISE de 9 y 12 sectores, la URL correspondería a `https://www.dane.gov.co/index.php/estadisticas-por-tema/cuentas-nacionales/indicador-de-seguimiento-a-la-economia-ise`
- `tipo_selector`: Contiene el *tipo* de selector específico que se va a usar para seleccionar el nodo o tag específico que contiene la base de datos o la información que se quiere descargar. Todas las funciones de descarga, independientemente de la *fuente*, están diseñadas para admitir dos tipos de selectores:
  - `css_selector`: `Selector CSS` para identificar el nodo o tag que contiene la información o la base de datos que va a ser descargada. Dentro de la ventana del desarrollador en el navegador que se esté usando hay que seleccionar el tag específico que se encuentre en el documento `HTML`, dar click derecho y dar en la opción `Copy > Copy selector`.
  - `xpath`: `XPath` para identificar el nodo o tag que contiene la información o la base de datos que va a ser descargada. Dentro de la ventana del desarrollador en el navegador que se esté usando hay que seleccionar el tag específico que se encuentre en el documento `HTML`, dar click derecho y dar en la opción `Copy > XPath`.

    Nota: Ambos selectores son igualmente validos y funcionales, y el uso del uno u el otro queda a gusto del desarrollador y el operario. Son simplemente, maneras diferentes de identificar el mismo nodo o tag.
- `tag_selector_descarga`: Contiene ya sea el `Selector CSS` o el `XPath`, dependiendo de lo que se haya específicado en `tipo_selector`, del nodo o tag que contiene la información que se va a descargar.
- `Fuente:` Contiene el nombre de la entidad gubernamental, la agencia de estadística, el gremio o la entidad del sector privado de donde proviene la información. Dicha variable es importante, en la medida que idenfica cuál función de descarga específica se debe usar para descargar los datos, dado que cada entidad diferente tiene su propia función de descarga específica. A fecha del 22 de mayo de 2024, existen funciones de descarga para las siguiente entidades:
  - DANE
  - Camacol
  - Aerocivil
  - FNC
  - DIAN
- `nombre_base_descarga`: Contiene el nombre con el que se va a descargar la base de datos. Dicho nombre es específicado por el desarrollador o el operario, y puede ser cambiado a discresión según la nomenclatura que emplee la entidad para denotar dicha base de datos específica que se haya descargado.

Cada fila del Excel en la Figura 14, corresponde a una base de datos diferente, es decir, por cada base de datos nueva que se quiere descargar es necesario agregar una fila adicional en el Excel específicando la información necesaria para la descarga de dicha base, i.e., hay que especificar en una nueva fila la `global_url`, el `enlace_variable`, el `tipo_selector`, el `tag_selector_descarga`, la `Fuente` y el `nombre_base_descarga` asociados a cada nueva base de datos que se quiere decargar.

En la Sección 3.1.1.2 se encuentra la manera copiar el `Selector CSS` o el `XPath` asociado al nodo o tag que contiene la información o base de datos de interés. Dicho `Selector CSS` o `XPath` luego de ser copiado de la ventana del desarrollador del navegador se pega directamente en la columna tag_selector_descarga asociada a la fila específica del Excel que hace refrencia a la base de datos específica que se queire descargar.

3.5 Paquetes para realizar el Web scraping

3.5.1 Paquetes para realizar Web scraping y testeo de web en R

Paquetes para realizar Web scraping y testeo web en R

A continuación se encuentra una lista de los paquetes que pueden ser empleados para realizar web y testeo web en R

- `rvest`: Paquete que pertenece al `tidyverse` y permite realizar web scraping tanto en páginas web estáticas mediante la función read_html como en páginas dinámicas mediante a función read_html_live. Es el paquete más sencillo y más comunmente usado para realizar web scraping en R y el que se usa más extensamente en la presente guía.
- `selenider`: Paquete que permite hacer web scraping y testeo web en R, permitiendo conexiones con los paquetes de R chromote y

`selenium` . Permite una intefaz de usuario más poderosa que `rvest` para tabajar con páginas web dinánimcas. También, se puede integrar con el paquete rvest.

- `chromote`: Chromote es una implementación en R del protocolo Chrome DevTools. Funciona con Chrome, Chromium, Opera, Vivaldi y otros navegadores basados en Chromium. En resumen, permite activar un navegador tipo chromium usarlo para realizar web scraping y testeo web en páginas dinámicas que renderizen usando JavaScript.
- polite: Paquete para hacer uso responsable del web scraping introduciente el cliente (*client*) al servidor (host) e interactua con paquetes como rvest para realizar un web scraping responsable.
- httr2: Permite realizar solicitudes HTTP (*HTTP requests*) y manejar procesos y respuestas a dichas solicitudes desde R.
- selenium: Selenium es una herramienta para la automatización de navegadores web. Es una interfaz de bajo nivel para la especificación WebDriver y una alternativa actualizada a RSelenium.
- RSelenium: Paquete que contiene un conjunto de enlaces en R para Selenium 2.0 Remote WebDriver.

3.5.1.1 `rvest`

El paquete principal que se va a usar para la descarga de las bases de datos vía **web scraping** en R en la presente guía es `rvest`.

! rvest

En palabras del autor, `rvest` le ayuda a extraer (o recolectar) datos de páginas web. Inspirado en bibliotecas como `Beautiful Soup` y `RoboBrowser`.
Si está realizando varias solicitudes de descarga vía **web scraping** en la misma página web, se recomienda utilizar `rvest` junto con el paquete `polite`. El paquete `polite` garantiza que se respete el archivo `robots.txt` de la página y no sobrecarga el sitio con demasiadas solicitudes (Wickham 2024).

i Funciones principales del paquete rvest para realizar web scraping en R

- `read_html`: Web scraping en páginas estáticas dado que opera directamente en el archivo HTML de la página web.
  - `read_html(x)`:
    - Input:
      - `x`: String que representa la URL de la que se quiere hacer la descarga
    - Output:
      - Retorna un objeto `xml_document` que básicamente es una representación del documento HTML de la página web dentro de R. Suele ser mucho más veloz que su contraparte dinámica read_html_live.
- `read_html_live`: Proporciona una interfaz alternativa que ejecuta un navegador web activo (Chrome) en segundo plano. Esto le permite acceder a elementos de la página HTML que se generan dinámicamente mediante javascript e interactuar con la página en vivo haciendo clic en botones o escribiendo formularios. Usa el paquete `chromote` que requiere una instalación de Google Chrome en el computador.
  - `read_html_live(url)`
    - Input:
      - `url`: String que representa la URL de la que se quiere hacer la descarga
    - Output:
      - Retorna un objeto `LiveHTML` que básicamente es un objeto de tipo `R6`. Dicho objeto `LiveHTML` permite hacer web scraping en páginas dinámicas renderizadas vía JavaScript e interactuar emulando clicks dentro de un navegador emulando a un humano.
- `html_elements`: Encuentra elementos de un documento **HTML** utilizando **selectores CSS** o expresiones **XPath**. Los selectores CSS son particularmente útiles junto con https://selectorgadget.com/, lo que hace que sea muy fácil descubrir el selector que se necesita.
  - `html_elements(x, css, xpath)`:
    - Input:
      - `x`: Ya sea un documento html importado a R (importado desde `read_html()`), un conjunto de nodos o un solo nodo (importados desde `read_chidren()`).
      - `css, xpath`: Elementos a seleccionar. Proporcione un **CSS selector** o una ruta **XPath** dependiendo de si se desea utilizar un selector de CSS o una expresión XPath 1.0.

        * Output:
            · Retorna el nodo específico del documento HTML específicado por el `CSS selector` o el `XPath`.

- `html_attr`: Atributo correspondiente al nodo que se especifique en `x`
    - `html_attr(x, name)`
        * Input:
            · `x`: Un documento HTML importado a R (desde `read_html()`), o un conjunto de nodos (desde `html_elements()`), o un nodo (desde `html_element()`)
            · `name`: Nombre del atributo del nodo seleccionado que se desea recuperar.
        * Output:
            · Un vector carácter con el valor que se encuentra en el atributo específicado por `name`

- `html_children`:
    - `html_children(x)`:
        * Input:
            · `x`: Un documento HTML importado a R (desde `read_html()`), o un conjunto de nodos (desde `html_elements()`), o un nodo (desde `html_element()`)
        * Output:
            · Retorna un objeto `xml_nodeset` el cuál contiene todos los nodos hijos directos del nodo específicado en el argumento `x` o el documento `html` expecificado en el argumento `x`.

La Figura 15 muestra un ejemplo de la extracción de un atributo de un nodo específico … de un documento `HTML` importado a R mediante la función `read_html`.

Figura 15: Recuperar el valor dentro del atributo 'href' del nodo específicado por 'a'. Tomado de DataCamp, curso Web Scraping en R.

Se observa de la Figura 15 que primero hay que tener el documento HTML importado a R por la función `read_html`. Luego, hay que ubicar el nodo específico de interés mediante un CSS Selector o un xpath utilizando la función `html_element`. Finalmente, se ubica el valor almacenado en el atributo "`href`" (que hace referencia a un link).

3.6 Código para realizar Web scraping en R

3.6.1 Paquetes necesarios para ejecutar las funciones del archivo `web_scraping_R_funciones_descarga.R` que permiten realizar el Web scraping en R

Se importan los paquetes necesarios para ejectuar las funciones dentro del código.

Descripción de los paquetes que requiere el Script `web_scraping_R_funciones_descarga.R`

- Paquetes de *propósito general*:
  - `tidyverse`**:** Paquete multiproposito de `R` que permite realizar varias tareas, entre ellas, analizar, manipular y gráficar datos.
    - Los siguientes paquetes hacen parte del `tidyverse`:
      1. `dplyr`: Paquete estándar para en análisis y manipulación de bases de datos.
      2. `ggplot2`: Paquete estándar para graficación en `R`.
      3. `tidyr`: Paquete para manipulaciones avanzadas de bases de datos.
      4. `forcats`: Paquete para el manejo de datos categóricos en `R`.
      5. `stringr`: Paquete para el manejo de *strings* (cadenas de carácteres en `R`). Permite entre otras cosas, el manejo de expresiones regulares (`regex`), para buscar y manipular patrones dentro de cadenas de texto.
      6. `readr`: Importar archivos de carácters csv y otros parecidos a R .
      7. `purrr`: Para realizar *programación funcional* en `R`.
      8. `tibble`: Para manipular objetos de tipo tibble, que son dataframes en `R` con funcionalidades adicionales.
    - Informacion adicional:
      - Vídeo de Posit sobre prácticas adecuadas en el tidyverse
- Paquetes adicionales del *tidyverse*:
  - `lubridate`: Paquete para el manejo de fechas y días en R.
  - `glue`: Permite formatear strings de manera avanzada en R. Específicamente, permite introducir y evaluar código de R dentro de los strings similar a como opera los `f-strings` en `python`. Es un mejoramiento de la función `paste0` de R.
- Paquetes para realizar **web scraping** en páginas web *estáticas* (que *no* necesiten `Javascript` para renderizar)
  - `rvest`: Paquete principal para realizar web scraping en R. Hace también parte del universo de conjuntos del *tidyverse.*
  - `xml2`: Paquete diseñado para trabajar archivos `HTML` y `XML` en R
- Paquetes para realizar **web scraping** en páginas web *dinámicas* (que necesiten `Javascript` para renderizar)
  - `RSelenium`: Provee una interfaz para el uso de **Selenium** en R.
- Paquetes adicionales que facilitan el **web scraping** en R
  - `httr2`: Paquete para enviar **HTTP** request a servidores web. Hace también parte del universo de conjuntos del *tidyverse.*
  - `polite`: Paquete para hacer uso responsable del web scraping
- Paquetes estandáres para manejo de archivos de **Excel (.xlsx)** en R
  - `readxl`: Paquete para abrir archivos Excel (.xlsx) en R. Hace también parte del universo de conjuntos del *tidyverse.*
  - `openxlsx`: Paquete alternativo para archivos Excel (.xlsx) en R
  - `data.table`: Paquete que provee funcionalidades adicionales para el análisis y procesamiento de *dataframes* en R.
  - `writexl`: Paquete para exportar dataframes o lista de dataframes de R a Excel.
- Paquetes para manipular y procesar "archivos complejos" de **Excel** que tengan información no rectangular
  - `tidyxl`: Importar datos de Excel sin forzarlos a tener un formato rectangular dentro de R.
  - `unpivotr`: Para lidiar con información no tabular de Excel en R. Contiene un conjunto de funciones extremadamente útil para procesar archivos Excel complejos.
- Paquete para manejo avanzado de series de tiempo en R
  - `xts`: Paquete que permite realizar un manejo avanzado de series de tiempo y facilitar el análisis con éstas. Permite trabajar series de tiempo con frecuencias regulares (e.g. mensuales, trimestrales) y no regulares, como lo pueden ser datos intradiarios que no necesariamente tiene una distancia temporal prestablecida. Muy utilizado en análisis con series de tiempo financieras.

3.6.2 Funciones para el proceso de desacarga de datos vía Web scraping de las diferentes bases

Para cada tipo de `Fuente` de una entidad gubernamental, una agencia de estadística, un gremio o una entidad del sector privado se le asocia una función específica para descargar sus datos. Todas éstas funciones tienen en común los siguientes parámetros:

- `enlace_variable`: Enlace específico donde se encuentra la información o base de datos que se quiere descargar.
- `tag_selector_descarga`: `Selector CSS` o `XPath` del `tag` o nodo específico dentro del documento `HTML` que contiene la información o la basse de datos que se va a descargar.
- `tipo_selector`: Específica si se quiere emplear un `Selector CSS` o un `XPath`
- `nombre_base`: Nombre que se le quiere dar a la base que se va a descargar

Adicionalmente, pueden tener como opcional el argumento:

- `global_url`: Enlace global de la entidad donde se va a descargar la información

Cada uno de los parámetros descritos anteriormente son reemplazados por la información que aparece en el Excel de la Figura 14, donde cada fila es específica para cada base de datos que se quiera descargar. La columna `Fuente` denota cuál función en específico debe usarse para hacer la descarga.

Si bien los argumentos entre las funciones son muy parecidos, la forma en la que se accede el `tag` o nodo específico donde se encuentra la información de descarga puede cambiar, por lo que a continuación se explicará las diferencias entre cada una de las funciones de descarga para cada fuente de información.

3.6.2.1 Función para la descarga de datos del DANE

El código muestra la función para descargar bases de datos del *DANE*. La función acepta tanto selectores `XPath` como `CSS Selectors` e identificando el `tag` que contiene la información a descargar mediante el parámetro `tag_selector_descarga` (que identifica un `tag` de tipo `<a> </a>`) se recupera el enlace de descarga del atributo `href` de dicho `tag` y se descarga la base de datos.

```
# Función para hacer web scraping en el DANE
descarga_DANE = function(global_url, enlace_variable, tag_selector_descarga, tipo_selector, nombre_base){

  # Archivo HTML de la página web importada a R
  html = read_html(enlace_variable)

  # La selección del tag cambia dependiente de si se uso el "xpath" o el "CSS selector" del tag para identificarlo dentro del
  if(tipo_selector == "xpath"){

    # URL del archivo Excel para descargar
    excel_url = html %>%
      html_elements(xpath = tag_selector_descarga) %>%
      html_attr("href")

  }else if(tipo_selector == "css_selector"){

    # URL del archivo Excel para descargar
    excel_url = html %>%
      html_elements(tag_selector_descarga) %>%
      html_attr("href")
  }

  # Descarga del archivo
  download.file(paste0(global_url, excel_url), destfile = nombre_base, mode = "wb")
}
```

3.6.2.2 Función para la descarga de datos de Camacol

El código muestra que la función para descargar bases de datos de *Camacol* tiene exactamente la misma funcionalidad que la función para descargar bases de datos del DANE.

```
# Función para hacer web scraping en Camacol
descarga_Camacol = function(global_url, enlace_variable, tag_selector_descarga, tipo_selector, nombre_base){

  # Archivo HTML de la página web importada a R
  html = read_html(enlace_variable)
```

```
  # La selección del tag cambia dependiente de si se uso el "xpath" o el "CSS selector" del tag para identificarlo dentro del
  if(tipo_selector == "xpath"){

    # URL del archivo Excel para descargar
    excel_url = html %>%
      html_elements(xpath = tag_selector_descarga) %>%
      html_attr("href")

  }else if(tipo_selector == "css_selector"){

    # URL del archivo Excel para descargar
    excel_url = html %>%
      html_elements(tag_selector_descarga) %>%
      html_attr("href")
  }

  # Descargar del archivo
  download.file(paste0(global_url, excel_url), destfile = nombre_base, mode = "wb")

}
```

3.6.2.3 Función para la descarga de datos de la Aerocivil

El código muestra la función para descargar las bases de datos de la *Aerocivil*. La función solo acepta selectores `XPath` dado que fue más sencillo identificar el nodo "padre" usando dicho tipo de identificador de nodos. La línea de código `html_elements(xpath = tag_selector_descarga)` lo que hace es identificar mediante el `XPath` proporcionado en el argumento `tag_selector_descarga` el `tag` *"padre"*, y luego `html_elements("li:first-child")` identifica el primer `tag` hijo tipo `<li> </li>`. Posteriormente, se identifica el `tag` hijo `<div></div>` mediante `html_elements("div:first-child")` y finalmente el `tag` hijo `<a></a>` mediante `html_elements(xpath = "./a[1]")`, es decir, el último `tag` `<a></a>` sería el *"bisnieto"* del tag *"padre"* identificado con `html_elements(xpath = tag_selector_descarga)`.

El código `encoded_url <- URLencode(excel_url)` se usa para preservar la codificación (*encoding*) de la URL, es decir, si la URL tiene carácteres alfanuméricos especiales como lo pueden ser `‘! $ & ' ( ) * + , ; = : / ? @ # [ ]’` los preserva.

Adicionalmente, la función no requiere una `global_url` para hacer la descarga.

```
# Función para hacer web scraping en la Aerocivil
descarga_Aerocivil = function(enlace_variable, tag_selector_descarga, tipo_selector, nombre_base){

  # Archivo HTML de la página web importada a R
  html = read_html(enlace_variable)

  # La selección del tag cambia dependiente de si se uso el "xpath" o el "CSS selector" del tag para identificarlo dentro del
  if(tipo_selector == "xpath"){

    # URL del archivo para descargar
    excel_url = html %>%
      html_elements(xpath = tag_selector_descarga) %>%
      html_elements("li:first-child") %>%
      html_elements("div:first-child") %>%
      html_elements(xpath = "./a[1]") %>%
      html_attr("href")

  }

  # Para que no tenga problemas con la base de datos
  encoded_url <- URLencode(excel_url)

  # Descargar del archivo
  download.file(encoded_url, destfile = nombre_base, mode = "wb")

}
```

3.6.2.4 Función para la descarga de datos de la FNC

El código muestra que la función para descargar bases de datos de la *FNC* tiene exactamente la misma funcionalidad que la funciones para descargar bases de datos del DANE o de Camacol, pero a diferencia de éstas, no requiere una `global_url` para hacer la descarga.

```
# Función para hacer web scraping en la Federación Nacionl de Cafeteros
descarga_FNC = function(enlace_variable, tag_selector_descarga, tipo_selector, nombre_base){

  # Archivo HTML de la página web importada a R
  html = read_html(enlace_variable)

  # La selección del tag cambia dependiente de si se uso el "xpath" o el "CSS selector" del tag para identificarlo dentro del
  if(tipo_selector == "xpath"){

    # URL del archivo Excel para descargar
    excel_url = html %>%
      html_elements(xpath = tag_selector_descarga) %>%
      html_attr("href")

  }else if(tipo_selector == "css_selector"){

    # URL del archivo Excel para descargar
    excel_url = html %>%
      html_elements(tag_selector_descarga) %>%
      html_attr("href")
  }


  # Descarga del archivo
  download.file(excel_url, destfile = nombre_base, mode = "wb")
}
```

3.6.2.5 Función para la descarga de datos de la DIAN

El código muestra la función para descargar bases de datos del *DIAN*. Su estructura inicialmente es exactamente igual a las funciones de descarga del DANE y de Camacol, y al igual que estas, permite tanto los `Selectores CSS` como los `XPath` para la identificación de `tags`. No obstante, dado que el archivo que se descarga de la DIAN es un `.zip` y no un archivo `.xls` o `.xlsx` es necesario agregarle unas funcionalidades adicionales a la función de descarga de la DIAN. La línea `temp <- tempfile()` crea un archivo temporal que almacena el archivo `.zip` luego se descarga el archivo `.zip` y se almacena en el archivo temporal creado previamente, de ahí se descomprime el archivo `.zip` y finalmente se borrar el archivo temporal creado del sistema.

```
# Función para hacer web scraping en la DIAN
descarga_DIAN = function(global_url, enlace_variable, tag_selector_descarga, tipo_selector, nombre_base){

  # Archivo HTML de la página web importada a R
  html = read_html(enlace_variable)

  # La selección del tag cambia dependiente de si se uso el "xpath" o el "CSS selector" del tag para identificarlo dentro del
  if(tipo_selector == "xpath"){

    # URL del archivo Excel para descargar
    excel_url = html %>%
      html_elements(xpath = tag_selector_descarga) %>%
      html_attr("href")

  }else if(tipo_selector == "css_selector"){

    # URL del archivo Excel para descargar
    excel_url = html %>%
      html_elements(tag_selector_descarga) %>%
      html_attr("href")
  }

  # Creción del archivo temporal para almacenar el archivo .zip
  temp <- tempfile()

  # Descarga del archivo ".zip"
  download.file(paste0(global_url, excel_url), temp, mode = "wb")

  # Descompresión del archivo ".zip". Se obtiene el archivo ".xlsx"
```

```
  unzip(zipfile = temp, exdir = "./")

  # Borra el archivo temporal del sistema
  unlink(temp)
}
```

3.6.3 Función para la descarga de todos los datos vía la base da datos `input_web_scraping`

Entendiendo cada una de las funciones de descarga de las diferentes fuentes de información, se procede a explicar la función que itera a través de la hoja **descarga** del Excel **input_web_scraping_MF.xlsx** y que usa la información en cada una de las filas de dicha hoja del archivo Excel para hacer la descarga de la base de datos.

La función tiene tres parámetros:

- **input_descarga**: Base de datos sobre la cual itera la función. En este caso el parámetro va a corresponder a la base de datos que se encuentra en la hoja **descarga** del excel **input_web_scraping_MF.xlsx**.
- **tmp_sleep**: Tiempo de espera entre cada base de datos que va a ser descargada. Es decir, es el tiempo que el sistema se suspende entre cada descarga de bases.
- **tmp_falla**: En caso de que alguna base de datos tenga un error en la descarga el programa se suspende el tiempo específicado por el parámetro y posteriormente vuelve a intentar realizar la descarga.

Inicialmente, la función contiene una función auxiliar llamada **descarga_automatica_bases** cuyo objetivo consiste solo en seleccionar la función correcta para emplear dependiendo del valor que se encuentre en la variable **Fuente** para la fila correspondiente en el Excel a la base de datos que se va a descargar.

Posteriormente, se itera través de las filas de la hoja **descarga** del Excel **input_web_scraping_MF.xlsx** y se usa la información de cada una de las filas sobre las que se está iterando para realizar la descarga dependiente de la función que se seleccione por la específicación de la variable **Fuente** de dicha fila. Dentro del loop hay una estructura de tipo **tryCatch** cuyo proposito consiste en intentar hacer la descarga de la base de datos, y en caso de que haya un error en dicha descarga suspende el programa **tmp_falla** y vuelve a intentar hacer la descarga. Finalmente, luego de realizar la descarga de la base el programa se suspende **tmp_sleep** y pasado ese tiempo continua con la siguiente iteración y vuelve a realizar la descarga de datos.

```
# Función para descargar todas las bases necesarias para correr el modelo
descarga_automatica_bases = function(input_descarga, tmp_sleep, tmp_falla){

  # Función auxiliar descarga: Indica que función se debe utilizar para hacer la descarga, dependiendo de la fuente de la bas
  funcion_auxiliar_descarga = function(input_descarga, i){

    if(input_descarga[i,]$Fuente == "DANE"){

      # Función para descargar bases del DANE
      descarga_DANE(global_url = input_descarga[i,]$global_url,
                    enlace_variable = input_descarga[i,]$enlace_variable,
                    tag_selector_descarga = input_descarga[i,]$tag_selector_descarga,
                    tipo_selector = input_descarga[i,]$tipo_selector,
                    nombre_base = input_descarga[i,]$nombre_base_descarga)

      # Se pausa el programa "tmp_sleep" segudnos para que no haya problemas en la descarga de los archivos
      Sys.sleep(tmp_sleep)

    }else if (input_descarga[i,]$Fuente == "CAMACOL"){

      # Función para descargar bases de Camacol
      descarga_Camacol(global_url = input_descarga[i,]$global_url,
                       enlace_variable = input_descarga[i,]$enlace_variable,
                       tag_selector_descarga = input_descarga[i,]$tag_selector_descarga,
                       tipo_selector = input_descarga[i,]$tipo_selector,
                       nombre_base = input_descarga[i,]$nombre_base_descarga)

      # Se pausa el programa "tmp_sleep" segudnos para que no haya problemas en la descarga de los archivos
      Sys.sleep(tmp_sleep)

    }else if(input_descarga[i,]$Fuente == "AEROCIVIL"){

      # Función para descargar bases Aerocivil
```

```
    descarga_Aerocivil(enlace_variable = input_descarga[i,]$enlace_variable,
                       tag_selector_descarga = input_descarga[i,]$tag_selector_descarga,
                       tipo_selector = input_descarga[i,]$tipo_selector,
                       nombre_base = input_descarga[i,]$nombre_base_descarga)

    # Se pausa el programa "tmp_sleep" segudnos para que no haya problemas en la descarga de los archivos
    Sys.sleep(tmp_sleep)

  }else if(input_descarga[i,]$Fuente == "FNC"){

    # Función para descargar bases FNC
    descarga_FNC(enlace_variable = input_descarga[i,]$enlace_variable,
                 tag_selector_descarga = input_descarga[i,]$tag_selector_descarga,
                 tipo_selector = input_descarga[i,]$tipo_selector,
                 nombre_base = input_descarga[i,]$nombre_base_descarga)

    # Se pausa el programa "tmp_sleep" segudnos para que no haya problemas en la descarga de los archivos
    Sys.sleep(tmp_sleep)

  }else if(input_descarga[i,]$Fuente == "DIAN"){

    # Función para descargar bases DIAN
    descarga_DIAN(global_url = input_descarga[i,]$global_url,
                  enlace_variable = input_descarga[i,]$enlace_variable,
                  tag_selector_descarga = input_descarga[i,]$tag_selector_descarga,
                  tipo_selector = input_descarga[i,]$tipo_selector,
                  nombre_base = input_descarga[i,]$nombre_base_descarga)

    # Se pausa el programa "tmp_sleep" segudnos para que no haya problemas en la descarga de los archivos
    Sys.sleep(tmp_sleep)

  }

}

# Loop que itera a través de la base de datos "input_descarga", y permite descargar las bases de datos una por una vía "web
for (i in 1:nrow(input_descarga)){

  # El TryCatch: Permite que si alguna base en específico tuvo problemas a la hora de hacer la descarga, el programa pause
  ## Nota: Una de las ventajas del "TryCatch" es que si hay una base de datos que tuvo un problema en la descarga, el progr
  ##       base de datos, sin que el programa detenga su ejecución por el error encontrado.
  tryCatch({
    # 1. Try: Intenta descargar la base de datos

    # Se ejecuta la función auxiliar "funcion_auxiliar_descarga" que permite descargar las bases de datos
    funcion_auxiliar_descarga(input_descarga, i)

  }, error = function(e) {
    # 2. Catch: En caso de que haya algún problema en la descarga, se pausa el programa "tmp_falla" segundos y luego se rei

    # Mensaje que indica que hubo un error en la descarga de la base "input_descarga[i,]$nombre_base_descarga"
    print(paste0("Error en la descarga de la base ",
                 input_descarga[i,]$nombre_base_descarga,
                 ". Reintentando realizar la Descarga. Se pausará el programa ",
                 tmp_falla,
                 " segundos y luego del tiempo de espera se reiniciará la descarga."))

    # Tiempo que se suspende el programa, antes de reintentar de nuevo la descarga
    Sys.sleep(tmp_falla)

    # Se vuelve a intentar la ejecución de la función auxiliar "funcion_auxiliar_descarga" que permite descargar la base de
    funcion_auxiliar_descarga(input_descarga, i)

  })

}
```

```
}
```

4 Procesamiento de los datos descargados para la creación de la base de datos usada por los modelos de frecuencias mixtas para el nowcast de PIB

4.1 Tecnologías necesarias para realizar el procesamiento automático de las bases de datos descargadas

4.1.1 Tidy Data

Hoy en día, entidades como Posit (compañía encargada del diseño y distribución del IDE RStudio y de paquetes fundamentales en R como lo es `tidyverse`) y autores como Hadley Wickham (principal desarrollador y supervisor del paquete `tidyverse`), recomiendan que los datos se encuentren en formato ***tidy***. Es así, que hoy en la *ciencia de datos* el estándar para el manejo y procesamiento de los datos requiera que en su mayoría los datos se encuentren en dicho formato ***tidy***.

En su artículo Tidy Data, Hadley Wickham argumenta que para que una base de datos se pueda considerar que está en formato `tidy` se tiene que satisfacer las siguientes 3 caracterísicas dentro de la base (Wickham 2014):

1. Cada variable tiene asociada una columna
2. Cada observación forma una fila
3. Cada tipo de unidad observacional forma una columna

Los conjuntos de datos reales generalmente violan dichos tres principios básicos que definen que una base de datos se encuentre en formato `tidy`, por lo que en muchos casos es necesario transformar la base de datos a este tipo de formato. Las cinco razones principales por las que una base de datos no se encuentra en formato `tidy` son (Wickham 2014):

- Los encabezados de columna son los valores en lugar del nombre de una variable
- Múltiples variables se encuentran almacenadas en una sola columna
- Las variables se encuentran tanto almacenadas en filas como en columnas
- Múltiples tipos de unidades observacionales se encuentran almacenadas en la misma tabla
- Una unidad observacional se encuentra almacenada en múltiples tablas

En caso de que la base de datos presente alguna de las condiciones mencioandas arriba, se hace necesario procesar la base de datos de tal forma que esta pase a un formato `tidy` que facilite el análisis y procesamiento de los datos. Para transformar una base de datos desordenada a formato `tidy`, en R se puede hacer uso de los paquetes `dplyr` y `tidyr`.

Para nuestro caso de interés, los paquetes `tidyxl` y `unpivotr`, que fueron usados para el procesamiento avanzado de bases de datos en ***Excel***, se basan fuertemente en importar las bases de datos en `Excel` a `R` en formato `tidy` tal cual se ha mencionado en esta sesión.

4.1.2 Expresiones regulares (Regex)

Las expresiones regulares (`regex`) es una forma/lenguaje para describir patrones dentro de una cadena de texto (***strings***). Una expresión regular, es simplemente una secuencia de carácteres, donde algunos carácteres tienen un significado específico dentro del contexto de las `regex`. Por ejemplo, el `+`, el `.` , el `^` y el `$` todos tienen un significado dentro de una expresión regular. Adicionalmente, existen reglas específicas para combinar dichas carácteres especiales dentro de una expresión regular para que esta tenga sentido.

La razón de ser de las expresiones regulares, y su importancia en el contexto que se usan en los códigos del presente manual, radica en que facilitan la detección de ciertos simbolos, palabras, frases y demás carácteres dentro de una cadena de texto. Por ejemplo, si yo quiero detectar todos los digitos dentro de una cadena de texto o si en un texto determinado aparece un nombre en específico de interés que quiero recuperar puedo emplear expresiones regulares para ello. En resumen, una expresión regular es una manera sintética y muy útil de encontrar patrones dentro de una cadena de texto.

La sintaxis moderna que se emplea hoy en día en las expresiones regulares fue popularizada por el lenguaje de programación `Perl` en los años 80s, y es la misma sintaxis que usan hoy en días las modernas librerias de expresiones regulares como lo son `stringr` para el caso de `R` y la libreria `re` para el caso de `python`.

> **i** Descripción de algunas de las expresiones regulares más comúnmente usadas
>
> A continuación, se da una lista no exhaustiva de la sintaxis moderna que usan hoy en días las expresiones regulares:
>
> - " `.` " : Comodín para denotar culquier carácter
> - "`\d`" : Digito
> - "`\D`" : No Digito

- "`\t`" : Tabulación
- "`\s`" : Espacio en blanco
- "`\w`" : Palabra (letras, digitos y __)
- " `^` " : Match al Inicio de la expresión regular
- " `$` " : Match al final de la expresión regular
- " `(x | y)` " : Match x o y

Para mayor información acerca de las expresiones regulares y su sintáxis el lenguaje se puede remitir al MIT Regular Expressions Cheat sheet o al DataCamp Regular Expressins Cheat Sheet, donde encontrará una lista más exhaustiva y mayor profundidad sobre las expresiones regulares.

4.2 Base de datos necesaria para realizar el procesamiento automático de los datos que se usan para generar la base final que emplean los modelos de frecuencias mixtas

Como se mencionó en secciones anteriores, dentro del directorio de trabajo `input_web_scraping` se encuentra el archivo `input_web_scraping_MF.xlsx`. Como se mencionó en la Sección 3.6, la hoja `descarga` contiene la información necesaria para descargar las bases de datos de las páginas web vía web scraping. Toda aquella hoja dentro de ese mismo Excel diferente a la hoja descarga contiene la inforamación necesaria para procesar e incluir las variables que se encuentran en dicha hoja dentro de la base de datos final que usan los modelos de frecuencias mixtas para el nowcast de PIB. Por ejemplo, el archivo `input_web_scraping_MF.xlsx` contiene una hoja llamada `procesamiento_bases_M` y otra hoja llamada `procesamiento_bases_T`, donde cada una tiene la información para procesar e incluir las variables mensuales y trimestrales dentro de la base de datos final, respectivamente.

En el caso de los modelos de frecuencias mixtas tiene sentido generar grupos de variables que clasifiquen y agrupen las variables mensuales en un solo grupo y las trimestrales en otro grupo, y por eso están las dos hojas para cada tipo de grupo. Pero es posible hacer desagregaciones más específicas en caso de necesitarse, por ejemplo se podría crear otro hoja llamada `procesamiento_bases_M_ISE` dentro del `Excel` que contenga solo las variables mensuales del ISE. Se recalca la posibilidad de hacer grupos de variables y agruparlos en hojas aparte, porque la agrupación de variables que se haga en el archivo `input_web_scraping_MF.xlsx` será la misma que tendrá la base de datos final. Por ejemplo, si se necesita clasificar las variables en tres hojas diferentes detro del Excel que constituye la base de datos final, entonces tiene que haber tres hojas de procesamiento con la información de las variables que consituyen cada grupo en las hojas correspondientes en el archivo `input_web_scraping_MF.xlsx` .

A continuación, la figura Figura 16, muestra el contenido de la hoja `procesamiento_bases_M` que contiene toda la información necesaria para procesar las variables mensuales que entrarían dentro de la base final que emplean los modelos de frecuencia mixta para el nowcast de PIB. No se muestra la hoja `procesamiento_bases_T` porque tiene exactamente la misma estructura y lógica. Asímismo, cualquier otra hoja adicional del Excel que se cree para agrupar variables que deban ser procesadas para ser parte de la base de dato final, tiene que tener la misma estructura que se muestra en la figura Figura 16.

nombre_variable	nombre_base	nombre_sheet	palabra_identificacion_tabla	filtro1	filtro2	filtro3	Fuente
cafe	FNC_Estadisticas.	9. Producción mer	Mes	Producción			FNC
ipi_carbon	DANE_IPI.xlsx	3. Indices total por	Dominios	Extracción de hulla (carbón de piedra			DANE
ipr_industria	DANE_EMMET_terr	8. Índices Desesta	Dominios	Producción \r\nreal			DANE
ise_ega	DANE_ISE_12_acti	Cuadro 2	Concepto	Suministro de electricidad, gas, vapo			DANE
ini_viv	CAMACOL_Tablas_	Iniciaciones	Fecha	13 Regionale	TOTAL		CAMACOL
desp_conc_edif	DANE_EC.xlsx	Anexo 2	Año	Edificaciones	Vivienda	Total	DANE
desp_conc_obras_c	DANE_EC.xlsx	Anexo 2	Año	Obras Civiles	Total		DANE
ise_comercio	DANE_ISE_12_acti	Cuadro 2	Concepto	Comercio al por mayor y al por menor			DANE
ivr_minorista	DANE_EMC.xlsx	3.1	Año	Total Comercio Minorista sin Combus			DANE
ingresos_hoteles	DANE_EMA.xlsx	Desestacionalizac	Año	Ingresos totales reales			DANE
ise_infocom	DANE_ISE_12_acti	Cuadro 2	Concepto	Información y comunicaciones			DANE
ise_financieras	DANE_ISE_12_acti	Cuadro 2	Concepto	Actividades financieras y de seguros			DANE
ise_inmobiliarias	DANE_ISE_12_acti	Cuadro 2	Concepto	Actividades inmobiliarias			DANE
ise_profesionales	DANE_ISE_12_acti	Cuadro 2	Concepto	Actividades profesionales, científicas			DANE
ise_admin_publica	DANE_ISE_12_acti	Cuadro 2	Concepto	Administración pública y defensa; pla			DANE
ise_artisticas	DANE_ISE_12_acti	Cuadro 2	Concepto	Actividades artísticas, de entretenimi			DANE
imp_indirectos			Año	2. IVA interno (2.1+2.2)			DIAN
ise_ter	DANE_ISE_12_acti	Cuadro 2	Concepto	Actividades terciarias			DANE
impo_capital_real	DANE_Importacio	Cuadro A13	CUODE	Bienes de capital y material de constr			DANE
expom	DANE_Exportacior	Tra y Notra	MES	Total exporta	Miles de Dólares FOB		DANE
impo_bienes_reales	DANE_Importacio	Cuadro A13	CUODE	Total importaciones			DANE

Figura 16: Base de datos necesaria para el procesamiento de las variables mensuales que van a hacer parte de la base de datos final que usarán los modelos de frecuencia mixta para el nowcast de PIB

Variables principales de la base de datos que específica la información necesaria para el procesamiento de las variables que hacen parte de la base final que usan los modelos de frecuencia mixta

Las variables principales de las hojas del archivo `Excel input_web_scraping_MF.xlsx` diferentes a la hoja `descarga` son:

- `nombre_variable`: Nombre de la variable tal cuál va a aparecer en la base final usada por los modelos de frecuencia mixta para el nowcast de PIB sectorial. Dicho nombre de la variable debe ser el mismo que aparece en la base base_final_pre_actualizacion.xlsx para que esta variable pueda ser actualizada de la última base mencionada.
- `nombre_base`: Contiene el nombre de la base de datos que fue descargada vía web scraping y que contiene la variable dentro de ella. Dicho nombre tiene que coincidir exactamente con el nombre que aparezca en la columna `nombre_base_descarga` de la hoja de Excel `descarga` para dicha base de datos.
- `nombre_sheet`: Nombre específico de la hoja dentro del documento `Excel` que fue descargado vía `web scraping` que contiene la variable de interés.
- `palabra_identificacion_tabla`: Necesario para identificar la tabla de datos específica dentro de la hoja del archivo `Excel` descargado que contiene la variable. Es decir, es una palabra que permite identificar la tabla donde se encuentra la variable de interés. En el contexto del código del presente manual, siempre va a corresponder a la primera palabra que se encuentra en la esquina superior izquierda de la tabla. Más adelante se ilustrará un ejemplo aplicado que ilustra mejor el rol de dicha variable.
- Variables de **filtramiento**: Después de haber identificado la tabla específico que contiene la variable de interés, es necesario localizar la posición exacta de dicha variable dentro de la tabla que la contiene. Dicha tarea se complejiza, y se vuelve una tarea *ad hoc* para cada base de datos específica, por dos razones: 1) las bases de datos descargadas de la web no tienen el formato `tidy` mencionado antes y 2) entre una misma institución las bases de datos pueden diferir considerablemente. Por ejemplo, las bases de datos del `DANE` generalmente no tienen el formato `tidy` mencionado previamente y adicionalmente difieren considerablemente entre ellas. Adicionalmente, es usual que una base de datos pueda tener más de un encabezado, en cuyo caso no solo la base de datos deja de tener el formato `tidy` sino también dificultad la identificación de la variable dentro de la tabla que la contiene. De lo anterior, es donde se vuelven relaventes las variables de filtramiento porque permiten identificar la variable en cuestión dentro de la tabla, dado que cada variable de filtramienta está asociada a un encabezado específico de la tabla. Por ejemplo, la base de datos del ISE del DANE si bien tiene tres encabezados: 1) un "encabezado" a la izquierda para denotar el Concepto o tipo de actividad, 2) un "encabezado" en la parte superior que denota el año y 3) un encabezado debajo del encabezado superior que denota el mes, solo necesita especificar en la variable de `filtro1` el valor relacionado al primer encabezado, es decir definir el Concepto o tipo de actividad, para identificar la serie completa. Pero, en otro ejemplo, la tabla dentro de la hoja `Tra y Notra` de la base de datos de

exportaciones del DANE que identifica la variable que contiene la información sobre la cantidad de toneladas métricas exportadas de carbón necesita la información de tres encabezados para poderse identificar: `filtro1`: "Exportaciones tradicionales", `filtro2`: "Carbón" y `filtro3`: "Toneladas métricas". En resumen,

  - `filtro1`: Corresponde al valor del primer encabezado, para identificar la variable.
  - `filtro2`: Corresponde al valor del segundo encabezado, para identificar la variable.
  - `filtro3`: Corresponde al valor del tercer encabezado, para identificar la variable.

  Nota: A fecha del 22 de mayo de 2024, las funciones de procesamiento de datos admiten máximo tres variables de filtro. No obstante, en caso de requerir aumentar el número de filtros esto se puede hacer fácilmente agregando una variable de filtro adicional en el código que define la función. En la actualidad, no se ha encontrado tablas con más de tres encabezados, por lo que no ha sido necesario tener más de tres variables de filtro.

- `Fuente:` Contiene el nombre de la entidad gubernamental, la agencia de estadística, el gremio o la entidad del sector privado de donde proviene la información. Dicha variable es importante, en la medida que idenfica cuál función de procesamiento específica se debe usar para procesar los datos que han sido descargados vía `web scraping`, dado que cada entidad diferente tiene su propia función de procesamiento específica. A fecha del 22 de mayo de 2024, existen funciones de descarga para las siguiente entidades:
  - DANE
  - Camacol
  - Aerocivil
  - FNC
  - DIAN

Cada fila del Excel en la Figura 16, corresponde a una única variable diferente,

es decir, por cada variable nueva que se quiera procesar e incluir en la base de datos final empleada por los modelos de frecuencia mixta es necesario agregar una fila adicional en el Excel específicano la información necesaria para la descarga de dicha base, i.e., hay que especificar en una nueva fila el `nombre_variable`, el `nombre_base`, el `nombre_sheet`, la `palabra_identificacion_tabla`, las variables de filtramiento `filtro1`, `filtro2` y `filtro3` y la `Fuente`, asociados a cada nueva variable que se quiera procesar.

Más adelante, cuando se expliquen los paquetes `tidyxl` y `unpivotr`, se profundizará en mayor detalle como las funciones utilizan la información de las variables `filtro1`, `filtro2` y `filtro3` para identificar las variables de interés.

4.3 Paquetes para realizar el procesamiento automático de los datos descargados en R

4.3.1 Manejo de strings y expresiones regulares en R: `stringr`

El paquete principal para manipular y trabajar con *strings*, y por ende para el manejo de expresiones regulares, en `R` es `stringr`.

! `stringr`

En palabras del autor, stringr proporciona un conjunto coherente de funciones diseñadas para facilitar el trabajo con cadena de carácteres (*strings*) en R.
stringr fue implementado sobre el paquete stringi, y ambos paquetes contienen un conjunto robusto de funciones dedicadas al tratamiento y procesamiento de cadenas de carácteres (strings) en R (Wickham 2023).

i Funciones principales del paquete `stringr` para el manejo de cadenas de carácteres (strings) y expresiones regulares en R

- str_detect:
  - `str_detect(string, pattern, negate = FALSE):`
    - Input:
      - `string`: Vector de carácteres
      - `pattern`: Expresión regular
    - output:
      - Retorna un vector `lógico` que indica si la cadena de carácteres específicada por `pattern` (*la expresión regular*) se encuentra o no en cada uno de los elementos del vector `string`
- str_starts:
  - `str_starts(string, pattern, negate = FALSE):`
    - Input:

            - `string`: Vector de carácteres
            - `pattern`: Expresión regular
        * Output:
            - Retorna un vector `lógico` que indica si la cadena de carácteres específicada por `pattern` (*la expresión regular*) se encuentra o no al inicio del parámetro `string`.

- `str_ends`:
    - `str_ends(string, pattern, negate = FALSE)`:
        * Input:
            - `string`: Vector de carácteres
            - `pattern`: Expresión regular
        * Output:
            - Retorna un vector `lógico` que indica si la cadena de carácteres específicada por `pattern` (*la expresión regular*) se encuentra o no al final del parámetro `string`.

- `str_extract`:
    - `str_extract(string, pattern, group = NULL)`
        * Input:
            - `string`: Vector de carácteres
            - `pattern`: Expresión regular
        * Output:
            - Extrae para cada elemento del vector `string` el primer "match" completo que contenga la cadena de carácteres específicada por la expresión regular en `pattern`

- `str_sub`:
    - `str_sub(string, start = 1L, end = -1L)`:
        * Input:
            - `string`: Vector de carácteres
        * Output:
            - Extrae los elementos del vector `string` que se encuentren en las posiciones específicadas por `start` y `end`

- `str_remove`:
    - `str_ends(string, pattern)`:
        * Input:
            - `string`: Vector de carácteres
            - `pattern`: Expresión regular
        * Output:
            - Retorna el parámetro `string` pero sin la cadena de carácteres específicada por la expresión regular en `pattern`. Es decir, sustituye la cadena de carácteres especificada por la expresión regular en `pattern` por `""`

- `str_replace`:
    - `str_replace(string, pattern, replacement)`:
        * Input:
            - `string`: Vector de carácteres
            - `pattern`: Expresión regular
            - `replacement`: Cadena de carácteres que reemplazará la cadena de carácteres específicada por la expresión regular en `pattern`.
        * Output:
            - Retorna el parámetro `string` pero reemplazando la cadena de carácteres específicada por la expresión regular en `pattern` por la cadena de carácteres que se especifique en el parámetro `replacement`. Es decir, las componentes de `string` que correspondan a la expresión regular en `pattern` se reemplazarán por la cadena de carácteres en `replacement`.

4.3.2 Procesamiento avanzado de datos de Excel en R: `tidyxl` y `unpivotr`

4.3.2.1 `tidyxl`

! `tidyxl`

En palabras del autor, `tidyxl` importa datos que no necesariamente tienen una estructura tabular tipo `tiny` de una hoja de Excel a R. En particular, la base de datos importada expone el contenido de cada celda que hace parte de la hoja de Excel importada, así como su posición, formato y otras características que pueda tener la celda en una estructura ordenada para su posterior manipulación, especialmente manipulaciones que se realizarán con el paquete `unpivotr` (Garmonsway 2023a).

`tidyxl` es la libreria que se usa para la importación de las bases de datos descargadas vía web scraping a R. Si bien, existen otras librerías que permiten importar archivos `.xlsx` a `R` como lo son `readxl` o `openxlsx`, la ventaja de usar la función `xlsx_cells` del mencionado paquete radica en que importa una base de datos en formato `tidy` donde cada fila de la base importada corresponde a una celda distinta de la hoja de Excel que se importa. Es decir, la base importanda tendrá como unidad de observación cada una de las celdas de la hoja de Excel importada. La ventaja de este enfoque, no solo radica en que la base de datos importada ya se encuentra en formato `tidy` sino también en que al estar la base de datos tan desagregada a nivel observacional, se facilita la manipulación y el procesamiento de datos proveniente en Excel y se tiene mayor precisión en la selección de las celdas de la hoja que se importo que contienen la información que se desea recuperar.

i Función principal del paquete `tidyxl` para la importación de archivos `.xlsx` a R

- `xlsx_cells`:
  - Input
    - `path`: *String* con la dirección del archivo `.xlsx` que se va a importar.
    - `sheets`: Vector de carácteres con el nombre de las hojas del `Excel` que se van a importar.
  - output:
    - Retorna un Data Frame cuya unidad observacional, es decir cada fila del Data Frame, corresponde a una celda distinta de la hoja de Excel que se importa. En total, el Data Frame para cada celda (fila) contiene 24 variables que resumen las principales propiedades de dicha celda en particular, como lo son su identificación en Excel (e.g. si es la celda `A1` o `D5`), su fila y su columna correspondiente en Excel, el tipo de dato que contiene, si está vacía o no, si tiene un valor numérico cuál es su valor numérico, si tiene una fecha cuál es su fecha y así sucesivamente, caracterizando la celda correspondiente a dicha fila del Data Frame.

⚠ Formato de los archivos que `tidyxl` permiten importar

`tidyxl` solo permite importar archivos de tipo `.xlsx`. Si el archivo está en formato `.xls`, es necesario primero transformar dicho tipo de archivo a formato `.xlsx` antes de poder ser importado a `R` mediante usar la función `xlsx_cells` . En el código del presente manual la función `verificar_si_es_xls` tiene la funcionalidad para transformar de manera automático archivos de tipo `.xls` a `.xlsx` para que puedan ser importados a R usando la función`xlsx_cells`.

4.3.2.1.1 Ejemplo del uso de la función xlsx_cells del paquete `tidyxl` para la importación de la base de datos `DANE_ISE_12_actividades.xlsx` en R

Se presenta un ejemplo ilustrativo de como utilizar la función xlsx_cells para importar datos en formato `.xlsx` a R y como se vería dentro de R el data frame importado. En el ejemplo, se importa mediante la función xlsx_cells del paquete `tidyxl` la base de datos `DANE_ISE_12_actividades.xlsx` a `R` que contiene toda la información relacionada con las variables del ISE que produce el DANE . Se observa que la base importada a R `DANE_ISE` contiene 13,753 observaciones y 24 variables. La razón por la que la base de datos tiene tantas observaciones, radica en que se importó, en filas diferentes, la información de cada una de las celdas de la hoja de excel importada.

```
# Base de datos que contiene la información del ISE del DANE importada a R mediante la función "xlsx_cells"
DANE_ISE = tidyxl::xlsx_cells(here("bases_datos/output_web_scraping/bases_descargadas/DANE_ISE_12_actividades.xlsx"),
                              sheet = "Cuadro 3")

# Base de datos importada
DANE_ISE
```

```
# A tibble: 13,753 x 24
   sheet    address   row   col is_blank content data_type error logical numeric
   <chr>    <chr>   <int> <int> <lgl>    <chr>   <chr>     <chr> <lgl>     <dbl>
 1 Cuadro 3 D1          1     4 TRUE     <NA>    blank     <NA>  NA           NA
 2 Cuadro 3 D2          2     4 TRUE     <NA>    blank     <NA>  NA           NA
 3 Cuadro 3 A3          3     1 TRUE     <NA>    blank     <NA>  NA           NA
```

```
 4 Cuadro 3 B3          3     2 TRUE     <NA>    blank     <NA>  NA           NA
 5 Cuadro 3 C3          3     3 TRUE     <NA>    blank     <NA>  NA           NA
 6 Cuadro 3 D3          3     4 TRUE     <NA>    blank     <NA>  NA           NA
 7 Cuadro 3 E3          3     5 TRUE     <NA>    blank     <NA>  NA           NA
 8 Cuadro 3 F3          3     6 TRUE     <NA>    blank     <NA>  NA           NA
 9 Cuadro 3 G3          3     7 TRUE     <NA>    blank     <NA>  NA           NA
10 Cuadro 3 H3          3     8 TRUE     <NA>    blank     <NA>  NA           NA
# i 13,743 more rows
# i 14 more variables: date <dttm>, character <chr>,
#   character_formatted <list>, formula <chr>, is_array <lgl>,
#   formula_ref <chr>, formula_group <int>, comment <chr>, height <dbl>,
#   width <dbl>, row_outline_level <dbl>, col_outline_level <dbl>,
#   style_format <chr>, local_format_id <int>
```

Inspeccionando con la función `glimpse(DANE_ISE)` se pueden ver el nombre, el tipo de dato y el contenido de las las 24 variables que conforman el Data Frame importado a R.

```
# Variables de la base de datos que se importó
glimpse(DANE_ISE)
```

```
Rows: 13,753
Columns: 24
$ sheet               <chr> "Cuadro 3", "Cuadro 3", "Cuadro 3", "Cuadro 3", "C~
$ address             <chr> "D1", "D2", "A3", "B3", "C3", "D3", "E3", "F3", "G~
$ row                 <int> 1, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,~
$ col                 <int> 4, 4, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1~
$ is_blank            <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TR~
$ content             <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ data_type           <chr> "blank", "blank", "blank", "blank", "blank", "blan~
$ error               <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ logical             <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ numeric             <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ date                <dttm> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ character           <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ character_formatted <list> <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <NULL>, <~
$ formula             <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ is_array            <lgl> FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, F~
$ formula_ref         <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ formula_group       <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ comment             <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
$ height              <dbl> 9, 9, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, 42, ~
$ width               <dbl> 10.14062, 10.14062, 14.00000, 14.85547, 77.14062, ~
$ row_outline_level   <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
$ col_outline_level   <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
$ style_format        <chr> "Normal", "Normal", "Normal", "Normal", "Normal", ~
$ local_format_id     <int> 10, 10, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31~
```

En particular se destacan las siguientes variables del Data Frame:

- `sheet <variable de tipo carácter>`: Nombre de la hoja en la que se encuentra la celda
- `address <variable de tipo carácter>`: Dirección de la celda en la hoja de Excel (e.g. `A1` o `D5`)
- `row <variable de tipo entero>`: Fila en el Excel donde se encuentra la celda
- `col <variable de tipo entero>`: Columna en el Excel donde se encuentra la celda
- `is_blank <variable de tipo lógico>`: Si la celda es vacía o no
- `content <variable de tipo carácter>`: Contenido dentro de la celda
- `data_type <variable de tipo carácter>`: Tipo de dato dentro de la celda
- `logical <variable de tipo lógica>`: En caso de que la celda contenga un valor lógica, muestra el valor lógico de ésta.
- `numeric <variable de tipo double>`: En caso de que la celda contenga un valor numérico, muestra el valor numérico de ésta.
- `date <variable de tipo date-time>`: En caso de que la celda contenga un valor tipo fecha, muestra el la fecha.
- `character <variable de tipo carácter>`: En caso de que la celda contenga una cadena de texto, muestra el contenido de dicha cadena.

4.3.2.2 `unpivotr`

! `unpivotr`

En palabras del autor, `unpivotr` está diseñado para trabajar con datos provenientes de una hoja de Excel que no necesariamente tienen una estructura tabular tipo `tiny`. El paquete unpivotr es especialmente útil cuándo la fuente de datos que contienen la variable de datos presentan cualquier de las siguientes características (Garmonsway 2023b):

- Tablas con múltiples encabezados.
- Celdas con formato informativo, es decir, el formato de la celda contiene información útil como lo puede ser el color de la celda o si la celda tiene algún tipo de resaltado en su fondo que contenga información que pueda ser útil.
- Encabezados en cualquier parte de la tabla que no necesariamente estén en la parte superior de cada variable de la tabla. Por ejemploe, "encabezados" al lado izquierdo de la tabla.
- Encabezados cuyo contenido no necesariamente es tipo texto. Por ejemplo, el encabezado es de tipo fecha.
- Múltiples tablas en una misma hoja de Excel.
- Comentarios informativos de una celda de Excel.

Conexión entre el paquete `tidyxl` y `unpivotr`

Los paquetes además de ser desarrollados por el mismo autor `Duncan Garmonsway` son complementarios el uno del otro. El paquete `tidyxl` está diseñado para importar datos de `Excel` que no necesariamente tengan carácter tabular tipo `tiny`, mientras que el paquete `unpivotr` está dieñado para procesar datos que no necesariamente tengan carácter tabular tipo `tiny` que ya hayan sido importados a R. Es por ello, que las funciones de `unpivotr` **NECESARIAMENTE** parten de que las bases de datos que entrán como parámetros en ellas hayan sido previamente importadas a R por la función xlsx_cells del paquete `tidyxl`. Esto se debe a que, dichas funciones parten de que las bases de datos no solo se encuentren en formato `tidy` sino además que el contenido de cada celda quede representado en una fila distinta del Data Frame para poder hacer el procesamiento subsecuente de las bases.

i Función principal del paquete `unpivotr` para el procesamiento de bases de datos no necesariamente en formato tipo `tidy` en R

- `partition`: Dadas las posiciones de las esquinas de una tabla de datos dentro de una hoja de Excel, la función filtra aquellas celdas que solo pertenezcan a dicha tabla enmarcada dentro de las esquinas específicadas.
  - `partition(cells, corners, align)`
    - Input:
      - `cells`: Data Frame que representa la totalidad de la hoja de Excel que ha sido importada a R mediante la función `xlsx_cells` y por ende cada fila de dicho Data Frame es una celda diferente.
      - `corners`: Vector de carácteres (*`strings`*) que específica el nombre o la cadena de texto que se encuentra en cada una de las celdas esquina. Se requiere específicar como mínimo una esquina y se puede específicar hasta máximo las cuatro esquinas.
      - `align`: Vector de carácteres (*`strings`*) que específica la posición de cada una de las celdas mencionadas en el parámetro `corner` respecto a la tabla que se está identificando. Las opciones posibles son `"top_left"` (default), `"top_right"`, `"bottom_left"`, `"bottom_right"`.
    - Output :
      - Retorna un Data Frame donde cada fila representa una tabla de celdas diferente seleccionadas por la función. Lo anterior es posible, dado que la columna `cells` de dicho Data Frame resultante es de tipo `list`, y por ende es capaz de almacenar toda la información de la tabla seleccionada en dicha columna. En resumen, cada fila del Data Frame representa una tabla diferente que la función seleccionó.
- `behead`: Toma un encabezado a la vez de una tabla representado por el parámetro `cells` y lo transforma para que sea ahora una nueva columna del Data Frame en lugar de un encabezado.
  - `behead(cells, direction, name)`
    - Input:
      - `cells`: Data Frame que representa un conjunto de celdas de una hoja de Excel que ha sido importada a R mediante la función xlsx_cells o que haya sido el resultado de algún procesamiento de dicha base de datos como el que se pueda dar con la función `partition`.
      - `direction`: La dirección entre las celdas de datos y el encabezado. Existen las siguientes opciones `"up"`, `"right"`, `"down"`, `"left"`, `"up-left"`, `"up-right"`, `"right-up"`, `"right-down"`, `"down-right"`, `"down-left"`, `"left-down"`, `"left-up"` dependiendo de la posición del encabezado dentro de la tabla.

- `name`: El nombre que se le va a dar a la nueva columna que resultará de transforma el encabezado a una columna nueva del Data Frame.

* Output:
  - Mismo Data Frame que se específico en el parámetro `cells` de la función, pero con una columna adicional que será la variable correspondiente al encabezado que se transformó en variable. El nombre de la variable adicional que se adicionó el Data Frame será el específicado en el parámetro `name`.

Nota: La función `behead` opera en un encabezado a la vez de la tabla. Por ejemplo, si la tabla tiene tres encabezados, entonces es necesario aplicar la función `behead` en tres ocasiones distintas, una por cada encabezado de la tabla.

4.3.2.2.1 Ejemplo1: Uso del paquete `unpivotr` - Extracción de la variable ISE Información y comunicaciones del `DANE`

El presente ejemplo muestra el proceso completo necesario para extraer como serie de tiempo la variable ISE Información y comunicaciones desde la importación del Excel que contiene la variable, hasta la consolidación de la variable en formato de serie de tiempo.

La Figura 17 muestra una de las tres tablas que aparecen en la hoja `Cuadro 2` del archivo Excel `DANE_ISE_12_actividades.xlsx`. Es importante destacar que las tres tablas que aparecen en dicha hoja de Excel, todas tienen en la esquina superior izquierda la palabra "Concepto" (resaltada en azul) que permite identificar cada una de las tablas dentro de la hoja del Excel. Resaltados en naranja, se encuentran los tres encabezados los cuáles son: encabezado 1) La clasificación específica del ISE (el concepto) que se encuentra en la parte izquierda de la tabla, encabezado 2) el año específico que se encuentra en la parte superior de la tabla y el encabezado 3) el mes que también se encuentra en la parte superior de la tabla, pero debajo del año. Resaltado en verde, se encuentra el valor del primer encabezado que permite identificar el ISE de información y comunicaciones.

Clasificación Cuentas Nacionales	Secciones CIIU Rev. 4 A.C. 12 agrupaciones	Concepto	2005												
			Enero	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre	Enero	Febrero
		Actividades primarias	**66.0**	**66.5**	**66.7**	**67.1**	**66.1**	**68.5**	**65.3**	**67.3**	**67.1**	**66.0**	**65.9**	**68.3**	**67.4**	**67.8**
	A	Agricultura, ganadería, caza, silvicultura y pesca	79.9	79.2	78.6	78.3	76.6	81.0	76.1	79.4	79.0	78.2	79.1	79.8	77.9	79.8
	B	Explotación de minas y canteras	56.3	56.7	58.2	58.7	57.0	60.4	57.2	57.5	57.1	56.8	55.5	59.7	61.0	59.0
		Actividades secundarias	**69.4**	**69.5**	**69.1**	**69.2**	**66.1**	**67.6**	**67.1**	**68.3**	**67.8**	**67.5**	**68.3**	**73.1**	**71.4**	**72.8**
	C	Industrias manufactureras	75.9	77.5	78.4	78.0	75.9	78.6	76.0	78.8	76.6	75.7	77.0	80.7	78.8	80.6
	F	Construcción	58.3	52.9	50.8	49.0	47.4	46.8	47.5	49.1	51.1	51.8	53.6	58.0	58.0	57.0
		Actividades terciarias	**61.8**	**61.9**	**62.0**	**63.0**	**62.8**	**62.8**	**62.9**	**63.2**	**63.3**	**63.9**	**64.0**	**64.5**	**65.1**	**65.6**
	D + E	Suministro de electricidad, gas, vapor y aire acondicionado; Distribución de agua; Evacuación y tratamiento de aguas residuales, gestión de desechos y actividades de saneamiento ambiental	74.5	75.4	75.4	76.2	75.4	76.4	76.1	77.1	76.9	74.7	76.1	77.3	77.6	79.1
	G + H + I	Comercio al por mayor y al por menor; Reparación de vehículos automotores y motocicletas; Transporte y almacenamiento; Alojamiento y servicios de comida	62.1	62.6	61.8	64.2	62.9	62.3	63.0	63.3	63.0	64.1	62.7	65.0	65.2	65.6
	J	Información y comunicaciones	49.0	50.1	51.8	52.6	52.9	51.8	47.9	50.6	52.6	57.5	61.0	59.8	59.3	58.7
	K	Actividades financieras y de seguros	42.6	42.7	42.4	44.4	44.3	45.3	45.8	46.4	46.5	43.8	45.4	43.9	46.7	47.4
	L	Actividades inmobiliarias	70.9	71.2	71.4	71.5	71.7	71.8	72.0	72.1	72.3	72.5	72.7	73.1	73.4	73.6
	M + N	Actividades profesionales, científicas y técnicas; Actividades de servicios administrativos y de apoyo	60.3	61.8	62.0	62.4	62.5	62.6	62.8	63.1	63.2	64.0	64.2	64.9	65.4	65.6
	O + P + Q	Administración pública y defensa; planes de seguridad social de afiliación obligatoria; Educación; Actividades de atención de la salud humana y de servicios sociales	64.2	63.0	62.3	63.5	63.9	63.9	63.7	63.5	63.4	63.5	63.6	63.7	64.9	65.8
	R + S + T	Actividades artísticas, de entretenimiento y recreación y otras actividades de servicios; Actividades de los hogares individuales en calidad de empleadores; Actividades no diferenciadas de los hogares individuales como productores de bienes y servicios para uso propio	66.9	65.4	65.8	66.1	66.5	66.8	67.6	68.1	68.2	68.4	69.3	68.6	69.0	69.4
		Indicador de Seguimiento a la Economía	**63.9**	**64.1**	**64.0**	**64.9**	**63.9**	**64.5**	**64.0**	**65.0**	**64.8**	**65.0**	**65.3**	**67.0**	**66.7**	**67.3**

Figura 17: Ejemplo de tabla que se encuentra en la hoja Cuadro 2 del archivo del DANE que contiene la información del ISE de 12 sectores.

En primer lugar, se específican los argumentos necesarios para poder usar el paquete `unpivotr` para realizar la extracción de la serie de tiempo del `ISE` de `información y comunicaciones`.

```
nombre_variable = "ise_infocom"
nombre_base = "DANE_ISE_12_actividades.xlsx"
nombre_sheet = "Cuadro 2"
palabra_identificacion_tabla = "Concepto"
filtro1 = "Información y comunicaciones"
```

Luego, se importa la base de datos usando la función xlsx_cells del paquete tidyxl.

```
# Base de datos que contiene la información del ISE del DANE importada a R mediante la función "xlsx_cells"
base = tidyxl::xlsx_cells(here(glue("bases_datos/output_web_scraping/bases_descargadas/{nombre_base}")), sheet = nombre_sheet

# Base de datos que fue importada a R en formato tidy por la función "xlsx_cells"
base
```

```
# A tibble: 13,752 x 24
   sheet    address   row   col is_blank content data_type error logical numeric
   <chr>    <chr>   <int> <int> <lgl>    <chr>   <chr>     <chr> <lgl>     <dbl>
 1 Cuadro 2 D1          1     4 TRUE     <NA>    blank     <NA>  NA           NA
```

```
 2 Cuadro 2 D2          2     4 TRUE     <NA>    blank     <NA>  NA           NA
 3 Cuadro 2 A3          3     1 TRUE     <NA>    blank     <NA>  NA           NA
 4 Cuadro 2 B3          3     2 TRUE     <NA>    blank     <NA>  NA           NA
 5 Cuadro 2 C3          3     3 TRUE     <NA>    blank     <NA>  NA           NA
 6 Cuadro 2 D3          3     4 TRUE     <NA>    blank     <NA>  NA           NA
 7 Cuadro 2 E3          3     5 TRUE     <NA>    blank     <NA>  NA           NA
 8 Cuadro 2 F3          3     6 TRUE     <NA>    blank     <NA>  NA           NA
 9 Cuadro 2 G3          3     7 TRUE     <NA>    blank     <NA>  NA           NA
10 Cuadro 2 H3          3     8 TRUE     <NA>    blank     <NA>  NA           NA
# i 13,742 more rows
# i 14 more variables: date <dttm>, character <chr>,
#   character_formatted <list>, formula <chr>, is_array <lgl>,
#   formula_ref <chr>, formula_group <int>, comment <chr>, height <dbl>,
#   width <dbl>, row_outline_level <dbl>, col_outline_level <dbl>,
#   style_format <chr>, local_format_id <int>
```

Posteriormente, se identifica la esquina superior izquierda de la tabla, que en este caso es la celda que contiene la palabra "Concepto"

```
# Identificación de la esquina superior izquierda de la tabla
esquina_sup_izq_tabla = dplyr::filter(base, character == palabra_identificacion_tabla)

# Celdas que contiene la palabra "palabra_identificacion_tabla"
esquina_sup_izq_tabla
```

```
# A tibble: 3 x 24
  sheet    address   row   col is_blank content data_type error logical numeric
  <chr>    <chr>   <int> <int> <lgl>    <chr>   <chr>     <chr> <lgl>     <dbl>
1 Cuadro 2 C11        11     3 FALSE    12      character <NA>  NA           NA
2 Cuadro 2 C45        45     3 FALSE    12      character <NA>  NA           NA
3 Cuadro 2 C79        79     3 FALSE    12      character <NA>  NA           NA
# i 14 more variables: date <dttm>, character <chr>,
#   character_formatted <list>, formula <chr>, is_array <lgl>,
#   formula_ref <chr>, formula_group <int>, comment <chr>, height <dbl>,
#   width <dbl>, row_outline_level <dbl>, col_outline_level <dbl>,
#   style_format <chr>, local_format_id <int>
```

Como se puede observar hay tres celdas con dicha palabra, dado que hay tres tablas en la mencionada hoja de Excel que contiene en su esquina superior izquierda esta palabra.

Ahora bien, usamos la función `partition` para seleccionar solo las celdas que se encuentran dentro las tres tablas que contienen en su esquina superior izquierda la palabra Concepto

```
# Partición de la base que contiene la informacion de la tabla de interés dentro de la hoja de excel
particion = partition(base, esquina_sup_izq_tabla)

# Tablas de la hoja de Excel que fuerón seleccionados por la función partition, dado que en su esquina superior izquierda con
particion
```

```
# A tibble: 3 x 25
  corner_row corner_col cells    sheet  address is_blank content data_type error
       <dbl>      <dbl> <list>   <chr>  <chr>   <lgl>    <chr>   <chr>     <chr>
1         11          3 <tibble> Cuadr~ C11     FALSE    12      character <NA>
2         45          3 <tibble> Cuadr~ C45     FALSE    12      character <NA>
3         79          3 <tibble> Cuadr~ C79     FALSE    12      character <NA>
# i 16 more variables: logical <lgl>, numeric <dbl>, date <dttm>,
#   character <chr>, character_formatted <list>, formula <chr>, is_array <lgl>,
#   formula_ref <chr>, formula_group <int>, comment <chr>, height <dbl>,
#   width <dbl>, row_outline_level <dbl>, col_outline_level <dbl>,
#   style_format <chr>, local_format_id <int>
```

Luego, se selecciona la primer tabla de celdas, dado que solo nos interesa dicha tabla que contiene la información del ISE en nivel y no las tasas de crecimiento que es el contenido de las otras dos tablas. Para dicha selección del Data Frame particion se selecciona la columna `cells` que contiene la información completa de la tabla y el `[[1]]` indica que solo estamos interesados en la primera fila, que equivale a la primera tabla de interés.

```
#Se selecciona solo la primera tabla cuya esquina superior izquierda contiene la palabra en la variable "esquina_sup_izq_tabl
tabla1 = particion$cells[[1]]


tabla1
```

```
# A tibble: 4,243 x 24
   sheet    address   row   col is_blank content data_type error logical numeric
   <chr>    <chr>   <int> <int> <lgl>    <chr>   <chr>     <chr> <lgl>     <dbl>
 1 Cuadro 2 C11        11     3 FALSE    12      character <NA>  NA           NA
 2 Cuadro 2 D11        11     4 FALSE    2005    numeric   <NA>  NA         2005
 3 Cuadro 2 E11        11     5 TRUE     <NA>    blank     <NA>  NA           NA
 4 Cuadro 2 F11        11     6 TRUE     <NA>    blank     <NA>  NA           NA
 5 Cuadro 2 G11        11     7 TRUE     <NA>    blank     <NA>  NA           NA
 6 Cuadro 2 H11        11     8 TRUE     <NA>    blank     <NA>  NA           NA
 7 Cuadro 2 I11        11     9 TRUE     <NA>    blank     <NA>  NA           NA
 8 Cuadro 2 J11        11    10 TRUE     <NA>    blank     <NA>  NA           NA
 9 Cuadro 2 K11        11    11 TRUE     <NA>    blank     <NA>  NA           NA
10 Cuadro 2 L11        11    12 TRUE     <NA>    blank     <NA>  NA           NA
# i 4,233 more rows
# i 14 more variables: date <dttm>, character <chr>,
#   character_formatted <list>, formula <chr>, is_array <lgl>,
#   formula_ref <chr>, formula_group <int>, comment <chr>, height <dbl>,
#   width <dbl>, row_outline_level <dbl>, col_outline_level <dbl>,
#   style_format <chr>, local_format_id <int>
```

Luego, se filtran todas las celdas de dicha tabla de interés que no sean vacías y posteriormente se emplea de manera sucesiva la función behead para transformar los encabezados de la tabla en columnas del Data Frame `tabla`. El primer llamado de la función `behead` transforma el encabezado de años en una variable llamada año, el segundo llamado de la función `behead` transforma el encabazado de meses en una variable llamada meses y el tercer llamado de la función `behead` transforma el encabezado que denota el tipo de actividad del ISE en una variable que se llama actividad, de tal forma que la tabla resultante ya no tenga encabezados y en su lugar tenga tres columnas adicionales que contienen la información de los encabezados para cada celda.

```
# Selección de las celdas de Excel que contienen la info de la tabla
tabla = tabla1 %>%
  filter(!is_blank) %>%
  behead("up-left", "año") %>%
  behead("up", "mes") %>%
  behead("left", "actividad") %>%
  select(row, col, data_type, numeric, character, date, año, mes, actividad)

# Tabla resultante de transformar los encabezados a variables dentro del Data Frame
tabla
```

```
# A tibble: 3,696 x 9
     row   col data_type numeric character date   año   mes
   <int> <int> <chr>       <dbl> <chr>     <dttm> <chr> <chr>
 1    14     4 numeric      66.0 <NA>      NA     2005  Enero
 2    14     5 numeric      66.5 <NA>      NA     2005  Febrero
 3    14     6 numeric      66.7 <NA>      NA     2005  Marzo
 4    14     7 numeric      67.1 <NA>      NA     2005  Abril
 5    14     8 numeric      66.1 <NA>      NA     2005  Mayo
 6    14     9 numeric      68.5 <NA>      NA     2005  Junio
 7    14    10 numeric      65.3 <NA>      NA     2005  Julio
 8    14    11 numeric      67.3 <NA>      NA     2005  Agosto
 9    14    12 numeric      67.1 <NA>      NA     2005  Septiembre
10    14    13 numeric      66.0 <NA>      NA     2005  Octubre
# i 3,686 more rows
# i 1 more variable: actividad <chr>
```

Posteriormente, se utiliza la información en la variable `filtro1` para identificar específicamente la serie que contiene la información sobre el `ISE de información y comunicaciones`. Para ello, se filtra la variable `actividad` para aquellas celdas que correspondan al `ISE de información y comunicaciones` y se seleccionan solo los valores numéricos.

```r
# Selección de la columna con la información de interés
tabla_selection = tabla %>%
  filter(actividad == filtro1) %>%
  filter(!is.na(numeric))

# Tabla que contiene solo la información de la variable de interés, luego de haber aplicado el filtro
tabla_selection
```

```
# A tibble: 231 x 9
     row   col data_type numeric character date   año   mes
   <int> <int> <chr>       <dbl> <chr>     <dttm> <chr> <chr>
 1    23     4 numeric      49.0 <NA>      NA     2005  Enero
 2    23     5 numeric      50.1 <NA>      NA     2005  Febrero
 3    23     6 numeric      51.8 <NA>      NA     2005  Marzo
 4    23     7 numeric      52.6 <NA>      NA     2005  Abril
 5    23     8 numeric      52.9 <NA>      NA     2005  Mayo
 6    23     9 numeric      51.8 <NA>      NA     2005  Junio
 7    23    10 numeric      47.9 <NA>      NA     2005  Julio
 8    23    11 numeric      50.6 <NA>      NA     2005  Agosto
 9    23    12 numeric      52.6 <NA>      NA     2005  Septiembre
10    23    13 numeric      57.5 <NA>      NA     2005  Octubre
# i 221 more rows
# i 1 more variable: actividad <chr>
```

A continuación, se identifican el primer año y el primer mes, desde donde inicia la serie de tiempo de la variable de interés

```r
# Selección del año y mes de inicio

## Año inicial
year_init = tabla_selection$año[1]; year_init
```

```
[1] "2005"
```

```r
## Mes inicial
mes_init = deteccion_primer_mes(tabla_selection$mes[1]); mes_init
```

```
[1] 1
```

Nota: En el paso anterior se uso la función `deteccion_primer_mes` que lo que permite es usando expresiones regulares transformar un mes escrito de manera convencional, por ejemplo Enero, en un valor numérico, por ejemplo 1. En secciones posteriores se explicará la función `deteccion_primer_mes`.

Finalmente, se crea el objeto de serie de tiempo tipo xts, que la librería que se usa en los códigos del presente manual para trabjar series de tiempo en R. Para ello, incialmente se seleccionan los valores numéricos de la serie de interés

```r
# Inicialmente, se seleccionan los valores numéricos de la serie de interés
valores_numericos_serie = tabla_selection$numeric

valores_numericos_serie
```

```
  [1]  48.97393  50.10735  51.84200  52.56708  52.87853  51.81394  47.91359
  [8]  50.57767  52.57047  57.53060  60.98117  59.77967  59.33344  58.72037
 [15]  60.36291  62.86560  63.04175  62.49175  59.64123  58.99113  59.04566
 [22]  60.96328  64.43701  62.74642  66.05355  67.19921  67.04543  66.06993
 [29]  66.98724  70.25578  73.46018  74.60028  73.93930  71.17806  69.50421
 [36]  73.12088  70.79422  72.22662  70.37297  70.89475  70.83733  72.22405
 [43]  74.14788  74.01142  73.79341  72.56960  68.35963  67.16140  69.24530
 [50]  66.17199  66.32207  66.03885  65.39864  64.22935  61.94788  62.01342
 [57]  63.84901  64.76385  66.55081  68.19400  68.53429  70.90708  73.36820
 [64]  75.34395  76.87874  77.02654  77.04683  78.06707  78.77583  78.69107
 [71]  79.97543  79.67069  81.64249  82.05277  82.70300  82.21931  82.30422
 [78]  83.23490  86.06981  87.07592  84.90456  85.76837  85.84376  85.47098
 [85]  83.62636  84.53351  84.16132  83.22157  84.17644  84.05878  84.20683
```

```
 [92]  84.67796  85.86746  86.66440  87.73676  89.48004  90.56320  90.65152
 [99]  88.95205  90.50434  91.32104  92.42247  93.51532  94.20522  96.62940
[106]  94.25006  94.44740  95.24613  96.76220  99.13449  99.15697 100.64769
[113] 100.04721  98.82263  97.25683  96.39867  97.66295  99.23369  99.51018
[120]  99.99150  99.77742  98.53221  98.91135  98.90368  98.61810  99.01048
[127] 100.64841 101.00065 101.57151 102.11643 100.62353 100.28622 103.61698
[134]  96.95774  97.27922  99.48585  99.43521  99.57516 100.67114 100.50067
[141]  99.60252  99.36865  98.26257  97.38152  98.72266  97.66925  99.43641
[148]  97.78469 101.24304  99.19681  97.71042  96.37230  98.21054  96.82849
[155]  99.60445 107.05442  98.03696  99.94487 100.15945 101.69542 101.55553
[162] 102.63184 103.07899 105.05668 104.64621 104.69718 102.21684 108.03164
[169] 102.76071 101.45583 102.94322 103.58056 105.02022 105.23656 103.05425
[176] 102.66922 103.07395 104.57426 104.28940 104.41182 102.87844 102.43562
[183] 100.75183  99.84677  98.08940  97.47867 103.31833 100.65955  99.20361
[190]  97.92481 101.56321 104.46375 103.91343 106.42300 108.05271 109.58700
[197] 110.41634 111.94923 114.54900 115.11986 117.80372 118.54802 123.02696
[204] 123.32514 126.36951 127.26093 128.51567 127.77392 129.00865 128.93210
[211] 130.19681 129.50342 129.25602 121.66079 125.70137 127.60997 129.41993
[218] 129.25650 129.80021 129.69269 129.39326 129.89632 127.62929 128.85971
[225] 129.27609 133.78473 129.72283 127.45817 128.02089 125.91818 126.81426
```

y luego a partir de los valores numéricos seleccionados se construye el objeto de serie de tiempo como tal. El nombre de la serie será el que aparece en la variable `nombre_variable` definida arriba.

```
# Creación del objeto ts
ts_obj = ts(valores_numericos_serie, start = c(year_init, mes_init), frequency = 12)

# Creación del objeto xts
xts_obj = as.xts(ts_obj)
colnames(xts_obj) = c(nombre_variable)

# Objeto xts con la información de interés
xts_obj
```

```
         ise_infocom
Jan 2005    48.97393
Feb 2005    50.10735
Mar 2005    51.84200
Apr 2005    52.56708
May 2005    52.87853
Jun 2005    51.81394
Jul 2005    47.91359
Aug 2005    50.57767
Sep 2005    52.57047
Oct 2005    57.53060
     ...
Jun 2023   129.89632
Jul 2023   127.62929
Aug 2023   128.85971
Sep 2023   129.27609
Oct 2023   133.78473
Nov 2023   129.72283
Dec 2023   127.45817
Jan 2024   128.02089
Feb 2024   125.91818
Mar 2024   126.81426
```

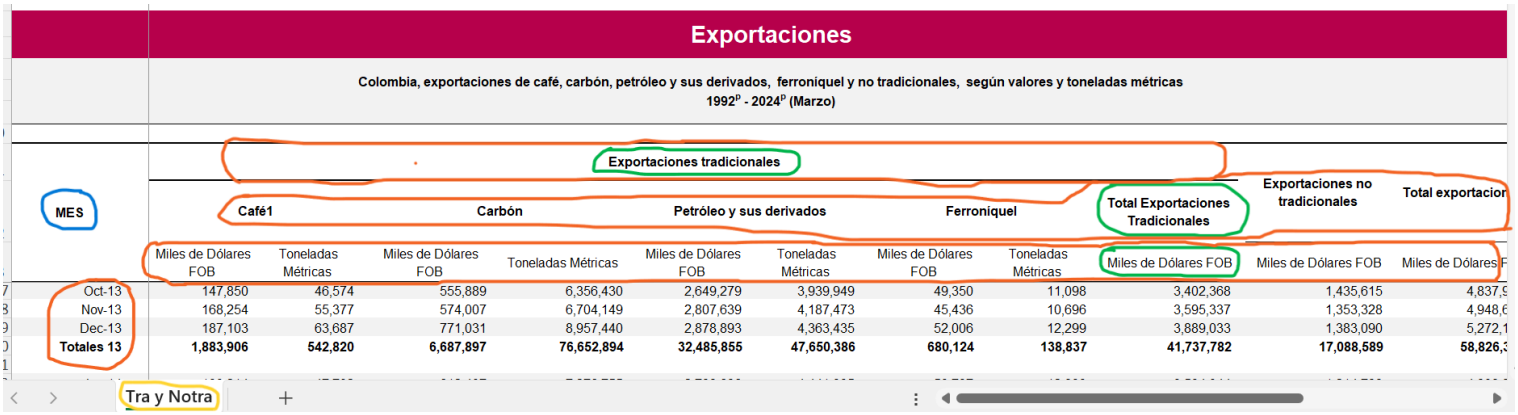
Es así, como se puede recuperara de manera automática la información en el Excel `DANE_ISE_12_actividades.xlsx`, extraer el ISE de información y comunicaciones y construir la serie de tiempo en R con la información de dicha variable.

4.3.2.2.2 Ejemplo2: Uso del paquete `unpivotr` - Extracción de la variable exportaciones del `DANE`

A continuación, se presenta un segundo ejemplo para adquirir mayor familiaridad con el procedimiento de usar el paquete `unpivotr` para extraer una variable específica de un archivo de `Excel` y transformarla en una serie de tiempo en R. Para ello se trabajará ahora con la base

de datos de exportaciones del DANE y se extraerá la variable de total de exportaciones tradicionales en miles de dólares FOB de la hoja `Tra y Notra`.

La figura Figura 18 muestra la tabla que se encuentra en la hoja `Tra y Notra` del Excel de exportaciones del DANE. Resaltada en azul, se específica la celda que corresponde a la esquina superior izquierda de la tabla que contiene la palabra "MES" que permite idnetificar la tabla dentro de la hoja de Excel. Resaltados en naranja, se encuentran los cuatro encabezados los cuáles son: encabezado 1) encabezado que agrupa a todas las exportaciones tradicionales, encabezado 2) que clasfica las exportaciones tradicionales y también incluye las exportaciones no tradicionales y el total exportación y el encabezado 3) que clasifica las exportaciones entre miles de dólares FOB y toneladas métricas y encabezado 4) que contiene las fechas. Resaltado verde, se encuentran los valores del primer, segundo y tercer encabezado que permiten identificar el total de exportaciones tradicionales en miles de dólares FOB.



Exportaciones

Colombia, exportaciones de café, carbón, petróleo y sus derivados, ferroniquel y no tradicionales, según valores y toneladas métricas
1992[p] - 2024[p] (Marzo)

MES	Exportaciones tradicionales									Exportaciones no tradicionales	Total exportacion
	Café1		Carbón		Petróleo y sus derivados		Ferroniquel		Total Exportaciones Tradicionales		
	Miles de Dólares FOB	Toneladas Métricas	Miles de Dólares FOB	Toneladas Métricas	Miles de Dólares FOB	Toneladas Métricas	Miles de Dólares FOB	Toneladas Métricas	Miles de Dólares FOB	Miles de Dólares FOB	Miles de Dólares F
Oct-13	147,850	46,574	555,889	6,356,430	2,649,279	3,939,949	49,350	11,098	3,402,368	1,435,615	4,837,9
Nov-13	168,254	55,377	574,007	6,704,149	2,807,639	4,187,473	45,436	10,696	3,595,337	1,353,328	4,948,6
Dec-13	187,103	63,687	771,031	8,957,440	2,878,893	4,363,435	52,006	12,299	3,889,033	1,383,090	5,272,1
Totales 13	**1,883,906**	**542,820**	**6,687,897**	**76,652,894**	**32,485,855**	**47,650,386**	**680,124**	**138,837**	**41,737,782**	**17,088,589**	**58,826,3**

Tra y Notra

Figura 18: Tabla que se encuentra en la hoja `Tra y Notra` del archivo del DANE que contiene la información de exportaciones.

En primer lugar, se específican los argumentos necesarios para poder usar el paquete `unpivotr` para realizar la extracción de la serie de tiempo del total de exportaciones tradicionales en miles de dólares FOB.

```
nombre_variable = "expo_tradicionales"
nombre_base = "DANE_Exportaciones.xlsx"
nombre_sheet = "Tra y Notra"
palabra_identificacion_tabla = "MES"
filtro1 = "Exportaciones tradicionales"
filtro2 = "Total Exportaciones Tradicionales"
filtro3 = "Miles de Dólares FOB"
```

Desde la importación de la base de datos usando la función xlsx_cells del paquete tidyxl hasta la identificación de la tabla que contiene las celdas de interés de la hoja de Excel importada, el procedimiento es exactamente le mismo que el del ejemplo anterior del `ISE` de `información y comunicaciones`.

```
# Base de datos que contiene la información del ISE del DANE importada a R mediante la función "xlsx_cells"
base = tidyxl::xlsx_cells(here(glue("bases_datos/output_web_scraping/bases_descargadas/{nombre_base}")), sheet = nombre_sheet

# Identificación de la esquina superior izquierda de la tabla
esquina_sup_izq_tabla = dplyr::filter(base, character == palabra_identificacion_tabla)

# Partición de la base que contiene la informacion de la tabla de interés dentro de la hoja de excel
particion = partition(base, esquina_sup_izq_tabla)

# Tabla de la hoja de Excel que fue seleccionada por la función partition, dado que en su esquina superior izquierda contiene
particion
```

```
# A tibble: 1 x 25
  corner_row corner_col cells    sheet  address is_blank content data_type error
       <dbl>      <dbl> <list>   <chr>  <chr>   <lgl>    <chr>   <chr>     <chr>
1         11          1 <tibble> Tra y~ A11     FALSE    47      character <NA>
# i 16 more variables: logical <lgl>, numeric <dbl>, date <dttm>,
#   character <chr>, character_formatted <list>, formula <chr>, is_array <lgl>,
#   formula_ref <chr>, formula_group <int>, comment <chr>, height <dbl>,
#   width <dbl>, row_outline_level <dbl>, col_outline_level <dbl>,
#   style_format <chr>, local_format_id <int>
```

El Data Frame `particion`, contiene toda la información relevante de la tabla que contiene la variable de interés. En específico, la columna `cells`, que es una variable tipo `list` dentro del Data Frame, contiene las celdas que conforman dicha tabla.

Ahora bien, de acá en adelante si difiere el código frente al que se presento en el ejemplo anterior del procesamiento del `ISE` de `información y comunicaciones`. Esto se debe a que como lo muestran la Figura 17 y la Figura 18 las tablas de los dos ejemplos difieren tanto en forma como en sus encabezados. Se filtran todas las celdas de dicha tabla de interés que no sean vacías y posteriormente se emplea de manera sucesiva la función behead para transformar los encabezados de la tabla en columnas del Data Frame `tabla`. El primer llamado de la función `behead` transforma el primer encabezado de exportaciones en una variable llamada "tipo_exportación", el segundo llamado de la función `behead` transforma el encabazado de producto de exportaciones en una variable llamada "producto_exportacion", el tercer llamado de la función `behead` transforma el tercer encabezado que denota la métrica de exportación en una variable que se llama "metrica_exportacion", y el cuarto llamado de la función `behead` transforma el cuarto encabezado que denota las fechas en una variable que se llama "mes" de tal forma que la tabla resultante ya no tenga encabezados y en su lugar tenga cuatro columnas adicionales que contienen la información de los encabezados para cada celda.

```
tabla = particion$cells[[1]] %>%
  filter(!is_blank) %>%
  behead("up-left", "tipo_exportacion") %>%
  behead("up-left", "producto_exportacion") %>%
  behead("up", "metrica_exportacion") %>%
  behead("left", "mes") %>%
  select(row, col, data_type, numeric, character, date, tipo_exportacion, producto_exportacion, metrica_exportacion, mes)

# Tabla resultante de transformar los encabezados a variables dentro del Data Frame
tabla
```

```
# A tibble: 6,388 x 10
     row   col data_type numeric character date   tipo_exportacion
   <int> <int> <chr>       <dbl> <chr>     <dttm> <chr>
 1    14     3 numeric   108864. <NA>      NA     Exportaciones tradicionales
 2    14     4 numeric    67119. <NA>      NA     Exportaciones tradicionales
 3    14     5 numeric        0  <NA>      NA     Exportaciones tradicionales
 4    15     3 numeric   114799. <NA>      NA     Exportaciones tradicionales
 5    15     4 numeric    73666. <NA>      NA     Exportaciones tradicionales
 6    15     5 numeric        0  <NA>      NA     Exportaciones tradicionales
 7    16     3 numeric    89464. <NA>      NA     Exportaciones tradicionales
 8    16     4 numeric    61685. <NA>      NA     Exportaciones tradicionales
 9    16     5 numeric        0  <NA>      NA     Exportaciones tradicionales
10    17     3 numeric   113535. <NA>      NA     Exportaciones tradicionales
# i 6,378 more rows
# i 3 more variables: producto_exportacion <chr>, metrica_exportacion <chr>,
#   mes <chr>
```

Posteriormente, se identifican el primer año y el primer mes, desde donde inicia la serie de tiempo de la variable de interés. Como ya se vio la variable mes del Data Frame contiene la fecha, por lo que el primer mes y año de la serie se debe extraer de `tabla$mes[1]`. Se observa que la primera fecha tiene el formato `"1992-01-01"`.

Para extraer el año mediante `str_extract(tabla$mes[1], regex("\\d{4}"))`, se emplea la función `str_extract` que selecciona el primer match de la expresión regular donde `regex("\\d{4}")` denota las cadenas de carácteres con 4 digitos.

Para extraer el mes mediante `as.numeric(str_extract_all(tabla$mes[1], regex("\\d{2}"))[[1]][3])`, se emplea la función `str_extract_all` que selecciona todos los match de la expresión regular donde `regex("\\d{2}")` denota las cadenas de carácteres con 2 digitos, el resultado siendo una lista. Se selecciona el primer elemento de la lista con `[[1]]` que es un vector de carácteres y se selecciona el tercer elemento del vector resultante que denota el mes de la fecha. Finalmente se transforma el resultado de lo anterio que es `"01"` a un valor numérico usando las función `as.numeric`.

```
# Primera fecha de la serie
tabla$mes[1] # el [1] se usa para extraer
```

```
[1] "1992-01-01"
```

```
# Selección del año y mes de inicio

## Año inicial
year_init = str_extract(tabla$mes[1], regex("\\d{4}")); year_init
```

```
[1] "1992"
```

```
## Mes inicial
mes_init = as.numeric(str_extract_all(tabla$mes[1], regex("\\d{2}"))[[1]][3]); mes_init
```

```
[1] 1
```

Se observa de la Figura 18 que además se necesita la información de los tres primeros encabezados para poder identificar la variable total de exportaciones tradicionales en miles de dólares FOB de la única tabla que hace parte de la hoja de Excel. De ahí, que sea neceseario específicar tres filtros, uno para cada encabezado diferente al encabezado de fecha. Se utiliza la información en las variables `filtro1` , `filtro2` y`filtro3` para identificar específicamente la serie que contiene la información sobre el total de exportaciones tradicionales en miles de dólares FOB. Para ello, se filtra la variable `tipo_exportacion` por el valor almacenado en la variable `filtro1`, se filtra la variable `producto_exportacion` por el valor almacenado en la variable `filtro2`, y se filtra la variable `metrica_exportacion` por el valor almacenado en la variable `filtro3`. Finalmente, se seleccionan solo los valores numéricos.

```
# Selección de la columna con la información de interés
tabla_selection = tabla %>%
  filter(tipo_exportacion == filtro1) %>%
  filter(producto_exportacion == paste0(filtro2, " ")) %>%
  filter(metrica_exportacion == filtro3) %>%
  filter(!str_detect(mes, "Totales")) %>%
  filter(!is.na(numeric))

# Vector numérico con la información de la variable de interés
tabla_selection$numeric
```

```
  [1]  281755.0  290900.8  247223.1  293149.6  302260.6  273670.8  308943.9
  [8]  264610.1  276209.6  298042.3  235066.6  280544.1  282833.6  295838.7
 [15]  281196.9  231688.5  269250.8  266459.8  209681.4  210027.6  239371.3
 [22]  314440.5  272872.8  255778.2  234154.9  258080.1  233153.7  268686.8
 [29]  228547.5  350788.9  362449.2  458487.3  385126.2  394243.0  411320.7
 [36]  386434.5  336641.8  342604.8  371688.3  353727.3  437419.5  411682.8
 [43]  365318.3  415251.2  397021.5  453617.9  450384.0  458541.7  338332.9
 [50]  390403.7  474565.0  488446.4  468685.4  459556.0  451550.2  399381.3
 [57]  458524.1  564488.6  537486.4  514640.4  456582.2  455458.5  417087.2
 [64]  525625.7  564923.4  491309.5  516165.5  480121.6  487125.3  474189.7
 [71]  535667.4  612503.3  485713.8  374213.3  411507.1  426377.6  543656.6
 [78]  427785.2  431815.6  377910.1  437249.1  423875.0  455257.6  481939.7
 [85]  381257.1  374049.2  416677.4  436603.0  458275.1  535322.4  524277.2
 [92]  528087.5  571678.5  527947.9  597485.1  760949.1  550672.3  618483.4
 [99]  552189.8  500910.8  604854.4  620430.8  566061.4  625779.3  530706.8
[106]  465317.4  630939.7  680778.4  526735.0  462369.4  413899.1  525664.8
[113]  474189.4  431680.2  378187.4  562777.0  498677.7  421626.9  356879.4
[120]  428462.5  390619.9  418631.1  410476.2  488429.0  432518.2  416731.7
[127]  409580.3  408864.5  466717.8  459800.3  467602.7  539478.8  492285.0
[134]  453302.7  454824.1  508646.3  598882.4  455073.6  613339.5  494896.7
[141]  429289.7  558402.9  477416.4  494465.8  651683.7  435245.9  527555.7
[148]  547172.9  607509.6  632118.9  723709.0  633562.5  704311.0  747612.2
[155]  702158.7  766763.6  679543.6  765850.2  872372.1  752262.7  809300.0
[162] 1109589.6  753936.3  947156.7  925667.6  870693.1  873337.3  963289.2
[169]  841833.6  952228.7  896744.5  925203.2 1084262.6  969219.3 1092644.3
[176] 1038212.1  927468.3 1121619.9  937093.1 1051412.4  897116.7  812670.0
[183] 1131956.4 1336264.7 1378056.8  922727.3 1370075.0 1231620.6 1093229.8
[190] 1073265.2 1446614.6 1676200.0 1564397.0 1407472.5 1655926.9 1786003.7
[197] 1853629.9 2262865.3 2124399.3 1839433.5 1789547.6 1400731.1  879130.0
[204]  792754.0 1414607.5  990819.9 1308621.3 1205341.7 1416291.7 1540387.9
[211] 1581010.6 1606232.3 1511263.8 1792468.1 1701490.4 1883983.6 1899003.9
[218] 1799068.6 1938414.8 2241856.7 2194382.1 2005764.3 2103871.4 2213423.0
[225] 1973550.5 2293729.0 2257931.4 2446708.6 2713478.1 2749446.1 3335025.8
[232] 3443622.7 3655668.7 3274470.6 3500376.0 3460925.2 3199816.0 3392655.7
[239] 3694771.6 3832260.6 3556084.2 3593660.7 4163022.4 3702603.2 3760061.8
[246] 3103269.8 3401243.3 2970482.0 3390811.3 3774540.1 3217102.4 3522407.8
[253] 3476151.1 3194223.7 3216026.6 3285626.2 3713365.9 3439671.5 3282606.0
[260] 3745959.2 3497414.1 3402367.8 3595337.0 3889032.6 3594043.8 3004982.0
[267] 3071353.1 3041627.4 4041162.0 3426084.1 3583295.3 3492624.2 3683361.7
[274] 2992927.4 2536922.9 2443696.9 1889088.1 2040987.7 2291703.3 1960365.3
[281] 2163639.4 1968289.8 1818778.3 1759764.7 1694904.1 1664962.2 1413897.5
```

```
[288] 1416059.5 1069648.2 1193862.5 1207826.3 1351850.6 1557839.5 1690467.7
[295] 1443263.2 1743306.4 1592704.7 1607827.8 1581698.4 2139622.1 1848945.0
[302] 1628300.7 1977127.7 1676552.3 2130696.9 1582296.6 1884348.8 1905158.6
[309] 2154011.7 2053344.3 1905561.1 2825992.2 2247326.2 1802484.1 2148086.9
[316] 2459785.1 2331366.8 2063961.1 2396168.4 2394826.1 2371499.5 2508636.3
[323] 2170290.6 2222136.4 1931440.5 1926419.7 2157773.5 2542609.8 2344017.8
[330] 1986661.0 1979766.5 1986212.3 1805222.9 2003654.0 1728243.7 2064958.8
[337] 2324900.1 1693679.8 1163473.1  889109.7 1094062.7 1142899.8 1228707.7
[344] 1337067.6 1177970.6 1175964.2 1136762.0 1438071.2 1463061.2 1540569.9
[351] 1695966.1 1469155.5 1691788.1 1705153.0 1758420.0 1868626.6 2219951.2
[358] 2208509.0 2438296.5 2727457.5 2424032.3 2421714.4 2911893.4 3698605.3
[365] 2740697.1 3707629.6 4109211.8 2611829.1 2995458.7 2538312.2 2869654.5
[372] 2854579.6 2314200.1 2580028.2 2503549.6 2084281.8 2428874.2 2227476.6
[379] 2377801.4 2200534.9 2416428.7 2372613.4 2320735.2 2602381.6 1969054.0
[386] 2029469.5 2095052.0
```

La úlltima parte del procesamiento, que es la transformación del vector numérico que contiene la información sobre el total de exportaciones tradicionales en miles de dólares FOB a un objeto de serie de tiempo `xts`, es exactamente igual que como se hizo con la serie del ISE de información y comunicaciones del ejemplo anterior.

```
# Creación del objeto ts
ts_obj = ts(tabla_selection$numeric, start = c(year_init, mes_init), frequency = 12)

# Creación del objeto xts
xts_obj = as.xts(ts_obj)
colnames(xts_obj) = c(nombre_variable)

# Objeto xts creado
xts_obj
```

```
         expo_tradicionales
Jan 1992           281755.0
Feb 1992           290900.8
Mar 1992           247223.1
Apr 1992           293149.6
May 1992           302260.6
Jun 1992           273670.8
Jul 1992           308943.9
Aug 1992           264610.1
Sep 1992           276209.6
Oct 1992           298042.3
     ...
Jun 2023          2227476.6
Jul 2023          2377801.4
Aug 2023          2200534.9
Sep 2023          2416428.7
Oct 2023          2372613.4
Nov 2023          2320735.2
Dec 2023          2602381.6
Jan 2024          1969054.0
Feb 2024          2029469.5
Mar 2024          2095052.0
```

4.4 Funciones para el procesamiento automático de los datos por tipo de fuente de información

La construcción de la base de datos final utilizada por los modelos de frecuencia mixta para el el nowcast de PIB requiere de los datos que fueron descargados vía web scraping como muestra la Sección 3.6 . No obstante, es necesario definir unas funciones auxiliares que complementarán las funciones de procesamiento de datos y que cumplen tareas específicas necesarias para que se pueda llevar a cabo dicho procesamiento como lo es la transformación de todos los archivos en formato `.xls` a `.xlsx` o el manejo de las expresiones regulares necesarias para la identificación de las fechas dentro de las funciones de procesamiento.

Las funciones en la presente sección hacen uso de la información que se encuentre en las hojas de procesamiento, es decir las hojas distintas a la hoja de `descarga`, del archivo `input_web_scraping_MF.xlsx` como se indicó en la Sección 4.2 .

4.4.1 Funciones auxiliares

4.4.1.1 Función para la transformación de archivos `.xls` en `.xlsx`

Se presenta el código de la función `verificar_si_es_xls` cuya funcionalidad dentro del programa consiste en transformar todo archivo descargado vía `web scraping` que se encuentre en formato `.xls` a formato `.xlsx`. Lo anterior se hace necesario, en la medida que la función `xlsx_cells` del paquete `tidyxl` necesariamente requiere que las bases de datos que va a importar a `R` estén estrictamente en formato `.xlsx`.

La función requiere los siguientes parámetros:

- `nombre_base`: Nombre de la base de datos que se va a procesar y que proviene de la columna `nombre_base` de la hoja de procesamiento de la base `input_web_scraping_MF.xlsx`. Los nombre de las bases pueden estar vacío (caso de la DIAN), terminar en formato `.xls` o en formato .`xlsx`.
- `path_microsoft_office`: Es el directorio donde se encuentra instalado `Microsoft Office` dentro del computador. En Windows, generalmente Microsoft Office se encuentra instalado en la ruta `C:/Program Files/Microsoft Office/root/Office16`.
- `path_bases_descargadas`: Directorio donde se encuentran las bases de datos que han sido descargadas vía web scraping (Sección 3.6) .
- `tmp_sleep`: Tiempo de espera que se suspende el programa mientras se hace efectiva la conversión de la base en formato `.xls` a formato `.xlsx`.

Incialmente, la funcion tiene un condicional que indica como proceder dependiendo de si el nombre de la base que se va a procesar que se encuentra en la columna `nombre_base` de la hoja de procesamiento de la base `input_web_scraping_MF.xlsx` se encuentra vacío (como sería en el caso de las bases de la DIAN que descargan un archivo `.zip`), termina en formato `.xls` o termina en formato `.xlsx`. Si el nombre está vacío la función no hace nada y si el nombre termina en formato `.xlsx` la función retorna la misma misma base en formato `.xlsx` sin ninguna transformación. Ahora bien, si el nombre terminar en formato `.xls` la función procede a hacer la conversión de la base en formato `.xls` a formato `.xlsx` . La identificación del nombre de la base se hace utilizando la función `str_ends` del paquete `stringr`.

Para hacer dicha conversión, lo primero que se hace es extraer el nombre de la base sin formato con la función `str_remove` de stringr. Luego se reemplazan en las direcciones de carpeta de los parámetros `path_microsoft_office` y `path_bases_descargadas` todo `/` por `\\\\` para que sean reconocidos por el programa que hace la conversión de archivos utilizando la función `str_replace` del paquete `stringr`.

```
# Nombre de la base de datos sin formato (i.e. sin ".xls" o ".xlsx")
nombre_base_limpio = str_remove(nombre_base, "\\.(xls|xlsx)$")

# Path donde se encuentra el programa de Microsoft Office necesario para transformar archivos ".xls" en archivos ".xlsx"
path_microsoft_office_backslash = str_replace_all(path_microsoft_office, "/", "\\\\")

# Path donde se encuentran las bases descargadas
path_bases_descargadas_backslash = str_replace_all(path_bases_descargadas, "/", "\\\\")
```

El comando de consola de `Microsoft Office` que permite hacer la conversión de un archivo `.xls` a `.xlsx` es `excelcnv.exe` y los flags `-oice` indican como debe comportarse dicho programa de consola.

Posteriormente, se definen las siguientes rutas usando la función glue:

- `xls2xlsx` : Ruta donde se encuentra el programa `excelcnv.exe` que permite transformar el archivo en formato `.xls` a formato `.xlsx`.
- `ruta_xls` : Ruta donde se encuentra el archivo `.xls` en las condiciones que requiere la consola para poderse usar junto al programa `excelcnv.exe`.
- `ruta_xlsx` : Ruta donde se guardará el archivo `.xlsx` en las condiciones que requiere la consola para poderse usar junto al programa `excelcnv.exe`.

```
# Comando de consola de Microsoft Office que permite trasnformar un archivo ".xls" en ".xlsx"
xls2xlsx = glue("\"{path_microsoft_office_backslash}\\excelcnv.exe\" -oice")

# Ruta del archivo ".xls" y del archivo ".xlsx" al que se va a transformar
ruta_xls = glue("\"{path_bases_descargadas_backslash}\\{nombre_base_limpio}.xls\"")
ruta_xlsx = glue("\"{path_bases_descargadas_backslash}\\{nombre_base_limpio}.xlsx\"")
```

Finalmente, usando la información en `args = paste(xls2xlsx, ruta_xls, ruta_xlsx)` se usa la función `system` dentro de R que permite ejecutar un comando de consola Windows (`cmd.exe`) arbitrario dentro de `R` y así hacer la conversión de los archivos de formato `.xls` a `.xlsx` .

```
args = paste(xls2xlsx, ruta_xls, ruta_xlsx)
system("cmd.exe" , input = args)
```

La función completa es:

```
# Función de transformación de la bases en formato ".xls" a formato ".xlsx"
verificar_si_es_xls = function(nombre_base, path_microsoft_office, path_bases_descargadas, tmp_sleep){

  if (is.na(nombre_base)){

    # Pass

  }else if (str_ends(nombre_base, "\\.xls$")){

    # Nombre de la base de datos sin formato (i.e. sin ".xls" o ".xlsx")
    nombre_base_limpio = str_remove(nombre_base, "\\.(xls|xlsx)$")

    # Path donde se encuentra el programa de Microsoft Office necesario para transformar archivos ".xls" en archivos ".xlsx"
    path_microsoft_office_backslash = str_replace_all(path_microsoft_office, "/", "\\\\")

    # Path donde se encuentran las bases descargadas
    path_bases_descargadas_backslash = str_replace_all(path_bases_descargadas, "/", "\\\\")

    # Comando de consola de Microsoft Office que permite trasnformar un archivo ".xls" en ".xlsx"
    xls2xlsx = glue("\"{path_microsoft_office_backslash}\\excelcnv.exe\" -oice")

    # Ruta del archivo ".xls" y del archivo ".xlsx" al que se va a transformar
    ruta_xls = glue("\"{path_bases_descargadas_backslash}\\{nombre_base_limpio}.xls\"")
    ruta_xlsx = glue("\"{path_bases_descargadas_backslash}\\{nombre_base_limpio}.xlsx\"")

    args = paste(xls2xlsx, ruta_xls, ruta_xlsx)
    system("cmd.exe" , input = args)

    # Se pausa el programa "tmp_sleep" segundos mientras se crea el archivo ".xlsx"
    Sys.sleep(tmp_sleep)

    # Nombre de la base transformada a formato ".xlsx"
    return(paste0(nombre_base_limpio, ".xlsx"))

  }else if(str_ends(nombre_base, "\\.xlsx$")){

    return(nombre_base)

  }

}
```

4.4.1.2 Funciones para extraer la fecha de cada base

Las expresiones regulares, también se emplean en las funciones auxiliares que extraen la fecha de las bases de datos que son procesadas. En particular, se usa la función str_detect porque lo que se busca es determinar si una cadena de carácteres representativa de una fecha se encuentra dentro del archivo o no. La cadena de carácteres representativa de una fecha que se quiere determinar si está o no dentro del archivo de Excel difiere dependiendiendo del mes o del trimestre específico.

> ⚠ ¿Cuál es la fecha específica que se busca?
>
> Solo se busca la primera fecha desde donde inicia la serie de tiempo de la variable de interés. Es decir, las funciones auxiliares de extración de fechas solo se usan para determinar cuál es la fecha de inicio de la serie correspondiente a la variable. Esto se debe a que, con la longitud de la serie se puede determinar fácilmente cuándo esta termina si se conoce la fecha inicial.
> No se requiere en todas las bases de datos usar las funciones de busqueda de fecha, dado que estas solo se deben emplear cuándo la fecha no se encuentra en un formato numérico. Por ejemplo, la función `deteccion_primer_mes` cuándo el mes se encuentra en formato `marzo` o `mar` en lugar de 3, mientras que la función `deteccion_primer_trimestre` se emplearía por ejemplo cuándo el segundo trimestre se especifique como `II` en lugar de 2.

4.4.1.2.1 Función para la extracción del primer mes de cada base de periodicidad mensual

La función `deteccion_primer_mes` tiene como único parámetro `mes_text` que correspondería a la cadena de texto que debería tener el primer desde donde inicia la serie de tiempo. Lo que busca la función, es determinar si la cadena de texto en `mes_text` corresponde a un mes en

formato de texto y no de número. Por ejemplo, la expresión `regex("enero|ene", ignore_case = TRUE)` indica toda cadena de texto que pueda ser `enero` o `ene`, ignorando mayúsculas o minúsculas. Por tanto, `str_detect(mes_text, regex("enero|ene", ignore_case = TRUE))` mira si la cadena de texto dentro de `mes_text` corresponde ya sea a `enero` o a `ene`. Lo anterior, se hace para cada uno de los meses del año.

```
# Función para la extracción del primer mes de cada base
deteccion_primer_mes = function(mes_text){

  if(str_detect(mes_text, regex("enero|ene", ignore_case = TRUE))){

    mes = 1

  }else if(str_detect(mes_text, regex("febrero|feb", ignore_case = TRUE))){

    mes = 2

  }else if(str_detect(mes_text, regex("marzo|mar", ignore_case = TRUE))){

    mes = 3

  }else if(str_detect(mes_text, regex("abril|abr", ignore_case = TRUE))){

    mes = 4

  }else if(str_detect(mes_text, regex("mayo|may", ignore_case = TRUE))){

    mes = 5

  }else if(str_detect(mes_text, regex("junio|jun", ignore_case = TRUE))){

    mes = 6

  }else if(str_detect(mes_text, regex("julio|jul", ignore_case = TRUE))){

    mes = 7

  }else if(str_detect(mes_text, regex("agosto|ago", ignore_case = TRUE))){

    mes = 8

  }else if(str_detect(mes_text, regex("septiembre|sep", ignore_case = TRUE))){

    mes = 9

  }else if(str_detect(mes_text, regex("octubre|oct", ignore_case = TRUE))){

    mes = 10

  }else if(str_detect(mes_text, regex("noviembre|nov", ignore_case = TRUE))){

    mes = 11

  }else if(str_detect(mes_text, regex("diciembre|dic", ignore_case = TRUE))){

    mes = 12

  }

  return(mes)

}
```

4.4.1.2.2 Función para la extracción del primer trimestre de cada base de periodicidad trimestral

La función deteccion_primer_trimestre tiene la misma funcionalidad que la función deteccion_primer_mes pero en lugar de hacer la detección de meses, hace la detección de trimestres, buscando si el primer trimestre tiene alguno de los carácteres romanos `I`, `II`, `III` o `IV`.

```r
# Función para la extracción del primer trimestre de cada base
deteccion_primer_trimestre = function(trimestre_text){

  if(str_detect(trimestre_text, regex("I", ignore_case = TRUE))){

    trimestre = 1

  }else if(str_detect(trimestre_text, regex("II", ignore_case = TRUE))){

    trimestre = 2

  }else if(str_detect(trimestre_text, regex("III", ignore_case = TRUE))){

    trimestre = 3

  }else if(str_detect(trimestre_text, regex("IV", ignore_case = TRUE))){

    trimestre = 4

  }

  return(trimestre)

}
```

4.4.2 Funciones para el procesamiento de las bases descargadas vía `web scraping` por fuente de información

Definidas las funciones auxiliares en la sección anterior, se procede a definir las funciones que se usan para procesar cada una de las bases de datos descargadas por fuente de información.

Para cada tipo de **`Fuente`** de una entidad gubernamental, una agencia de estadística, un gremio o una entidad del sector privado se le asocia una función específica para procesar los datos que han sido descargados de su página web vía **`web scraping`**. Todas éstas funciones tienen en común los siguientes parámetros:

- **`nombre_variable`**: Nombre de la variable tal cuál va a aparecer en la base de datos final usada por los modelos de frecuencia mixta para el nowcast de PIB sectorial.
- **`nombre_base`**: Contiene el nombre de la base de datos que fue descargada vía web scraping y que contiene la variable dentro de ella.
- **`nombre_sheet`**: Nombre específico de la hoja dentro del documento **`Excel`** que fue descargado vía **`web scraping`** que contiene la variable de interés.
- **`palabra_identificacion_tabla`**: Palabra que identifica la tabla de datos específica dentro de la hoja del archivo **`Excel`** descargado que contiene la variable.
- Parámetros de **`filtramiento`**: Se usan para identificar la variable de interés dentro de la tabla de datos que la contiene. Cada una de las variables de filtro corresponde a un valor de un encabezado. Por ejemplo, si la tabla tiene tres encabezados y se necesita la información de los tres encabezados para identificar de manera única la variable de interés, entonces habrán tres variables de filtro para dicha identificación.

  Por ejemplo, la base de datos del ISE del DANE si bien tiene tres encabezados: 1) un "encabezado" a la izquierda para denotar el Concepto o tipo de actividad, 2) un "encabezado" en la parte superior que denota el año y 3) un encabezado debajo del encabezado superior que denota el mes, solo necesita especificar en la variable de **`filtro1`** el valor relacionado al primer encabezado, es decir definir el Concepto o tipo de actividad, para identificar la serie completa. Pero, en otro ejemplo, la tabla dentro de la hoja **`Tra y Notra`** de la base de datos de exportaciones del DANE que identifica la variable que contiene la información sobre la cantidad de toneladas métricas exportadas de carbón necesita la información de tres encabezados para poderse identificar: **`filtro1`**: "Exportaciones tradicionales", **`filtro2`**: "Carbón" y **`filtro3`**: "Toneladas métricas". En resumen,

  - **`filtro1`**: Corresponde al valor del primer encabezado, para identificar la variable.
  - **`filtro2`**: Corresponde al valor del segundo encabezado, para identificar la variable.
  - **`filtro3`**: Corresponde al valor del tercer encabezado, para identificar la variable.

  Nota: A fecha del 22 de mayo de 2024, las funciones de procesamiento de datos admiten máximo tres variables de filtro. No obstante, en caso de requerir aumentar el número de filtros esto se puede hacer fácilmente agregando una variable de filtro adicional en el código que define la función. En la actualidad, no se ha encontrado tablas con más de tres encabezados, por lo que no ha sido necesario tener más de tres variables de filtro.

  Nota: Hay algunas funciones que no tienen las tres variables de filtro especificadas arriba. Lo anterior se debe a que dichas fuentes de información no requieren las tres variables de filtro para procesar sus bases de datos.

- `Fuente:` Contiene el nombre de la entidad gubernamental, la agencia de estadística, el gremio o la entidad del sector privado de donde proviene la información. Dicha variable es importante, en la medida que idenfica cuál función de procesamiento específica se debe usar para procesar los datos que han sido descargados vía `web scraping`, dado que cada entidad diferente tiene su propia función de procesamiento específica. A fecha del 22 de mayo de 2024, existen funciones de procesamiento para las siguiente entidades:
  - DANE
  - Camacol
  - Aerocivil
  - FNC
  - DIAN
  - Banrep

4.4.2.1 Función para el procesamiento de las bases de datos mensuales del DANE

La función `procesamiento_DANE` corresponde a la función que se usa para procesar las bases de datos que contienen información mensual que han sido descargadas del `DANE` vía web scraping.

La función tiene una lógica de ir de lo general a lo particular en el procesamiento de los datos. Inicialmente, hay que identificar el archivo `Excel` específico del DANE que se quiere procesar (si bien lo ideal sería que la función fuera capaz de procesar cada archivo de Excel del DANE sin importar la información que contenga dentro de ella, en la práctica no fue posible realizar esa generalización dado que el formato de cada base de datos del DANE cambia sustancialmente dependiendo del tipo de información y variable que se tenga dentro de él), luego la hoja específica de dicha archivo de Excel que se está procesando y finalmente la variable en específico que se quiere tratar.

A la fecha del 22 de mayo de 2024, la función permite procesar los siguientes archivos de Excel del `DANE`:

1. **Índice de Seguimiento a la Economía (ISE)**: `DANE_ISE_9_actividades.xlsx` ó `DANE_ISE_12_actividades.xlsx`
2. **Encuesta Mensual Manufacturera con Enfoque Territorial (EMMET)**: `DANE_EMMET_territorial.xlsx`
3. **Encuesta Mensual de Comercio (EMC)**: `DANE_EMC.xlsx`
4. **Comercio Internacional - Importaciones**: `DANE_Importaciones.xlsx`
5. **Comercio Internacional - Exportaciones**: `DANE_Exportaciones.xlsx`
6. **Estadísticas de cemento gris (ECG)**: `DANE_ECG.xlsx`
7. **Estadísticas de Licencias de Construcción (ELIC)**: `DANE_ELIC.xlsx`
8. **Índice de Precios del Productor (IPP):** `DANE_IPP.xlsx`
9. **Índice de Precios al Consumidor (IPC):** `DANE_IPC.xlsx`
10. **Encuesta mensual de alojamiento (EMA):** `DANE_EMA.xlsx`
11. **Gran encuesta integrada de hogares (GEIH)**: `DANE_Mercado_laboral.xlsx`
12. **Encuesta de sacrificio de ganado (ESAG)**: `DANE_Sacrificio_ganado.xlsx`
13. **Índice de producción industrial (IPI):** `DANE_IPI.xlsx`
14. **Estadísticas de Concreto Premezclado (EC)**: `DANE_EC.xlsx`

Para cada archivo de Excel, se pueden manejar diferentes variables. Por ejemplo, para los archivos de `DANE_ISE_9_actividades.xlsx` ó `DANE_ISE_12_actividades.xlsx` se puede recuperar todos la información de cualquier `ISE` sectorial modificando el parámetro de `filtro1` que idenfica al ISE sectorial en específico de interés. La forma de tratar cada variable varía dependiendo de la estructura de la base de datos que lo contiene como se expusó antes en la Sección 4.3.2.2 cuándo se hizo la explicación del paquete `unpivotr` y se ilustrarón los dos ejemplos.

Ahora bien, el código de la función `procesamiento_DANE` se compone de tres partes importantes. La primera componente es común para culquier archivo de Excel que se importe y se procese del DANE. Lo primero que hay que hacer es importar el archivo `.xlsx` mediante la función `xlsx_cells`, se ubica la palabra que se encuentra en la celda de la esquina superior izquierda y se almacena en la variable `esquina_sup_izq_tabla` y finalmente se extraen las celdas del Excel que hacen parte de la tabla `particion`.

```
# Importación de la base de datos mediante la función "xlsx_cells"
base = tidyxl::xlsx_cells(nombre_base, sheets = nombre_sheet)


# Identificación de la esquina superior izquierda de la tabla
esquina_sup_izq_tabla = dplyr::filter(base, character == palabra_identificacion_tabla)

# Particion de la base que contiene la informacion de la tabla de interés dentro de la hoja de excel
particion = partition(base, esquina_sup_izq_tabla)
```

La segunda componente de la función, ya es específica y programada para que se ajuste a cada archivo Excel particular del DANE que la función sea capaz de tratar. Incializa con la línea de código que dice `# ==== Inicio: Específico para cada base ====` y finaliza con la línea de código que dice `# ==== Fin: Específico para cada base ====` . Dentro de esas dos líneas de código se encuentra un condicional que indica que porción del código se debe correr dependiendo del archivo de `Excel` específico que se va a procesar. Dentro de cada porción del código específico para cada archivo de Excel puede haber otro condicional para indicar cuál de hoja de Excel del archivo hay que usar. Luego, puede haber uno o más condicionales que se usan para identificar la variable en específico que se quiere tratar. En este punto, es donde

se utiliza la información de las variables de filtro porque son esas variables las que permiten identificar la variable dentro de la tabla que las contenga en la hoja de Excel específica.

Por ejemplo, el siguiente fragmento de código permite identificar las variables provenientes del Excel `DANE_Exportaciones.xlsx`. Lo primero que hay que hacer es transformar la `particion` de tal forma que los encabezados de la tabla se transformen en variables dentro del Data Frame que representa la tabla. Luego, se especifican el año y el mes de la fecha inicial y luego mediante dos condicionales que emplean la información de las variables de `filtro` se extraen las variables de interés específicas. En este ejemplo, se necesitan tres variables de `filtro` para identificar cada una de las variables que se quiere extraer.

```
# 2.1.5 DANE_Exportaciones.xlsx ----

# Código para la base de datos que contiene las variables de "exportación"

# Selección de las celdas de Excel que contienen la info de la tabla
tabla = particion$cells[[1]] %>%
  filter(!is_blank) %>%
  behead("up-left", "tipo_exportacion") %>%
  behead("up-left", "producto_exportacion") %>%
  behead("up", "metrica_exportacion") %>%
  behead("left", "mes") %>%
  select(row, col, data_type, numeric, character, date, tipo_exportacion, producto_exportacion, metrica_exportacion, mes)

# Selección del año y mes de inicio

## Año inicial
year_init = str_extract(tabla$mes[1], regex("\\d{4}"))

## Mes inicial
mes_init = as.numeric(str_extract_all(tabla$mes[1], regex("\\d{2}"))[[1]][3]) # Toca hacer todo esto para lograr extraer el -

# Se filtra la información dependiendo de la variable de exportación de interés
if (filtro1 == "Exportaciones tradicionales"){

  if (filtro2 == "Total Exportaciones Tradicionales"){

    # Selección de la columna con la información de interés
    tabla_selection = tabla %>%
      filter(tipo_exportacion == filtro1) %>%
      filter(producto_exportacion == paste0(filtro2, " ")) %>%
      filter(metrica_exportacion == filtro3) %>%
      filter(!str_detect(mes, "Totales")) %>%
      filter(!is.na(numeric))

  }else{

    # Selección de la columna con la información de interés
    tabla_selection = tabla %>%
      filter(tipo_exportacion == filtro1) %>%
      filter(producto_exportacion == filtro2) %>%
      filter(metrica_exportacion == filtro3) %>%
      filter(!str_detect(mes, "Totales")) %>%
      filter(!is.na(numeric))

  }

}else{

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    filter(tipo_exportacion == filtro1) %>%
    filter(metrica_exportacion == filtro2) %>%
    filter(!str_detect(mes, "Totales")) %>%
    filter(!is.na(numeric))

}
```

Finalmente, la última componente de la función nuevamente es común para cualquier archivo de `Excel` que se importe y se procese del

DANE. Lo primero que hay que destacar es que las variables proveniente de los Excel `DANE_Importaciones.xlsx` y `DANE_ELIC.xlsx` tienen un tratamiento especial, dado que para las variables del primer Excel hay que reescalar sus unidades, mientras que para las variables del segundo Excel hay que realizar un empalme hacía atrás. Si las variables provienen de archivos de Excel distintos a los mencionados arriba, entonces se crea un objeto ts `ts_obj = ts(tabla_selection$numeric, start = c(year_init, mes_init), frequency = 12)` con el vector de celdas numéricas que se extrae de `tabla_selection$numeric` y luego dicho objeto ts se transforma a un objeto `xts`. Finalmente, se le da el nombre al objeto `xts` del parámetro `nombre_variable` y se retorna el objeto `xts_obj`.

```
if (!(nombre_base %in% c("DANE_Importaciones.xlsx", "DANE_ELIC.xlsx"))){

  # Creación del objeto ts
  ts_obj = ts(tabla_selection$numeric, start = c(year_init, mes_init), frequency = 12)

  # Creación del objeto xts
  xts_obj = as.xts(ts_obj)
  colnames(xts_obj) = c(nombre_variable)

  # Retorna el objeto xts
  return(xts_obj)

}
```

El código completo de la función `procesamiento_DANE` se presenta a continuación:

```
# Función para procesar las bases mensuales del DANE
procesamiento_DANE = function(nombre_base, nombre_sheet, palabra_identificacion_tabla, nombre_variable, filtro1, filtro2, fil

  # Importación de la base de datos mediante la función "xlsx_cells"
  base = tidyxl::xlsx_cells(nombre_base, sheets = nombre_sheet)


  # Identificación de la esquina superior izquierda de la tabla
  esquina_sup_izq_tabla = dplyr::filter(base, character == palabra_identificacion_tabla)

  # Particion de la base que contiene la informacion de la tabla de interés dentro de la hoja de excel
  particion = partition(base, esquina_sup_izq_tabla)

  # ==== Inicio: Específico para cada base ====

  # La selección de la información cambia dependiendo de la base de datos
  if ((nombre_base == "DANE_ISE_9_actividades.xlsx") || (nombre_base == "DANE_ISE_12_actividades.xlsx")){

    # 2.1.1 DANE_ISE_9_actividades.xlsx ó DANE_ISE_12_actividades.xlsx ----

    # Código para la base de datos que contiene las variables del "ISE"

    # Selección de las celdas de Excel que contienen la info de la tabla
    tabla = particion$cells[[1]] %>%
      filter(!is_blank) %>%
      behead("up-left", "año") %>%
      behead("up", "mes") %>%
      behead("left", "actividad") %>%
      select(row, col, data_type, numeric, character, date, año, mes, actividad)

    # Selección de la columna con la información de interés
    tabla_selection = tabla %>%
      filter(actividad == filtro1) %>%
      filter(!is.na(numeric))

    # Selección del año y mes de inicio

    ## Año inicial
    year_init = tabla_selection$año[1]

    ## Mes inicial
    mes_init = deteccion_primer_mes(tabla_selection$mes[1])
```

```
}else if(nombre_base == "DANE_EMMET_territorial.xlsx"){

  # 2.1.2 DANE_EMMET_territorial.xlsx ----

  # Código para las variables de la base "EMMET" (Industria)

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up", "encabezados") %>%
    select(row, col, data_type, numeric, character, date, encabezados)

  # El condicional se hace necesario por el espacio que hay a la hora de buscar "Producción \r\nreal"
  if (filtro1 == "Producción \\r\\nreal"){

    # Selección de la columna con la información de interés
    tabla_selection = tabla %>%
      filter(encabezados == "Producción \r\nreal ") %>%
      filter(!is.na(numeric))

  }else{

    # Selección de la columna con la información de interés
    tabla_selection = tabla %>%
      filter(encabezados == filtro1) %>%
      filter(!is.na(numeric))

  }

  # Selección del año y mes de inicio

  ## Año inicial
  tabla_year_init = tabla %>% filter(encabezados == "Año")
  year_init = tabla_year_init$numeric[1]

  ## Mes inicial
  tabla_mes_init = tabla %>% filter(encabezados == "Mes")
  mes_init = tabla_mes_init$numeric[1]

}else if(nombre_base == "DANE_EMC.xlsx"){

  # 2.1.3 DANE_EMC.xlsx ----

  # Código para las variables de la base "EMC" (Comercio)

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up", "encabezados") %>%
    select(row, col, data_type, numeric, character, date, encabezados)

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    filter(encabezados == filtro1) %>%
    filter(!is.na(numeric))

  # Selección del año y mes de inicio

  ## Año inicial
  tabla_year_init = tabla %>% filter(encabezados == "Año")
  year_init = tabla_year_init$numeric[1]

  ## Mes inicial
  tabla_mes_init = tabla %>% filter(encabezados == "Mes")
  mes_init = deteccion_primer_mes(tabla_mes_init$character[1])
```

```
}else if(nombre_base == "DANE_Importaciones.xlsx"){

  # 2.1.4 DANE_Importaciones.xlsx ----

  # Se filtra la base para que solo se seleccione la celda que tiene la información "Mes (Año-Año)p"
  base_init_year_mes = base %>%
    filter(str_detect(base$character, regex(".*\\(\\d{4} - \\d{4}\\).*", ignore_case = TRUE)))

  # Selección del año y mes de inicio

  ## Año inicial
  year_init = str_extract(base_init_year_mes$character, regex("\\d{4}"))

  ## Mes inicial
  mes_init = deteccion_primer_mes(str_extract(base_init_year_mes$character, regex("[A-Za-z]+")))

  # Código para la base de datos que contiene las variables de "importación"

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up-left", "encabezados1") %>%
    behead("up-left", "encabezados2") %>%
    behead("left", "CUODE") %>%
    behead("left", "descripcion_importacion") %>%
    select(row, col, data_type, numeric, character, date, encabezados1, encabezados2, CUODE, descripcion_importacion)

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    filter(descripcion_importacion == filtro1) %>%
    filter(encabezados1 == paste0(year_init, "p")) %>%
    filter(!is.na(numeric))

  # Dependiendo de si la base es de Enero u otra mes del año, el archivo Excel cambia
  if (nrow(tabla_selection) == 1){

    # Creación del objeto ts
    ts_obj = ts(tabla_selection[1,]$numeric, start = c(year_init, mes_init), frequency = 12)

    # Creación del objeto xts
    xts_obj = as.xts(ts_obj)
    colnames(xts_obj) = c(nombre_variable)

    # Retorna el objeto xts
    return(xts_obj)

  }else if (nrow(tabla_selection) == 2){

    # Creación del objeto ts
    ts_obj = ts(tabla_selection[2,]$numeric, start = c(year_init, mes_init), frequency = 12)

    # Creación del objeto xts
    xts_obj = as.xts(ts_obj)
    colnames(xts_obj) = c(nombre_variable)

    # Retorna el objeto xts
    return(xts_obj)

  }

}else if(nombre_base == "DANE_Exportaciones.xlsx"){

  # 2.1.5 DANE_Exportaciones.xlsx ----

  # Código para la base de datos que contiene las variables de "exportación"
```

```
  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up-left", "tipo_exportacion") %>%
    behead("up-left", "producto_exportacion") %>%
    behead("up", "metrica_exportacion") %>%
    behead("left", "mes") %>%
    select(row, col, data_type, numeric, character, date, tipo_exportacion, producto_exportacion, metrica_exportacion, mes)

  # Selección del año y mes de inicio

  ## Año inicial
  year_init = str_extract(tabla$mes[1], regex("\\d{4}"))

  ## Mes inicial
  mes_init = as.numeric(str_extract_all(tabla$mes[1], regex("\\d{2}"))[[1]][3]) # Toca hacer todo esto para lograr extraer

  # Se filtra la información dependiendo de la variable de exportación de interés
  if (filtro1 == "Exportaciones tradicionales"){

    if (filtro2 == "Total Exportaciones Tradicionales"){

      # Selección de la columna con la información de interés
      tabla_selection = tabla %>%
        filter(tipo_exportacion == filtro1) %>%
        filter(producto_exportacion == paste0(filtro2, " ")) %>%
        filter(metrica_exportacion == filtro3) %>%
        filter(!str_detect(mes, "Totales")) %>%
        filter(!is.na(numeric))

    }else{

      # Selección de la columna con la información de interés
      tabla_selection = tabla %>%
        filter(tipo_exportacion == filtro1) %>%
        filter(producto_exportacion == filtro2) %>%
        filter(metrica_exportacion == filtro3) %>%
        filter(!str_detect(mes, "Totales")) %>%
        filter(!is.na(numeric))

    }

  }else{

    # Selección de la columna con la información de interés
    tabla_selection = tabla %>%
      filter(tipo_exportacion == filtro1) %>%
      filter(metrica_exportacion == filtro2) %>%
      filter(!str_detect(mes, "Totales")) %>%
      filter(!is.na(numeric))

  }

}else if(nombre_base == "DANE_ECG.xlsx"){

  # 2.1.6 DANE_ECG.xlsx ----

  # Código para la base de datos que contiene a la variable "despachos_cemento"

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up-left", "encabezados1") %>%
    behead("up", "encabezados2") %>%
    behead("left-up", "año") %>%
    behead("left", "mes") %>%
```

```
    select(row, col, data_type, numeric, character, date, encabezados1, encabezados2, año, mes)

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    filter(encabezados1 == paste0(filtro1, " ")) %>%
    filter(encabezados2 == filtro2) %>%
    filter(!is.na(numeric))

  # Selección del año y mes de inicio

  ## Año inicial
  year_init = tabla_selection$año[1]

  ## Mes inicial
  mes_init = deteccion_primer_mes(tabla_selection$mes[1])

}else if(nombre_base == "DANE_ELIC.xlsx"){

  # 2.1.7 DANE_ELIC.xlsx ----

  # Código para la base de datos que contiene a la variable "area_construccion"

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up-left", "encabezados1") %>%
    behead("up", "encabezados2") %>%
    behead("left-up", "año") %>%
    behead("left", "mes") %>%
    select(row, col, data_type, numeric, character, date, encabezados1, encabezados2, año, mes)

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    filter(encabezados1 == filtro1) %>%
    filter(encabezados2 == "302 \r\nmunicipios") %>%
    filter(!is.na(numeric))

  # Promedio area construida año 2015
  promedio_area_construida_2015 = 2616877

  # Estandarización de la variable
  tabla_final = tabla_selection %>%
    mutate(area_construccion =  (numeric * 100) / promedio_area_construida_2015)

  # Selección del año y mes de inicio

  ## Año inicial
  year_init = tabla_final$año[1]

  ## Mes inicial
  mes_init = deteccion_primer_mes(tabla_final$mes[1])

  # Creación del objeto ts
  ts_obj = ts(tabla_final$area_construccion, start = c(year_init, mes_init), frequency = 12)

  # Creación del objeto xts
  xts_obj = as.xts(ts_obj)
  colnames(xts_obj) = c(nombre_variable)

  # Retorna el objeto xts
  return(xts_obj)

}else if(nombre_base == "DANE_IPP.xlsx"){

  # 2.1.8 DANE_IPP.xlsx ----
```

```
  # Código para la base de datos que contiene a la variable "despachos_cemento"

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up-left", "encabezado1") %>%
    behead("up", "encabezado2") %>%
    select(row, col, data_type, numeric, character, date, encabezado1, encabezado2)

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    filter(encabezado1 == filtro1) %>%
    filter(encabezado2 == filtro2) %>%
    filter(!is.na(numeric))

  # Selección del año y mes de inicio

  ## Año inicial
  year_init = na.omit(str_extract(base$character, regex("\\d{4}")))[2]

  # Esquina superior izquierda de la tabla
  esquina_sup_izq_tabla2 = dplyr::filter(base, character == year_init)

  # Particion de la base que contiene la informacion de la tabla de interés dentro de la hoja de excel
  particion2 = partition(base, esquina_sup_izq_tabla2)

  tabla2 = particion2$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("left-up", "año") %>%
    behead("left", "mes") %>%
    select(row, col, data_type, numeric, character, date, año, mes)

  ## Mes inicial
  mes_init = deteccion_primer_mes(tabla2$mes[1])

}else if(nombre_base == "DANE_IPC.xlsx"){

  # 2.1.9 DANE_IPC.xlsx ----

  # Código para la base de datos que contiene a la variable "despachos_cemento"

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up", "año") %>%
    behead("left", "mes") %>%
    select(row, col, data_type, numeric, character, date, año, mes)

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    arrange(año) %>%
    filter(!is.na(numeric))

  # Selección del año y mes de inicio

  ## Año inicial
  year_init = tabla_selection$año[1]

  ## Mes inicial
  mes_init = deteccion_primer_mes(tabla_selection$mes[1])

}else if(nombre_base == "DANE_EMA.xlsx"){

  # 2.1.10 DANE_EMA.xlsx ----

  # Código para la base de datos que contiene a la variable "despachos_cemento"
```

```
  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up", "encabezados") %>%
    behead("left-up", "año") %>%
    behead("left", "mes") %>%
    select(row, col, data_type, numeric, character, date, encabezados, año, mes)

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    filter(encabezados == filtro1) %>%
    filter(!is.na(numeric))

  # Selección del año y mes de inicio

  ## Año inicial
  year_init = tabla_selection$año[1]

  ## Mes inicial
  mes_init = deteccion_primer_mes(tabla_selection$mes[1])

}else if(nombre_base == "DANE_Mercado_laboral.xlsx"){

  # 2.1.11 DANE_Mercado_laboral.xlsx ----

  # Código para la base de datos que contiene las variables de "mercado laboral"

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up-left", "año") %>%
    behead("up", "mes") %>%
    behead("left", "concepto_mercado_laboral") %>%
    select(row, col, data_type, numeric, character, date, año, mes, concepto_mercado_laboral)

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    filter(concepto_mercado_laboral == filtro1) %>%
    filter(!is.na(numeric))

  # Selección del año y mes de inicio

  ## Año inicial
  year_init = tabla$año[1]

  ## Mes inicial
  mes_init = deteccion_primer_mes(tabla$mes[1])

}else if(nombre_base == "DANE_Sacrificio_ganado.xlsx"){

  # 2.1.12 DANE_Sacrificio_ganado.xlsx ----

  # Código para la base de datos que contiene las variables de "Sacrificio de ganado"

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up-left", "encabezados1") %>%
    behead("up", "encabezados2") %>%
    behead("left", "periodo") %>%
    select(row, col, data_type, numeric, character, date, encabezados1, encabezados2, periodo)

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    filter(encabezados1 == filtro1) %>%
```

```
    filter(encabezados2 == filtro2) %>%
    filter(periodo != "Total general") %>%
    filter(!is.na(numeric))

  # Selección del año y mes de inicio

  ## Año inicial
  year_init = na.omit(str_extract(base$character, regex("\\d{4}")))[1]

  ## Mes inicial
  mes_init = deteccion_primer_mes(tabla_selection$periodo[1])

}else if(nombre_base == "DANE_IPI.xlsx"){

  # 2.1.13 DANE_IPI.xlsx ----

  # Código para la base de datos que contiene a la variable "produccion_carbon_volum"

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up", "encabezados") %>%
    behead("left", "dominios") %>%
    behead("left", "año") %>%
    behead("left", "mes") %>%
    behead("left", "clases_industriales") %>%
    select(row, col, data_type, numeric, character, date, encabezados, año, mes, clases_industriales)

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    filter(clases_industriales == filtro1) %>%
    filter(!is.na(numeric))

  # Selección del año y mes de inicio

  ## Año inicial
  year_init = tabla_selection$año[1]

  ## Mes inicial
  mes_init = tabla_selection$mes[1]

}else if(nombre_base == "DANE_EC.xlsx"){

  # 2.1.14 DANE_EC.xlsx ----

  # Código para la base de datos que contiene las variables de "estadísticas de concreto"

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up-left", "encabezados1") %>%
    behead("up", "encabezados2") %>%
    behead("left", "año") %>%
    behead("left", "mes") %>%
    select(row, col, data_type, numeric, character, date, encabezados1, encabezados2, año, mes)

  # Se filtra la información dependiendo de la variable de "estadística de concreto" de interés
  if (filtro1 == "Edificaciones"){

    # Selección de la columna con la información de interés

    # Selección 1
    tabla_selection1 = tabla %>%
      filter(encabezados1 == filtro1) %>%
      filter(!is.na(numeric))
```

```
      # Selección 2
      tabla_selection2 = tabla %>%
        filter(encabezados1 == filtro2) %>%
        filter(encabezados2 == filtro3) %>%
        filter(!is.na(numeric))

      # Vector numérico con la información de interés
      numeric = tabla_selection1$numeric + tabla_selection2$numeric

      # Data frame que contiene el vector numérico de interés
      tabla_selection = as.data.frame(numeric)

      # Data frame que también contiene los años y los meses
      tabla_selection = tabla_selection %>%
        mutate(año = tabla_selection1$año,
               mes = tabla_selection1$mes)


    }else if(filtro1 == "Obras Civiles (desagregación CPC versión 2.1)"){

      # Selección de la columna con la información de interés
      tabla_selection = tabla %>%
        filter(encabezados1 == filtro1) %>%
        filter(encabezados2 == filtro2) %>%
        filter(!is.na(numeric))

    }

    # Selección del año y mes de inicio

    ## Año inicial
    year_init = tabla_selection$año[1]

    ## Mes inicial
    mes_init = deteccion_primer_mes(tabla_selection$mes[1])

  }

  # ==== Fin: Específico para cada base ====

  if (!(nombre_base %in% c("DANE_Importaciones.xlsx", "DANE_ELIC.xlsx"))){

    # Creación del objeto ts
    ts_obj = ts(tabla_selection$numeric, start = c(year_init, mes_init), frequency = 12)

    # Creación del objeto xts
    xts_obj = as.xts(ts_obj)
    colnames(xts_obj) = c(nombre_variable)

    # Retorna el objeto xts
    return(xts_obj)

  }

}
```

En caso de que se quiera extender la función anterior agregando un Excel adicional del DANE para que pueda ser procesado, toca agregar una condición adicional en donde se especifique el nombre de la base del Excel adicional que se va a agregar y además se coloque dentro de esta nueva condición el código de procesamiento de las variables que hagan parte de esa nuevo Excel que se va a clasificar, todo ello dentro de las líneas de código `# ==== Inicio: Específico para cada base ====` y `# ==== Fin: Específico para cada base ====`. En resumen, extener la función para que procese un nuevo Excel consiste en agregar una nueva condición al condicional que clasifica los archivos de Excel de entrada que va a ser procesados.

⚠ Relación entre el condicional principal de la función `procesamiento_DANE` y la base de datos `input_web_scraping_MF.xlsx`

Nota: Los nombres que hacen parte de las condiciones (e.g. `nombre_base == "DANE_EMMET_territorial.xlsx"` o `nombre_base == "DANE_Mercado_laboral.xlsx`) del condicional externo que clasifica los archivos de Excel que van a ser procesados deben corresponder a los nombre que aparecen en la columna `nombre_base` de las hojas de procesamiento, es decir de las hojas que son diferentes a la hoja de `descarga`, de la base de datos `input_web_scraping_MF.xlsx`. Si los nombres de los archivos `Excel` que aparecen en las condiciones del condicional mencionado no corresponden a los nombre que aparecen en la columna `nombre_base`, el código no va a funcionar.

4.4.2.2 Función para el procesamiento de las bases de datos trimestrales del DANE

La función `procesamiento_DANE_trimestral` corresponde a la función que se usa para procesar las bases de datos que contienen información trimestral que han sido descargadas del `DANE` vía web scraping.

Al igual que su análogo mensual la función tiene una lógica de ir de lo general a lo particular en el procesamiento de los datos, manejando la misma estructura conceptual que la función equivalente para procesar datos mensuales expuesta anteriormente.

Al igual que la función de procesamiento de datos mensuales del DANE, la función de procesamiento de datos trimestrales tiene tres componentes con la misma estructura que el equivalente mensual. Una primera componente para generar la partición que contiene la tabla de información, una segunda componente que es específica para cada archivo de Excel que se impote (en este caso a mayo 22 de 2024 solo se importan las bases `DANE_PIB_demanda.xlsx` y `DANE_PIB_oferta.xlsx`) y que tiene una estructura similar a su análogo mensual que va desde la línea de código `# ==== Inicio: Específico para cada base ====` hasta la línea de código `# ==== Fin: Específico para cada base ====`, y finalmente una tercera componente que se encarga en generar el objeto de serie de tiempo `xts`.

```
# Función para procesar las bases mensuales del DANE
procesamiento_DANE_trimestral = function(nombre_base, nombre_sheet, palabra_identificacion_tabla, nombre_variable, filtro1){

  # Importación de la base de datos mediante la función "xlsx_cells"
  base = tidyxl::xlsx_cells(nombre_base, sheets = nombre_sheet)


  # Identificación de la esquina superior izquierda de la tabla
  esquina_sup_izq_tabla = dplyr::filter(base, character == palabra_identificacion_tabla)

  # Particion de la base que contiene la informacion de la tabla de interés dentro de la hoja de excel
  particion = partition(base, esquina_sup_izq_tabla)

  # ==== Inicio: Específico para cada base ====

  # La selección de la información cambia dependiendo de la base de datos
  if (nombre_base == "DANE_PIB_demanda.xlsx"){

    # 2.2.1 DANE_PIB_demanda.xlsx ----

    # Código para la base de datos que contiene las variables del"PIB por demanda"

    # Selección de las celdas de Excel que contienen la info de la tabla
    tabla = particion$cells[[1]] %>%
      filter(!is_blank) %>%
      behead("up-left", "año") %>%
      behead("up", "trimestre") %>%
      behead("left", "actividad") %>%
      select(row, col, data_type, numeric, character, date, año, trimestre, actividad)

    # Selección de la columna con la información de interés
    tabla_selection = tabla %>%
      filter(actividad == filtro1) %>%
      filter(!is.na(numeric))

    # Selección del año y mes de inicio

    ## Año inicial
    year_init = tabla_selection$año[1]

    ## Trimestre inicial
    trimestre_init = deteccion_primer_trimestre(tabla_selection$trimestre[1])
```

```
  }else if(nombre_base == "DANE_PIB_oferta.xlsx"){

    # 2.2.2 DANE_PIB_oferta.xlsx ----

    # Código para la base de datos que contiene las variables del"PIB por oferta"

    # Selección de las celdas de Excel que contienen la info de la tabla
    tabla = particion$cells[[1]] %>%
      filter(!is_blank) %>%
      behead("up-left", "año") %>%
      behead("up", "trimestre") %>%
      behead("left", "actividad") %>%
      select(row, col, data_type, numeric, character, date, año, trimestre, actividad)

    # Selección de la columna con la información de interés
    tabla_selection = tabla %>%
      filter(actividad == filtro1) %>%
      filter(!is.na(numeric))

    # Selección del año y mes de inicio

    ## Año inicial
    year_init = tabla_selection$año[1]

    ## Trimestre inicial
    trimestre_init = deteccion_primer_trimestre(tabla_selection$trimestre[1])

  }

  # ==== Fin: Específico para cada base ====

  # Selección del año y trimestre de inicio

  # Creación del objeto ts
  ts_obj = ts(tabla_selection$numeric, start = c(year_init, trimestre_init), frequency = 4)

  # Creación del objeto xts
  xts_obj = as.xts(ts_obj)
  colnames(xts_obj) = c(nombre_variable)

  # Retorna el objeto xts
  return(xts_obj)

}
```

4.4.2.3 Función para el procesamiento de las bases de datos de Camacol

La función `procesamiento_Camacol` corresponde a la función que se usa para procesar las bases de datos que contienen información mensual que han sido descargadas de `CAMACOL` vía web scraping. Su estructura es muy similar a la función de procesamiento de datos del `DANE`: 1) se selecciona la tabla que contiene las celdas de interés y se almacena en la variable `particion`, luego se extrae la información numérica de las variables de interés y finalmente se crear el objeto `xts`.

```
# Función para procesar las bases de Camacol
procesamiento_Camacol = function(nombre_base, nombre_sheet, palabra_identificacion_tabla, nombre_variable, filtro1, filtro2){

  # Importación base de datos
  base = tidyxl::xlsx_cells(nombre_base, sheets = nombre_sheet) %>%
    select(row, col, data_type, numeric, character, date)

  # Esquina superior izquierda de la tabla
  esquina_sup_izq_tabla = dplyr::filter(base, character == palabra_identificacion_tabla)

  # Particion de la base que contiene la informacion de la tabla de interés dentro de la hoja de excel
  particion = partition(base, esquina_sup_izq_tabla)
```

```
  # Tabla de datos con la información de interés
  tabla = particion$cells[[1]] %>%
    behead("NNW", "regiones") %>%
    behead("N", "tipo_de_vivienda")

  # Tabla de datos filtrada solo con la variable de interés
  tabla_selection = tabla %>%
    filter(regiones == filtro1) %>%
    filter(tipo_de_vivienda == filtro2) %>%
    filter(!is.na(numeric))

  # Para que no me seleccione las últimas 6 filas, que no hacen parte de la tabla
  tabla_selection = head(tabla_selection, nrow(tabla_selection) - 6)

  # Selección del año y mes de inicio

  ## Año inicial
  year_init = year(na.omit(tabla$date)[1])

  ## Mes inicial
  mes_init = month(na.omit(tabla$date)[1])

  # Creación del objeto ts
  ts_obj = ts(tabla_selection$numeric, start = c(year_init, mes_init), frequency = 12)

  # Creación del objeto xts
  xts_obj = as.xts(ts_obj)
  colnames(xts_obj) = c(nombre_variable)

  # Retorna el objeto xts
  return(xts_obj)

}
```

4.4.2.4 Función para el procesamiento de las bases de datos de la Aerocivil

La función `procesamiento_Aerocivil` corresponde a la función que se usa para procesar las bases de datos que contienen información mensual que han sido descargadas de `CAMACOL` vía web scraping. Su estructura es muy similar a la función de procesamiento de datos del `DANE`: 1) se selecciona la tabla que contiene las celdas de interés y se almacena en la variable `particion`, luego se extrae la información numérica de las variables de interés y finalmente se crear el objeto `xts`.

```
# Función para procesar las bases de la Aerocivil
procesamiento_Aerocivil = function(nombre_base, nombre_sheet, palabra_identificacion_tabla, nombre_variable, filtro1){

  # Importación base de datos
  base = tidyxl::xlsx_cells(nombre_base)

  # Esquina superior izquierda de la tabla
  esquina_sup_izq_tabla = dplyr::filter(base, character == palabra_identificacion_tabla)

  # Particion de la base que contiene la informacion de la tabla de interés dentro de la hoja de excel
  particion = partition(base, esquina_sup_izq_tabla)

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up", "encabezados") %>%
    select(row, col, data_type, numeric, character, date, encabezados)

  # Selección del año y mes de inicio

  ## Año inicial
  year_init =  year(na.omit(tabla$date)[1])

  ## Mes inicial
```

```
  mes_init = month(na.omit(tabla$date)[1])

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    filter(encabezados == filtro1) %>%
    filter(!is.na(numeric))

  # Suma de los valores de la columna de interés
  tabla_sum = tabla_selection %>%
    summarize(suma = sum(numeric))

  # Creación del objeto ts
  ts_obj = ts(tabla_sum$suma, start = c(year_init, mes_init), frequency = 12)

  # Creación del objeto xts
  xts_obj = as.xts(ts_obj)
  colnames(xts_obj) = c(nombre_variable)

  # Retorna el objeto xts
  return(xts_obj)

}
```

4.4.2.5 Función para el procesamiento de las bases de datos de la FNC

La función **procesamiento_FNC** corresponde a la función que se usa para procesar las bases de datos que contienen información mensual que han sido descargadas de **CAMACOL** vía web scraping. Su estructura es muy similar a la función de procesamiento de datos del **DANE**: 1) se selecciona la tabla que contiene las celdas de interés y se almacena en la variable **particion**, luego se extrae la información numérica de las variables de interés y finalmente se crear el objeto xts.

```
# Función para procesar las bases de la FNC
procesamiento_FNC = function(nombre_base, nombre_sheet, palabra_identificacion_tabla, nombre_variable, filtro1){

  # Importación base de datos
  base = tidyxl::xlsx_cells(nombre_base, sheets = nombre_sheet)

  # Esquina superior izquierda de la tabla
  esquina_sup_izq_tabla = dplyr::filter(base, character == palabra_identificacion_tabla)

  # Particion de la base que contiene la informacion de la tabla de interés dentro de la hoja de excel
  particion = partition(base, esquina_sup_izq_tabla)

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up", "encabezados") %>%
    behead("left", "fecha") %>%
    select(row, col, data_type, numeric, character, date, encabezados, fecha)

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    filter(encabezados == filtro1) %>%
    filter(!is.na(numeric))

  # Selección del año y mes de inicio

  ## Año inicial
  year_init = year(na.omit(tabla_selection$fecha)[1])

  ## Mes inicial
  mes_init = month(na.omit(tabla_selection$fecha)[1])

  # Creación del objeto ts
  ts_obj = ts(tabla_selection$numeric, start = c(year_init, mes_init), frequency = 12)
```

```
  # Creación del objeto xts
  xts_obj = as.xts(ts_obj)
  colnames(xts_obj) = c(nombre_variable)

  # Retorna el objeto xts
  return(xts_obj)

}
```

4.4.2.6 Función para el procesamiento de las bases de datos de la DIAN

La función `procesamiento_DIAN` corresponde a la función que se usa para procesar las bases de datos que contienen información mensual que han sido descargadas de `CAMACOL` vía web scraping. Su estructura es muy similar a la función de procesamiento de datos del `DANE`: 1) se selecciona la tabla que contiene las celdas de interés y se almacena en la variable `particion`, luego se extrae la información numérica de las variables de interés y finalmente se crear el objeto `xts`.

```
# Función para procesar las bases de la DIAN
procesamiento_DIAN = function(nombre_base, nombre_sheet, palabra_identificacion_tabla, nombre_variable, filtro1){

  # Importación base de datos
  base = tidyxl::xlsx_cells(nombre_base)

  # Esquina superior izquierda de la tabla
  esquina_sup_izq_tabla = dplyr::filter(base, character == palabra_identificacion_tabla)

  # Particion de la base que contiene la informacion de la tabla de interés dentro de la hoja de excel
  particion = partition(base, esquina_sup_izq_tabla)

  # Selección de las celdas de Excel que contienen la info de la tabla
  tabla = particion$cells[[1]] %>%
    filter(!is_blank) %>%
    behead("up", "encabezados") %>%
    behead("left", "año") %>%
    behead("left", "mes") %>%
    select(row, col, data_type, numeric, character, date, encabezados, año, mes)

  # Selección de la columna con la información de interés
  tabla_selection = tabla %>%
    filter(encabezados == filtro1) %>%
    filter(!str_detect(año, "TOTAL")) %>%
    filter(!is.na(numeric))

  # Selección del año y mes de inicio

  ## Año inicial
  year_init = tabla_selection$año[1]

  ## Mes inicial
  mes_init = deteccion_primer_mes(tabla_selection$mes[1])

  # Creación del objeto ts
  ts_obj = ts(tabla_selection$numeric, start = c(year_init, mes_init), frequency = 12)

  # Creación del objeto xts
  xts_obj = as.xts(ts_obj)
  colnames(xts_obj) = c(nombre_variable)

  # Retorna el objeto xts
  return(xts_obj)

}
```

4.5 Funciones para el procesamiento de todas las bases de datos y para la actualización de la base final usada por los modelos de frecuencias mixtas para el nowcast de PIB

4.5.1 Funciones para el procesamiento de todas las bases de datos vía la base de datos `input_web_scraping_MF.xlsx`

Entendiendo cada una de las funciones de procesamiento de las diferentes fuentes de información, se procede a explicar la función que itera a través de las hojas de procesamiento, es decir aquellas hojas diferentes a la hoja **descarga** del Excel **input_web_scraping_MF.xlsx** y que usa la información en cada una de las filas de dichas hojas del archivo Excel para hacer el procesamiento de las variables que harán parte de la base final que usan los modelos de frecuencia mixta para hacer el nowcast de PIB.

4.5.1.1 Función auxiliar `funcion_try_catch`

En primer lugar, se definie una función auxiliar **funcion_try_catch** cuya funcionalidad consiste en identificar el nombre correcto de la hoja del archivo de Excel que se importo que contiene la variable de interés. La necesidad de la función radica en que en algunas ocasiones el nombre de las hojas de Excel pueden tener un espacio en blanco al final del nombre, por lo que si no se contempla esa posibilidad es posible que el código no pueda identificar la hoja de Excel en cuestión. Los parámetros de la función son:

- **funcion_procesamiento_base**: Nombre específico de la función que procesará la base da datos. El nombre corresponderá a alguna de las funciones para el procesamiento de las bases descargadas vía web scraping por fuente de información que se definió en la sección anterior. Por tanto, el parámetro como tal en si mismo es una función definida anteriormente, siguiendo un enfoque de programación funcional. Las funciones que pueden ser parte del parámetro pueden ser cualquier de las siguientes funciones:
  - **procesamiento_DANE**
  - **procesamiento_DANE_trimestral**
  - **procesamiento_Camacol**
  - **procesamiento_Aerocivil**
  - **procesamiento_FNC**
  - **procesamiento_DIAN**
- **nombre_base:** El nombre del archivo .xlsx que se va a procesar. Proviene de la hoja de procesamiento de la base **input_web_scraping_MF.xlsx**.
- **nombre_sheet:** El nombre de la hoja del archivo Excel que contiene la variable de interés. No contiene espacios ni antes ni después del nombre. Proviene de la hoja de procesamiento de la base **input_web_scraping_MF.xlsx**.
- **…** : Todas las variables adicionales que requieren las función específicada en el parámetro **funcion_procesamiento_base** para poderse ejecutar.

Ahora bien, la función **funcion_try_catch** inicialmente (sentencia **tryCatch**) intenta procesar la base de datos con el nombre de la hoja del archivo Excel especificado en el parámetro **nombre_sheet** y en caso de encontrar un error significa que el nombre de la hoja de Excel está mal escrito y debe contener un espacio en blanco al final de ésta por lo que se agrega dicho espacio en blanco al nombre de la hoja del archivo de Excel y se vuelve a intentar procesar la base de datos en cuestión.

```
# Función que ejecuta "try-catch" para cada variable procesada, dependiende de si el nombre de la hoja Excel en "nombre_sheet
## Nota: Note que acá se hace una especie de programación funcional en la medida que dentro de esta función entra como paráme
##       de las que fuerón definidas arriba. El elipisis "..." denota un número arbitario de parámetros que pueden entrar den
funcion_try_catch = function(funcion_procesamiento_base, nombre_base, nombre_sheet, ...){

  # Nota:
  ### Try: Nombre de la "Sheet" de excel no tiene espacio al final del nombre de la hoja de excel
  ### Catch: Nombre de la "Sheet" de excel tiene espacio al final del nombre de la hoja de excel

  tryCatch({

    # Función que procesa las bases de datos cuándo
    xts_obj = funcion_procesamiento_base(nombre_base, nombre_sheet, ...)

    return(xts_obj)

  }, error = function(e) {
    # Código que se ejecuta cuando no se puede desestacionalizar por default
    # En este caso se aplica la opción "outlier = NULL"

    nombre_sheet = paste0(nombre_sheet, " ")

    # Se emplea la desetacionalización con la opción "outlier = NULL"
```

```
    xts_obj = funcion_procesamiento_base(nombre_base, nombre_sheet, ...)

    return(xts_obj)

  })
}
```

4.5.1.2 Función `procesamiento_datos_modelos`

La función **`procesamiento_datos_modelos`** se encarga de procesar todas los archivos de Excel descargados vía **`web scraping`** haciendo uso de la información que se encuentra en las hojas de procesamiento, es decir aquellas hojas diferentes a la hoja de **`descarga`** la base de datos **`input_web_scraping_MF.xlsx`**.

La función contiene los siguientes parámetros:

- **`input_procesamiento_bases`**: Es el parámetro que representa la base de datos **`input_web_scraping_MF.xlsx`**. Contiene toda la información necesaria para hacer el procesamiento de los datos descargados vía web scraping y extraer las variables que harán parte de la base de datos final que usan los modelos de Mixed Frequency para el nowcast de PIB. Cada fila de una determinada hoja de procesamiento representa una nueva variable que será parte de la mencionada base de datos final.
- **`year_filter`**: Especifica el año desde donde debe empezar las serie de tiempo de las variable que harán parte de la base de datos final de los modelos de Mixed Frequency para el nowcast de PIB. Por ejemplo, si se específica el año 2000, todas las series de las variables de la hoja de procesamiento que se esté procesando empezarán desde el año 2000 en la base de datos final.
- **`generar_base`**: Variable lógica (**`booleana`**) donde **`true`** significa que se generá un archivo Excel con la información de las variables generadas por la función y **`false`** significa que no se generá dicho archivo Excel. Generalmente, se deja en false el parámetro porque la base que resulta de esta función es una base intemerdia en la medida que contiene la información actualizada pero no necesariamente toda la información histórica y sirve como insumo para actualizar la base de datos final que no se ha actualizado aún (**`base_final_pre_actualizada.xlsx`**) .
- **`name_excel`**: Nombre del archivo Excel en caso de que se genere dicho archivo, es decir, en caso de tal de que le parámetro **`generar_base`** sea **`true`**.
- **`dir_salida`**: Indica la direccion del directorio donde se debería guardar el archivo Excel en caso de crearse.
- **`path_microsoft_office`**: Indica la dirección del directorio donde se encuentra instalado el programado de Microsoft Office. Esto es necesario, en la medida que la función usa la función **`verificar_si_es_xls`** donde esta última, que convierte archivos **`.xls`** en **`.xlsx`** en caso de que el archivo **`Excel`** venga en ese formato, requiere del **`path_microsoft_office`** para poder encontrar el programa **`excelcnv.exe`** que realiza la conversión de archivos.
- **`path_bases_descargadas`**: Indica la direccion del directorio donde se encuentran las bases de datos descargadas vía **`web scraping`**.
- **`tmp_sleep`**: Indica el tiempo de espera que se suspende el programa mientras la función **`verificar_si_es_xls`** realiza la conversión de los archivos en formato **`.xls`** a formato **`.xlsx`** en caso de ser necesario.

Inicialmente, la función identifica las variables que van a tener un tratamiento especial, en particular, todas las variables de exportaciones e importaciones van a tener un tratamiento diferenciado en la medida que deben se reescaladas en sus unidades. Luego, se itera a través de las filas de la hoja de procesamiento específica del Excel **`input_web_scraping_MF.xlsx`** que se esté procesando y se usa la información de cada una de las filas sobre las que se está iterando para realizar el procesamiento de cada variable.

Para realizar lo anterior, se nota que en cada iteración de la función se ejecuta inicialmente la función **`verificar_si_es_xls`** con lo que se busca saber si el archivo Excel que contiene la variable especificada en ese fila específica en donde se encuentra la iteración tiene formato **`.xls`** o no y en caso de tenerlo hacer la conversión a formato **`.xlsx`**. Luego, hay un condicional que utiliza la información en la columna **`Fuente`** de la hoja de procesamiento de **`input_web_scraping_MF.xlsx`** que sirve para identificar la función de procesamiento específica que se debe usar dependiendo de la fuente de información de donde provenga la base de datos que se va a procesar. El resultado de lo anterior, es que cada función de procesamiento en cada iteración específica genera una serie de tiempo de tipo **`xts`** para la variable que se esté procesando en dicha iteración. Para guardar todas las series de tiempo en un solo objeto se crea el objeto **`xts_matrix`** el cuál es una matriz de objetos **`xts`** conde en cada iteración se va agregando como columna adicional la serie de tiempo de la variable sobre la que se está iterando en ese instante hasta completar la iteración y llenar la matriz. Ahora bien, si las variables tienen un tratamiento especial se realiza dicho tratamiento (como sería en el caso de las variables de importaciones y exportaciones) y finalmente se realiza el procesamiento necesario para transformar el objeto **`xts_matrix`** en un archivo Excel que se exporta en caso tal de que el usuario lo especfique así. Es importante destacar que la función retorna es el objeto **`xts_matrix`** y **no** el archivo de Excel generado, dado que el objeto **`xts_matrix`** servirá como insumo para la función **`actualizacion_bases_finales`** que actualiza la base de datos final que usan los modelos de frecuencia mixta para el nowcast de PIB.

```
# Función para el procesamiento de todos los datos
procesamiento_datos_modelos = function(input_procesamiento_bases, year_filter, generar_base, name_excel, dir_salida, path_mic

  # Vector con el nombre de las variables de "exportaciones" e "importaciones" que deben ser transformadas para tener las mis
  variables_para_transformar_impo_expo = c("impo_total_usd",
                                           "impo_consumo",
```

```
                                        "impo_intermedios",
                                        "impo_bienescapital",
                                        "expo_total_usd",
                                        "expo_tradicionales",
                                        "expo_no_tradicionales",
                                        "impo_capital_real",
                                        "expom",
                                        "impo_bienes_reales")

# Iteración a traves de la base que contiene la información de las variables que fueron descargadas
for (i in 1:nrow(input_procesamiento_bases)){

  # Verificación de si la base está en formato ".xls" y en caso de estarlo transforma la base a formato ".xlsx"
  nombre_base_verificado = verificar_si_es_xls(nombre_base = input_procesamiento_bases[i,]$nombre_base,
                                       path_microsoft_office,
                                       path_bases_descargadas,
                                       tmp_sleep)

  # Condicional para crear los objetos xts por cada base procesada (i.e. cada base que se procesa, genera un objeto xts dis

  if(input_procesamiento_bases[i,]$Fuente == "DANE"){

    # Función para generar objeto xts de las bases del DANE
    xts_obj = funcion_try_catch(funcion_procesamiento_base = procesamiento_DANE,
                                nombre_base = nombre_base_verificado,
                                nombre_sheet = input_procesamiento_bases[i,]$nombre_sheet,
                                palabra_identificacion_tabla = input_procesamiento_bases[i,]$palabra_identificacion_tabla,
                                nombre_variable = input_procesamiento_bases[i,]$nombre_variable,
                                filtro1 = input_procesamiento_bases[i,]$filtro1,
                                filtro2 = input_procesamiento_bases[i,]$filtro2,
                                filtro3 = input_procesamiento_bases[i,]$filtro3)


  }else if(input_procesamiento_bases[i,]$Fuente == "DANE_TRIMESTRAL"){

    # Función para generar objeto xts de las bases del DANE
    xts_obj = funcion_try_catch(funcion_procesamiento_base = procesamiento_DANE_trimestral,
                                nombre_base = nombre_base_verificado,
                                nombre_sheet = input_procesamiento_bases[i,]$nombre_sheet,
                                palabra_identificacion_tabla = input_procesamiento_bases[i,]$palabra_identificacion_tabla,
                                nombre_variable = input_procesamiento_bases[i,]$nombre_variable,
                                filtro1 = input_procesamiento_bases[i,]$filtro1)


  }else if (input_procesamiento_bases[i,]$Fuente == "CAMACOL"){

    # Función para generar objeto xts de las bases de Camacol
    xts_obj = funcion_try_catch(funcion_procesamiento_base = procesamiento_Camacol,
                                nombre_base = nombre_base_verificado,
                                nombre_sheet = input_procesamiento_bases[i,]$nombre_sheet,
                                palabra_identificacion_tabla = input_procesamiento_bases[i,]$palabra_identificacion_tabla,
                                nombre_variable = input_procesamiento_bases[i,]$nombre_variable,
                                filtro1 = input_procesamiento_bases[i,]$filtro1,
                                filtro2 = input_procesamiento_bases[i,]$filtro2)

  }else if(input_procesamiento_bases[i,]$Fuente == "AEROCIVIL"){

    # Función para generar objeto xts de las bases de la Aerocivil
    xts_obj = funcion_try_catch(funcion_procesamiento_base = procesamiento_Aerocivil,
                                nombre_base = nombre_base_verificado,
                                nombre_sheet = input_procesamiento_bases[i,]$nombre_sheet,
                                palabra_identificacion_tabla = input_procesamiento_bases[i,]$palabra_identificacion_tabla,
                                nombre_variable = input_procesamiento_bases[i,]$nombre_variable,
                                filtro1 = input_procesamiento_bases[i,]$filtro1)

  }else if(input_procesamiento_bases[i,]$Fuente == "FNC"){
```

```
    # Función para generar objeto xts de las bases de la FNC
    xts_obj = funcion_try_catch(funcion_procesamiento_base = procesamiento_FNC,
                                nombre_base = nombre_base_verificado,
                                nombre_sheet = input_procesamiento_bases[i,]$nombre_sheet,
                                palabra_identificacion_tabla = input_procesamiento_bases[i,]$palabra_identificacion_tabla,
                                nombre_variable = input_procesamiento_bases[i,]$nombre_variable,
                                filtro1 = input_procesamiento_bases[i,]$filtro1)

  }else if(input_procesamiento_bases[i,]$Fuente == "DIAN"){

    # Lista de archivos en el directorio
    nombres_bases_de_datos_descargadas = list.files()

    # Uso de la función "str_detect" del paquete "stringr" para detectar el archivo descargado de la DIAN con la palabra "re
    nombre_base_DIAN = nombres_bases_de_datos_descargadas[str_detect(nombres_bases_de_datos_descargadas, "recaudo-mensual")]

    # Función para generar objeto xts de las bases de la DIAN
    xts_obj = funcion_try_catch(funcion_procesamiento_base = procesamiento_DIAN,
                                nombre_base = nombre_base_DIAN,
                                nombre_sheet = input_procesamiento_bases[i,]$nombre_sheet,
                                palabra_identificacion_tabla = input_procesamiento_bases[i,]$palabra_identificacion_tabla,
                                nombre_variable = input_procesamiento_bases[i,]$nombre_variable,
                                filtro1 = input_procesamiento_bases[i,]$filtro1)

  }else if(input_procesamiento_bases[i,]$Fuente == "BANREP"){

    # Función para generar objeto xts de las bases de la BANREP
    xts_obj = funcion_try_catch(funcion_procesamiento_base = procesamiento_Banrep,
                                nombre_base = nombre_base_verificado,
                                nombre_sheet = input_procesamiento_bases[i,]$nombre_sheet,
                                palabra_identificacion_tabla = input_procesamiento_bases[i,]$palabra_identificacion_tabla,
                                nombre_variable = input_procesamiento_bases[i,]$nombre_variable,
                                filtro1 = input_procesamiento_bases[i,]$filtro1,
                                filtro2 = input_procesamiento_bases[i,]$filtro2)

  }

  # Condicional para generar la "matriz" de objetos xts

  # Si es la primer vez que se itera se crea el objeto "xts_matrix" que es la matriz que almacena todas las series de tiempo
  if(i == 1){

    # Inicalización del objeto "xts_matrix" en la primera iteración
    xts_matrix = xts_obj

  }else{

    # En cada iteración se llenar la matriz de objetos xts con un nueva columna ("serie" para cada variable)
    xts_matrix = merge.xts(xts_matrix, xts_obj)

  }

  # Transformación de las variables para que tengan las mismas unidades que las variables que se encuentran en la base origi

  ## Transformación de las variables de importaciones y exportaciones para que tengan las mismas unidades que las variables
  if(colnames(xts_obj) %in% variables_para_transformar_impo_expo){

    # Divide la columna "colnames(xts_obj)" de la matriz xts_matrix por 1000
    xts_matrix[, colnames(xts_obj)] = apply(xts_matrix[, colnames(xts_obj)], 2, function(x) x/1000)

  }

}

# Procesamiento luego de generar el objeto "xts_matrix" (i.e. la matriz de objetos xts)
```

```r
  ## Se filtra las series, para que solo contengan información desde "year_filter" en adelante
  xts_filtered = xts_matrix[lubridate::year(index(xts_matrix)) >= year_filter]

  ## Se estrae las fechas del objeto "xts_filtered"
  Fecha = index(xts_filtered)

  ## Se transforma el objeto "xts_filtered" en una data frame
  todos_df_filter = as.data.frame(xts_filtered)

  ## Se transforma los rownames del dataframe en la columna de fechas
  todos_df_filter = rownames_to_column(todos_df_filter, var = "Fecha")

  ## Se guarda dicha columna de fechas con los valores que vienen del index de "xts_filtered"
  todos_df_filter = todos_df_filter %>%
    mutate(Fecha = Fecha)

  ## Se específica el directorio de salida donde se almacenará la base de datos final
  setwd(dir_salida)

  ## Condicional para decidir si se guarda o no como archivo excel la base de datos con las variables procesadas pero aún sin
  if (generar_base){

    ## Se exporta el dataframe "todos_df_filter como un archivo Excel
    write.xlsx(todos_df_filter, name_excel)

  }

  ## Se retorna el objeto "xts_filtered" que va a servir como insumo para la función "actualizacion_bases_finales" que actual
  return(xts_filtered)

}
```

¿Cuántas veces hay que ejecutar la función `procesamiento_datos_modelos`?

Como se mencionó en secciona anteriores, la base de datos `input_web_scraping_MF.xlsx` puede tener varias hojas de procesamiento dependiendo de como se quieran organizar las variables que irán en la base de datos final que usan los modelos de frecuencias mixtas para el nowcast de PIB. Si se quieren tres hojas distintas en la base final de datos, entonces se necesitan tres hojas de procesamiento con la información de cada variable que va a pertenecer a cada hoja en específico. Ahora bien, la función `procesamiento_datos_modelos` se debe correr tantas veces haya hojas de procesamiento en la base de datos `input_web_scraping_MF.xlsx`. Si la mencionada base de datos tiene tres hojas de procesamiento, entonces la función `procesamiento_datos_modelos` se debe correr tres veces, una por cada hoja de procesamiento.
En el caso particular de la base final que usan los modelos de frecuencia mixta para el nocast de PIB, se debe correr dos veces porque hay dos hojas de procesamiento.

4.5.2 Función para actualizar la base de datos final que usan los modelos de frecuencia mixta para el nowcast de PIB

Finalmente, se actualiza la base de datos final que usan los modelos de frecuencia mixta para el nowcast de PIB. Para ello, se usa la función `actualizacion_bases_finales` que lo que hace es utilizar la información actualizada de las varaibles que fuerón descargados, procesados y unificadas en una nueva base, para actualizar la base de datos final que no se ha actualizado aún `base_final_pre_actualizacion.xlsx`. Los parámetros de la función son los siguientes:

- `base_original`: Parámetro que representa la base de datos final que se debe actualizar. En el caso de los modelos de frecuencia mixta sería la base `base_final_pre_actualizacion.xlsx`.
- `base_actualizada_xts`: Parámetro que representa la base de datos que tiene la información de las variables actualizadas de la base de datos final. Es una matriz de objetos `xts` (`matrix_xts`) y proviene de ejecutar la función `procesamiento_datos_modelos`.
- `year_filter`: Especifica el año desde donde debe empezar las serie de tiempo de las variable que harán parte de la base de datos final de los modelos de Mixed Frequency para el nowcast de PIB. Por ejemplo, si se específica el año 2000, todas las series de las variables de la hoja de procesamiento que se esté procesando empezarán desde el año 2000 en la base de datos final.
- `periodicidad`: Denota la periodicidad de las series de tiempo que se van a actualizar. Para series de tiempo mensuales se usa `periodicidad = "1 month"`, mientras que para series de tiempo trimestrales se usa `periodicidad = "3 months"`.
- `generar_base`: Variable lógica (`booleana`) donde `true` significa que se generá un archivo Excel con la información de las variables generadas por la función y `false` significa que no se generá dicho archivo Excel.

- `name_excel`: Nombre del archivo Excel en caso de que se genere dicho archivo, es decir, en caso de tal de que le parámetro `generar_base` sea `true`.
- `start_fecha`: Especifica la fecha desde donde inician las series de tiempo de las variables que se encuentran en la base de datos final que no se ha actualizado aún, es decir, las series de tiempo en la base representada por el parámetro `base_original` que en el caso de los modelos de frecuencia mixta vendría a ser la base `base_final_pre_actualizacion.xlsx`. Indica año, mes y día de inicio de dichas series en formato `YYYY-MM-DD`.
- `dir_salida`: Indica la direccion del directorio donde se debería guardar el archivo Excel en caso de crearse.

Incialamente, la función `actualizacion_bases_finales` transforma la base de datos que se encuentra almacenada en el parámetro `base_original` en una matriz de series de tiempo de objetos `xts`. Lo anterior, con la finalidad de que sea compatbile su tratamiento y manipulación con la base de datos `base_actualizada_xts` que ya se encuentra como una matriz de series de tiempo de objetos xts al ser el resultado de la función `procesamiento_datos_modelos`. Luego se extraen los nombres de las dos bases de datos con los comandos `names_base_original = names(base_original_xts)` y `names_base_actualizada = names(base_actualizada_xts)` (se destaca que las dos bases de datos no tienen necesariamente la misma cantidad de varaibles, dado que no todas las variables de la base de datos final se actualizan dado que algunas variables puede que se actualizen de manera manual y no vía `web scrapinig`, por lo que las dimensiones de `names_base_original` y `names_base_actualizada` no tiene que coincidir).

Posteriormente, se procede a actualizar la información en la base de datos final. Para realizar dicha actualización se itera sobre cada uno de los nombre de las variables que se encuentran en el vector de nombres de varaibles `names_base_original` y lo primero que se revisa es si dicha variable se actualizó o no vía web scraping mediante la condición `logical_vector = names_base_original[i_orig] == names_base_actualizada` dado que si me da un valor de 1 indica que el nombre de la variable en `names_base_original[i_orig]` se encuentra en `names_base_actualizada` y por ende se actualizo vía web scraping, mientras que si da un valor de 0 significa que `names_base_original[i_orig]` no se encuentra en `names_base_actualizada` y por ende no se actualizo vía web scraping.

En caso de que la condición sea 0 y no se haya actualizada la variable vía web scraping la función no modifica la variable y la retorna tal cuál estaba en la base de datos pre-actualización mediante la línea de código `new_xts = base_original_xts[, i_orig]`. En caso de que la condición sea 1 y sí se haya actualizado la variable vía web scraping la función identifica el índice de `names_base_actualizada` donde se encuentra la variable que se va a actualizar `i_actual = which(logical_vector)` y luego usa la función `merge.xts` `xts_merge = merge.xts(base_original_xts[, i_orig], base_actualizada_xts[, i_actual])` para compara la versión desactualizada `base_original_xts[, i_orig]` y la versión actualizada de la variable `base_actualizada_xts[, i_actual]`. Luego extraigo las fecha del objeto `xts_merge` dado que esas serán las fechas finales de la seirie de tiempo resultante de la actualización. Posteriormente, se itera a través de las filas del objeto `xts_merge` y se mira en cada fila u observación si hay información nueva o no para esa fila u observación en específico y se llena el vector `new_vect` con las observaciones que resulten del proceso de actualización, donde por observación nos referimos a cada valor específico en una fecha determinada de la serie de tiempo. Es decir, si en esa observación la serie de tiempo de la variables actualizada tiene un NA o está vacía entonces esa observación no se actualiza y se almacena en `new_vect` la observación que se encuentra en la variable antes de ser actualizada (dado que no hubo una actualización), de lo contrario en caso de que si haya una actualización se almacena el valor de dicha observación de la variable actualizada en `new_vect`. Al final de la actualización, se retorna el vector `new_vect` como un objeto `xts` `new_xts = xts(new_vect, order.by = xts_merge_index)` cuyo índice temporal está dado por `xts_merge_index`. Todo lo anterior lo ilustra el siguiente código:

```
    # Vector lógico que permite detectar en que parte de la "base_actualizada_xts" se encuentra la variable de la "base_origi
    # Lo que se está haciendo es una comparación de nombres de las dos bases ("base_actualizada_xts" y "base_original_xts") y
    logical_vector = names_base_original[i_orig] == names_base_actualizada

    # Condición para deterctar si es necesario actualizar los datos de la variable o no.
    ## sum(logical_vector) == 1: Indica que sí, dado que se descargarón datos nuevos
    ## sum(logical_vector) == 0: Indica que no, dado que no se descargarón datos nuevos
    if (sum(logical_vector) == 1){

      # Vector que va almacenar los datos de la variable. Si no se han actualiado deja los de la variable de la base original
      new_vect = c()

      # Índice que indica en que posición de la "base_actualizada_xts" se encuentra la variable de la "base_original_xts"
      i_actual = which(logical_vector)

      # Genero el objeto xts que incluye las dos series de tiempo de la variable de interés, i.e., la anterior ("base_original
      xts_merge = merge.xts(base_original_xts[, i_orig], base_actualizada_xts[, i_actual])

      # Extraigo el ínidice temporal del objeto "xts_merge" que contiene las dos series de tiempo de la variable de ineterés
      xts_merge_index = index(xts_merge)

      # Extaigo el número de observacione que se encuentran en el objeto "xts_merge"
      xts_merge_nrow = dim(xts_merge)[1]

      # Lleno el nuevo vector que va a contener la información de la variable de interés en la base actualizada
      for (i_merge in 1:xts_merge_nrow){
```

```
      # Si la serie actualizada ("base_actualizada_xts") de la variable tiene NA, entonces se dejan los valores de la serie
      if (is.na(xts_merge[, 2][i_merge])){

        # Se llena con los valores de la serie original ("base_original_xts") de la variable
        new_vect[i_merge] = xts_merge[, 1][i_merge]

      }else{

        # De lo contrario, si la serie actualizada no tiene NA, entonces se llena con los valores de la serie actualizada (
        new_vect[i_merge] = xts_merge[, 2][i_merge]
      }

    }

    # Se transforma el vector que se acaba de llenar en un objeto xts
    new_xts = xts(new_vect, order.by = xts_merge_index)

  }else{

    # En caso de que no haya un versión actualizada de la variable (i.e. la varible no se encuentre en "base_actualizada_xts
    new_xts = base_original_xts[, i_orig]

  }
```

Finalmente, se genera la matriz de objetos **xts** **xts_matrix** cuyas columnas serán las series de tiempo de las variables actualizadas en el procedimiento anterior

```
  # Condicional para generar la "matriz" de objetos xts

  # Si es la primer vez que se itera se crea el objeto "xts_matrix" que es la matriz que almacena todas las series de tiempo
  if(i_orig == 1){

    # Inicalización del objeto "xts_matrix" en la primera iteración
    xts_matrix = new_xts

  }else{

    # En cada iteración se llenar la matriz de objetos xts con un nueva columna ("serie" para cada variable)
    xts_matrix = merge.xts(xts_matrix, new_xts)

  }
```

y finalmente se genera el achivo Excel con la información en el objeto **xts_matrix** tal cual se hizo en la función **procesamiento_datos_modelos**.

```
# Se renombran las variables de "xts_matrix" para que los nombres de las series correspondan a los mismos nombre de las varial
  names(xts_matrix) = names_base_original

  # Se filtra las series, para que solo contengan información desde "year_filter" en adelante
  xts_filtered = xts_matrix[lubridate::year(index(xts_matrix)) >= year_filter]

  # Se estrae las fechas del objeto "xts_filtered"
  Fecha = index(xts_filtered)

  # Se transforma el objeto "xts_filtered" en una data frame
  todos_df_filter = as.data.frame(xts_filtered)

  # Se transforma los rownames del dataframe en la columna de fechas
  todos_df_filter = rownames_to_column(todos_df_filter, var = "Fecha")

  # Se guarda dicha columna de fechas con los valores que vienen del index de "xts_filtered"
  todos_df_filter = todos_df_filter %>%
    mutate(Fecha = my(Fecha))

  # Se específica el directorio de salida donde se almacenará la base de datos final
  setwd(dir_salida)
```

```
  if (generar_base){

    # Se exporta el dataframe "todos_df_filter como un archivo Excel
    write.xlsx(todos_df_filter, name_excel)

  }
```

El código completo de la función es el siguiente:

```
# Función para actualizar las bases de datos originales
actualizacion_bases_finales = function(base_original, base_actualizada_xts, year_filter, periodicidad, generar_base, name_exc

  # Parte del código que prepara las base de datos antes del proceso de actualización de la base de datos original que se qui

  ## "base_original_xts"

  # Fecha de inicio de la base de datos original ("base_original") que se quiere actualizar
  fecha_inicio = ymd(start_fecha)

  # Genera la variable que contiente todas las fechas de la base de datos original ("base_original") que se quiere actualizar
  fechas = as.yearmon(seq(fecha_inicio, by = periodicidad, length.out = nrow(base_original)))

  # Modifica "base_original" para que incluya las fechas de la base de datos
  base_original = base_original %>%
    mutate(Fecha = fechas)

  # Transforma "base_original" (base de datos que contiene las series originales de le base de datos original antes de ser ac
  base_original_xts = xts(base_original[, -1], order.by = base_original$Fecha)

  # Vector con los nombres de las variables que se encuentran en la base de datos con las series originales ("base_original_x
  names_base_original = names(base_original_xts)

  ## "base_original_xts"

  # Vector con los nombres de las variables que se encuentran en la base de datos con las series actualizadas ("base_actualiz
  names_base_actualizada = names(base_actualizada_xts)


  # Parte del código de la función que actualiza las bases de datos

  # Loop1: Iteración a través de las variables de "base_original_xts"
  for(i_orig in 1:length(names_base_original)){

    # Vector lógico que permite detectar en que parte de la "base_actualizada_xts" se encuentra la variable de la "base_origi
    # Lo que se está haciendo es una comparación de nombres de las dos bases ("base_actualizada_xts" y "base_original_xts") y
    logical_vector = names_base_original[i_orig] == names_base_actualizada

    # Condición para deterctar si es necesario actualizar los datos de la variable o no.
    ## sum(logical_vector) == 1: Indica que sí, dado que se descargarón datos nuevos
    ## sum(logical_vector) == 0: Indica que no, dado que no se descargarón datos nuevos
    if (sum(logical_vector) == 1){

      # Vector que va almacenar los datos de la variable. Si no se han actualiado deja los de la variable de la base original
      new_vect = c()

      # Índice que indica en que posición de la "base_actualizada_xts" se encuentra la variable de la "base_original_xts"
      i_actual = which(logical_vector)

      # Genero el objeto xts que incluye las dos series de tiempo de la variable de interés, i.e., la anterior ("base_original
      xts_merge = merge.xts(base_original_xts[, i_orig], base_actualizada_xts[, i_actual])

      # Extraigo el ínidice temporal del objeto "xts_merge" que contiene las dos series de tiempo de la variable de ineterés
      xts_merge_index = index(xts_merge)

      # Extaigo el número de observacione que se encuentran en el objeto "xts_merge"
```

```
    xts_merge_nrow = dim(xts_merge)[1]

    # Lleno el nuevo vector que va a contener la información de la variable de interés en la base actualizada
    for (i_merge in 1:xts_merge_nrow){

      # Si la serie actualizada ("base_actualizada_xts") de la variable tiene NA, entonces se dejan los valores de la serie
      if (is.na(xts_merge[, 2][i_merge])){

        # Se llena con los valores de la serie original ("base_original_xts") de la variable
        new_vect[i_merge] = xts_merge[, 1][i_merge]

      }else{

        # De lo contrario, si la serie actualizada no tiene NA, entonces se llena con los valores de la serie actualizada (
        new_vect[i_merge] = xts_merge[, 2][i_merge]
      }

    }

    # Se transforma el vector que se acaba de llenar en un objeto xts
    new_xts = xts(new_vect, order.by = xts_merge_index)

  }else{

    # En caso de que no haya un versión actualizada de la variable (i.e. la varible no se encuentre en "base_actualizada_xts
    new_xts = base_original_xts[, i_orig]

  }

  # Condicional para generar la "matriz" de objetos xts

  # Si es la primer vez que se itera se crea el objeto "xts_matrix" que es la matriz que almacena todas las series de tiempo
  if(i_orig == 1){

    # Inicalización del objeto "xts_matrix" en la primera iteración
    xts_matrix = new_xts

  }else{

    # En cada iteración se llenar la matriz de objetos xts con un nueva columna ("serie" para cada variable)
    xts_matrix = merge.xts(xts_matrix, new_xts)

  }

}

# Se renombran las variables de "xts_matrix" para que los nombres de las series correspondan a los mismos nombre de las vari
names(xts_matrix) = names_base_original

# Se filtra las series, para que solo contengan información desde "year_filter" en adelante
xts_filtered = xts_matrix[lubridate::year(index(xts_matrix)) >= year_filter]

# Se estrae las fechas del objeto "xts_filtered"
Fecha = index(xts_filtered)

# Se transforma el objeto "xts_filtered" en una data frame
todos_df_filter = as.data.frame(xts_filtered)

# Se transforma los rownames del dataframe en la columna de fechas
todos_df_filter = rownames_to_column(todos_df_filter, var = "Fecha")

# Se guarda dicha columna de fechas con los valores que vienen del index de "xts_filtered"
todos_df_filter = todos_df_filter %>%
  mutate(Fecha = my(Fecha))

# Se específica el directorio de salida donde se almacenará la base de datos final
```

```
  setwd(dir_salida)

  if (generar_base){

    # Se exporta el dataframe "todos_df_filter como un archivo Excel
    write.xlsx(todos_df_filter, name_excel)

  }

  # Se retorna el data frame "todos_df_filter"
  return(todos_df_filter)

}
```

⚠ Antes de actualizar la base de datos final tener la última versión de dicha base pre-actualización

Es importante destacar, que cada vez que se vaya a actualizar la base de datos final, en el caso de los modelos de frecuencia mixta la base de datos `base_final_pre_actualizacion.xlsx,` se debe tener la última versión disponibile de dicha base antes de hacer la actualización. Por ejemplo, si se queire actualizar la base de datos con la nueva información disponible de mayo, se debe tener la última versión disponible de `base_final_pre_actualizacion.xlsx` a fecha de abril, y cuándo se quiera hacer la actualización con la información disponible de junio, se debe tener la última versión disponible de `base_final_pre_actualizacion.xlsx` a fecha de mayo.

La función `actualizacion_bases_finales` es agnóstica a la base de datos

La función `actualizacion_bases_finales` es agnóstica a la base de datos, con ello, lo que se quiere decir es que dicha función sirve para actualizar cualquier base de datos, y no solo la base de datos final que usan los modelos de frecuencia mixta. Lo único que se necesita es una base de datos que se necesita actualizar representada por el parámetro `base_original` y una base de datos actualizada representada por el parámetro `base_actualizada_xts` (en formato de una matriz de series `xts`) y ya con esos dos bases es posible actualizar la base de datos `base_original` que se pretende actualizar.

¿Cuántas veces hay que ejecutar la función `actualizacion_bases_finales`?

Como se mencionó en secciona anteriores, la base de datos `input_web_scraping_MF.xlsx` puede tener varias hojas de procesamiento dependiendo de como se quieran organizar las variables que irán en la base de datos final que usan los modelos de frecuencias mixtas para el nowcast de PIB. Si se quieren tres hojas distintas en la base final de datos, entonces se necesitan tres hojas de procesamiento con la información de cada variable que va a pertenecer a cada hoja en específico. Ahora bien, la función `actualizacion_bases_finales` se debe correr tantas veces haya hojas de procesamiento en la base de datos `input_web_scraping_MF.xlsx`. Si la mencionada base de datos tiene tres hojas de procesamiento, entonces la función `actualizacion_bases_finales` se debe correr tres veces, una por cada hoja de procesamiento.
En el caso particular de la base final que usan los modelos de frecuencia mixta para el nocast de PIB, se debe correr dos veces porque hay dos hojas de procesamiento.

4.5.3 Función que genera la base de datos final que usan los modelos de frecuencia mixta para el nowcast de PIB

Finalmente, la función `combinar_bases_en_un_solo_excel` genera la base de datos final en un solo archivo de Excel. Para ello, la función combina los diferentes bases de datos que resultarón de aplicar la función `actualizacion_bases_finales` a las diferentes hojas de procesamiento de la base `input_web_scraping_MF.xlsx` en una solo archivo de Excel con múltiples hojas. La función tiene los siguientes parámetros:

- `name_excel` : Nombre del archivo de Excel de la base de datos final actualizada
- `dir_salida` : Dirección del dirctorio donde se almacenará la base de datos final
- `...` : Las bases de datos actualizdas por la función `actualizacion_bases_finales`, donde cada base de datos específica se convertirá en una nueva hoja de la base de datos final que se actualizó.

```
# Función que genera la base de datos final en un solo archivo de Excel
combinar_bases_en_un_solo_excel = function(name_excel, dir_salida, ...){

  ## Se específica el directorio de salida donde se almacenará la base de datos final
  setwd(dir_salida)

  ##
```

```
  lista_bases = list(...)

  ##
  write.xlsx(lista_bases, name_excel)

}
```

La aplicación de la función `combinar_bases_en_un_solo_excel` para construir la base de datos final actualizdaa que utilizarán los modelos de frecuencias mixtas para hacer el nowcast de PIB se vería así:

```
## Base que contiene tanto los datos finales actualizados
combinar_bases_en_un_solo_excel(name_excel = "base_todas_variables_actualizadas_MF.xlsx",
                                dir_salida = bases_finales,
                                variables_mensuales = base_final_actualizada_mensual,
                                variables_trimestrales = base_final_actualizada_trimestral)
```

Bibliografía

Garmonsway, Duncan. 2023a. «tidyxl: Read Untidy Excel Files». https://CRAN.R-project.org/package=tidyxl.

———. 2023b. «unpivotr: Unpivot Complex and Irregular Data Layouts». https://CRAN.R-project.org/package=unpivotr.

Wickham, Hadley. 2014. «Tidy Data». *Journal of Statistical Software* 59 (10). https://doi.org/10.18637/jss.v059.i10.

———. 2023. «stringr: Simple, Consistent Wrappers for Common String Operations». https://CRAN.R-project.org/package=stringr.

———. 2024. «rvest: Easily Harvest (Scrape) Web Pages». https://CRAN.R-project.org/package=rvest.