



## **PRÁCTICA 1 – Web Scraping**

### Índice

1. Contexto.....	2
2. Descripción del dataset.....	4
3. Contenido.....	5
4. Agradecimientos .....	24
5. Inspiración.....	24
6. Licencia.....	24
7. Código.....	25
8. Dataset .....	26

## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

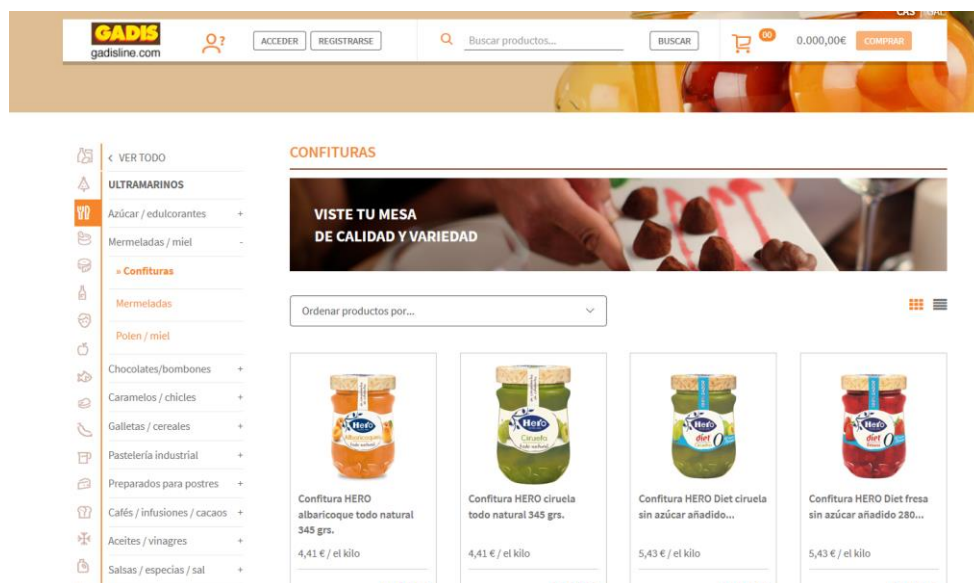
### 1. Contexto

Para esta práctica se ha realizado un web scraping de la página web [www.gadisline.com](http://www.gadisline.com). Se trata de la web de compra on line de Gadis, cadena de supermercados de gran éxito en Galicia.





El proyecto desarrollado escanea toda la estructura de productos ofertados y permite seleccionar una categoría en la que profundizar y obtener información más detallada. Esto es, nos permite descargar todos los productos de la categoría seleccionada detallando su precio, precio unitario, cantidad neta, la url de acceso y la url que obtiene su imagen. Es más, en un paso más allá y profundizando todavía más en el código de la web, se obtiene también la información nutricional del producto y sus ingredientes. Se realiza también la descarga de las imágenes miniatura en un repositorio creado automáticamente.

La limitación por categoría se debe al elevado volumen de datos manejado en la web, que demoraría el tiempo de ejecución de modo importante.




## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

[ACCEDER](#) [REGISTRARSE](#)

[BUSCAR](#)

 0.000,00€ [COMPRAR](#)

[« VOLVER A CONFITURAS](#)

Confitura HERO albaricoque todo natural  
345 grs.

**1,52 €**  
4,41 €/ el kilo

- 01 +

[AÑADIR](#)

Si quieres, puedes incluir una nota sobre este producto para tenerla en cuenta al servir tu pedido. 

Por ejemplo: caducidad superior a cuatro días

#### ANÁLISIS NUTRICIONAL

La información del nutriente se aplica cuando el producto está preparado  
Tamaño de referencia por 100 g  
Energía: (244 kcal)  
Grasas: (0,1 g)  
Hidratos de carbono: (60 g)  
Azúcares: (29 g)  
Fibra alimentaria: (0,6 g)  
Proteínas: (0,4 g)

**Nombre Legal Producto**  
Confitura Extra de Albaricoque.

**Cantidad neta del alimento:**  
345 g

#### INGREDIENTES

Albaricoque, azúcar, azúcar de caña, limón concentrado, gelificante (pectina de cítricos) y concentrado de acerola.

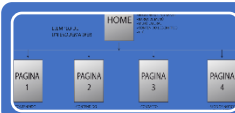
#### CONSEJOS DE CONSERVACIÓN

Conservar en lugar fresco y seco. Una vez abierto conservar en el frigorífico.

Las webs de retail como la elegida contienen información estática en cuanto al horizonte temporal, muestran los precios en el momento actual y no es posible realizar consultas del estado en días anteriores. Es por ello que, se ha desarrollado también un histórico de precios, donde se almacenarán los precios descargados de la categoría seleccionada con la fecha actual, permitiendo actualizaciones diarias y almacenando un histórico.

## 2. Descripción del dataset

El proyecto de web scraping desarrollado genera finalmente 5 datasets para recoger toda la información que se ha ido escaneando:



### CATEGORIAS

- Árbol de categorías de la gama ofertada por gadisline.com



### PRODUCTOS

- Listado de productos con su información básica.



% Valores Diarios*	
Grasa Total	14g 22%
Grasas Saturadas	5g 10%
Grasas Saturadas	5g 10%
Carbohidratos	30g 6%
Fibra Dietética	5g 20%
Azúcares	5g 10%
Sodio	0g 0%
Proteína	10mg 1%
Calorías	0mg

### INFO NUTRICIONAL

- Información nutricional aportada.



### INGREDIENTES

- Ingredientes declarados.



### HISTORICO\_PRECIOS

### 3. Contenido

#### 3.1 Descripción de los dataset:

##### 1) Dataset CATEGORIAS:

Como indicábamos anteriormente, el primer paso es escanear toda la estructura de productos alojados en la web. Para ello se ha de leer el árbol situado a la izquierda de la página y sus tres posibles niveles:



Esto genera un dataset con 6 columnas:

- Nivel 1: Texto del nivel 1.
- Nivel 2: Texto del nivel 2.
- Nivel 3: Texto del nivel 3.
- Url: Url para acceder al nivel 3.
- NumArticulos: Número de artículos que contiene el nivel 3.
- UltActualizacion: Fecha en que se ha actualizado esta información.

NOTA: Aunque este dataset contiene todo el árbol de productos ofertados por gadisline.com, el campo NumArticulos solo se informará para la categoría seleccionada, aunque es posible ir almacenando información de varias categorías en ejecuciones consecutivas.

##### 2) Dataset PRODUCTOS:

Una vez leída la estructura, el usuario ha de seleccionar una categoría a descargar y se procede a la captura de la información de los productos generando el siguiente dataset:

- Id: Identificador numérico del producto. Ver ★ en la imagen siguiente.
- Nombre: Nombre del producto.
- PVP: Precio de venta al público.
- PVP\_Unidad\_Medida: Precio unitario.
- Cantidad\_Neta: Cantidad neta contenida en el producto.
- URL\_Producto: Url de acceso al detalle de producto.
- URL\_Imagen: Url de acceso a la imagen del producto.
- UltActualización: Fecha de última actualización.



### 3) Dataset INFO\_NUTRICIONAL:

En este tercer dataset se almacenará la información nutricional de cada id de producto. Se ha decidido mantener esta información por separado dado que esta información no viene informada para todos los productos y se trata de una cadena de texto de larga longitud que rompería la estructura más limitada y tradicional del dataset PRODUCTOS, con lo que le restaría legibilidad.

Se trata de un dataset sencillo con dos campos:

- Id: Identificador numérico de producto.
- Info\_nutricional: Cadena de texto con la información nutricional informada.
- UltActualizacion: Fecha de última actualización.

### ANÁLISIS NUTRICIONAL

La información del nutriente se aplica cuando el producto está preparado

Tamaño de referencia por 100 g

Energía: (244 kcal)

Grasas: (0,1 g)

Hidratos de carbono: (60 g)

Azúcares: (29 g)

Fibra alimentaria: (0,6 g)

Proteínas: (0,4 g)

## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

### 4) Dataset INGREDIENTES:

Completando la información de producto se almacenan en este dataset los ingredientes declarados para cada producto. Se trata igual que en el caso anterior de un dataset de 2 columnas, que se ha mantenido independientemente por los mismo motivos:

- Id: Identificador numérico de producto.
- Info\_nutricional: Cadena de texto con los ingredientes declarados.
- UltActualizacion: Fecha de última actualización.

#### INGREDIENTES

Albaricoque, azúcar, azúcar de caña, limón concentrado, gelificante (pectina de cítricos) y concentrado de acerola.

### 5) Dataset HISTORICO\_PRECIOS:

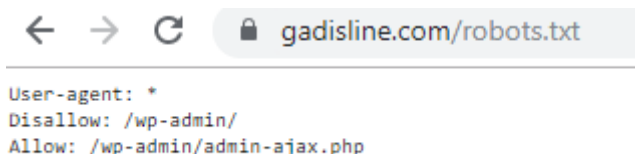
El último dataset compone el histórico de precios en base a consultas que realiza el usuario progresivamente en el tiempo. Consta de los siguientes campos:

- Id: Identificador numérico de producto.
- DiaCaptura: Día en que se ha realizado la consulta.
- PVP: Precio registrado para ese día.
- PVP\_Unidad\_Medida: Precio unitario.
- UltActualizacion: Fecha de última actualización en formato DD/MM/AA HH:SS

## 3.2 Descripción del proceso de captura:

Para la captura de toda esta información se han tenido que ir salvando diversos obstáculos presentados en el código de [www.gadisline.com](http://www.gadisline.com). El primero de ellos ha sido ya el acceso inicial. Vamos a desarrollar los hitos más importantes que se han alcanzado en la captura de la información:

En primer lugar, hemos consultado el fichero robots.txt para ver si no infringimos ninguna de las restricciones, es decir si nos autorizan a poder acceder mediante bot a las carpetas donde nosotros queremos acceder para obtener los precios y características de los productos.



```
← → ↻ 🔒 gadisline.com/robots.txt
User-agent: *
Disallow: /wp-admin/
Allow: /wp-admin/admin-ajax.php
```

## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

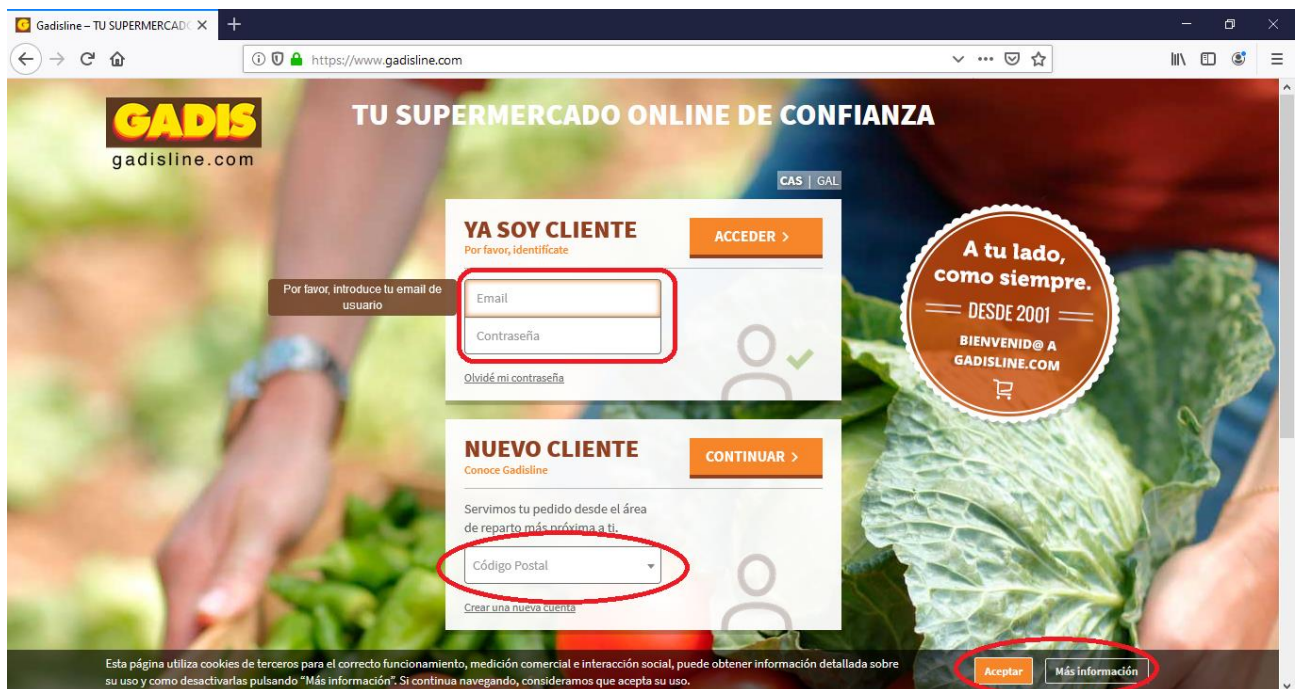
Javier Fernández y Rosa M. Suárez

En este caso no está restringido el acceso a la carpeta y páginas donde vamos a ir sacando toda la información que queremos de los productos.

La primera dificultad con la que nos encontramos es que al acceder por primera vez a la página <http://www.gadisline.com>, esta nos pide:

- O bien introducir un usuario registrado y su contraseña
- O bien introducir un código postal.

Pero incluso además nos sale en la parte inferior un aviso “Esta página utiliza cookies de terceros para el correcto funcionamiento, medición comercial e interacción social, puede obtener información detallada sobre su uso y como desactivarlas pulsando “Más información”. Si continua navegando, consideramos que acepta su uso.” como se puede ver en la parte inferior de la imagen.



Como vamos a tener que efectuar interacción con la página, hemos escogido Selenium Webdriver (que es un automatizador de navegador) para hacer la mayoría del escraqueo de esta página. Más adelante para la parte de obtener los ingredientes y análisis nutricional, al ser unas páginas ya estáticas, hemos optado por BeautifulSoup, simplemente para de esa forma utilizar 2 técnicas diferentes (se podía resolver perfectamente todo con Selenium). En este caso lo hemos realizado con Chromedriver que es descargable desde <https://chromedriver.chromium.org/downloads>

Para resolver el problema de la aceptación de las cookies, localizamos el elemento del botón Aceptar, y vemos cuáles son sus atributos. Eso lo hacemos desde el navegador web con la opción “inspeccionar elemento”. En este caso nos encontramos con lo siguiente:



## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

```
▼ <div class="row">
  ::before
  ▶ <div class="col-sm-9 col-xs-12" style="font-size: 13px;text-align: left;padding-top:
    5px;padding-bottom: 5px;">...</div>
  ▼ <div class="col-sm-3 col-xs-12">
    <a id="removecookie" class="allow">Aceptar</a> event
    ☐
    <a id="more" class="cookiemore" href="politica-de-privacidad-y-cookies" target="_blank">
      Más información</a>
  </div>
```

Lo que hemos hecho es intentar localizar este elemento cuyo “id” es “removecookie” y simular un click que es realmente lo que haríamos nosotros desde el navegador.

```
# La página tiene un botón de aceptar las cookies, que vamos a simular que aceptamos para que desaparezca de ahí
try:
    botonAceptar = driver.find_element_by_id('removecookie')
    botonAceptar.click()
except:
    pass
```

Una vez que hemos hecho eso, nos queda la parte de simular o bien que somos un usuario registrado o bien que elegimos un código postal válido. Nosotros inicialmente intentamos poner un valor de código postal como si realmente lo hubiésemos introducido (aunque realmente la página no lo permite, no deja escribir el código postal, te obliga a seleccionarlo desde un combo con unos valores ya preestablecidos).

Aún así hicimos la prueba de intentar meter el valor directamente en la caja del código postal (buscamos el elemento donde está el código postal)

```
tabindex="5" aria-labelledby="select2-
cl_postal_code-container" data-toggle="tooltip"
data-original-title="Por favor, selecciona un
código postal">
  ▼ <span class="select2-selection__rendered" id=
    "select2-cl_postal_code-container" aria-label=
    "NUEVO CLIENTE. Conoce Gadisline. Por favor,
    selecciona un código postal">
    .. <span class="select2-selection__placeholder">
      Código Postal</span> == $0
    </span>
    ▶ <span class="select2-selection__arrow" role=
      "presentation">...</span>
    </span>
  </span>
  <span class="dropdown-wrapper" aria-hidden="true">
  </span>
```

Y hemos simulado que metíamos directamente un valor en la caja. (En este caso simulábamos que poníamos el código postal 15001 en dicha caja)

```
#Vamos a simular que escribimos directamente en la caja del código postal.
#Aunque realmente esto no va a llegar a funcionar, ya que es necesario pasar por el desplegable de códigos postales.
#En este caso simulamos que introducimos el valor de 15001 para la caja del código postal.
cajaCP = driver.find_element_by_class_name("select2-selection__placeholder")
driver.execute_script('arguments[0].innerHTML = "15001";', cajaCP)
```

## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

Realmente lo hace, es decir si ejecutamos ese código el valor lo muestra tal cual, pero luego al intentar acceder (simulando pulsación del botón Continuar) nos acababa dando un error, es decir la comprobación no estaba en el elemento de la caja, sino que en el javascript que ejecuta cuando desplegamos la lista y elegimos un código postal, que luego permite continuar.

Por tanto, nos vimos obligados a “simular” que nosotros automáticamente elegíamos de la lista desplegable uno de los códigos postales. Para ello lo que hemos hecho fue localizar el elemento del selector del código postal.

```
▼<div class="col-xs-7">
  <p>Servimos tu pedido desde el área de reparto más próxima a ti.</p>
  ▼<form id="form_postalcode" class="modal_form modal_client_form">
    ...
    ▶<select id="cl_postal_code" class="postalCodesSelect form-control
    form_field select2-hidden-accessible" data-validation-engine=
    "validate[required]" title data-toggle="tooltip" data-placeholder=
    "Código Postal" tabindex="-1" data-original-title="Por favor,
    selecciona un código postal" aria-hidden="true">...</select> == $0
    ▼<span class="select2 select2-container select2-container--default
    select2-container--above" dir="ltr" style="width: 209px;">
      ▼<span class="selection">
        ▼<span class="select2-selection select2-selection--single" role=
        "combobox" aria-autocomplete="list" aria-haspopup="true" aria-
        expanded="false" title tabindex="5" aria-labelledby="select2-
        cl_postal_code-container" data-toggle="tooltip" data-original-
        title="Por favor, selecciona un código postal">
          ▼<span class="select2-selection__rendered" id="select2-
          cl_postal_code-container" aria-label="NUEVO CLIENTE. Conoce
          Gadisline. Por favor, selecciona un código postal">
            <span class="select2-selection__placeholder">Código Postal
            </span>
          </span>
        </span>
      </span>
    </div>
```

Observamos que dicho elemento era el “select” cuyo “id” = “cl\_postal\_code” y entonces con Python lo que hemos hecho fue localizarle y simular la pulsación de un ENTER, de forma que eso nos generaba el despliegue de la lista de códigos postales.

Encapsulamos el objeto selector de la página web en un en un objeto “Select” de selenium (from selenium.webdriver.support.ui import Select) y ejecutamos el método “select\_by\_value” con un valor prefijado, en este caso hemos utilizado el código postal: 15002.

```
#Simulamos que vamos a introducir el código postal
elementoSeleccionCP = driver.find_element_by_id('cl_postal_code')
elementoSeleccionCP = driver.find_element_by_id('cl_postal_code')
elementoSeleccionCP.send_keys("")
elementoSeleccionCP.send_keys(Keys.RETURN)

#Para poder seleccionar el valor dentro del combobox, creamos un Select del propio combo, que luego nos
#permite usar el método "select_by_value". En este caso escogemos el código postal 15002
valorCP = Select(driver.find_element_by_id('cl_postal_code'))
valorCP.select_by_value("15002")
```

Una vez que tenemos seleccionado el código postal, ya nos queda solamente pulsar el botón naranja “Continuar” para pasar ya a la página que nos interesa ir.

## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

```
▼<div class="row">
  ::before
  ▼<div class="col-md-12 sub_display sub_display_newclient">
    ▼<div class="row display_header">
      ::before
      ▶<div class="col-xs-7">...</div>
      ▼<div class="col-xs-5">
        ▼<button tabindex="6" aria-label="NUEVO CLIENTE. Conoce Gadisline.
        CONTINUAR" class="btn principal_btn btn_new_client ladda-button" data-
        style="zoom-out" style="font-size: 15.8571px;">
          ▶<span class="ladda-label">...</span> == $0
        </button>
      </div>
      ▶<div class="col-xs-12">...</div>
    ::after
  .
```

Localizamos el objeto button por el nombre de la clase, y simulamos la pulsación de un click.

```
#Una vez seleccionado el código postal, tenemos que simular la pulsación del botón naranja "CONTINUAR >"
botonContinuar = driver.find_element_by_class_name('btn.principal_btn.btn_new_client.ladda-button').click()

print("Entrando en la página de Gadisline donde se encuentran los precios...")
```

¡¡ Ya hemos conseguido pasar la primera página!!

Otra verificación que hemos realizado es ver realmente qué user-agent estábamos usando por si fuese necesario cambiarlo. Como realmente nosotros estamos usando el Chrome webdriver, realmente está actuando como si fuese un navegador Chrome de cualquier usuario que visita la página. Aun así, hemos obtenido y mostrado el user-agent utilizado:

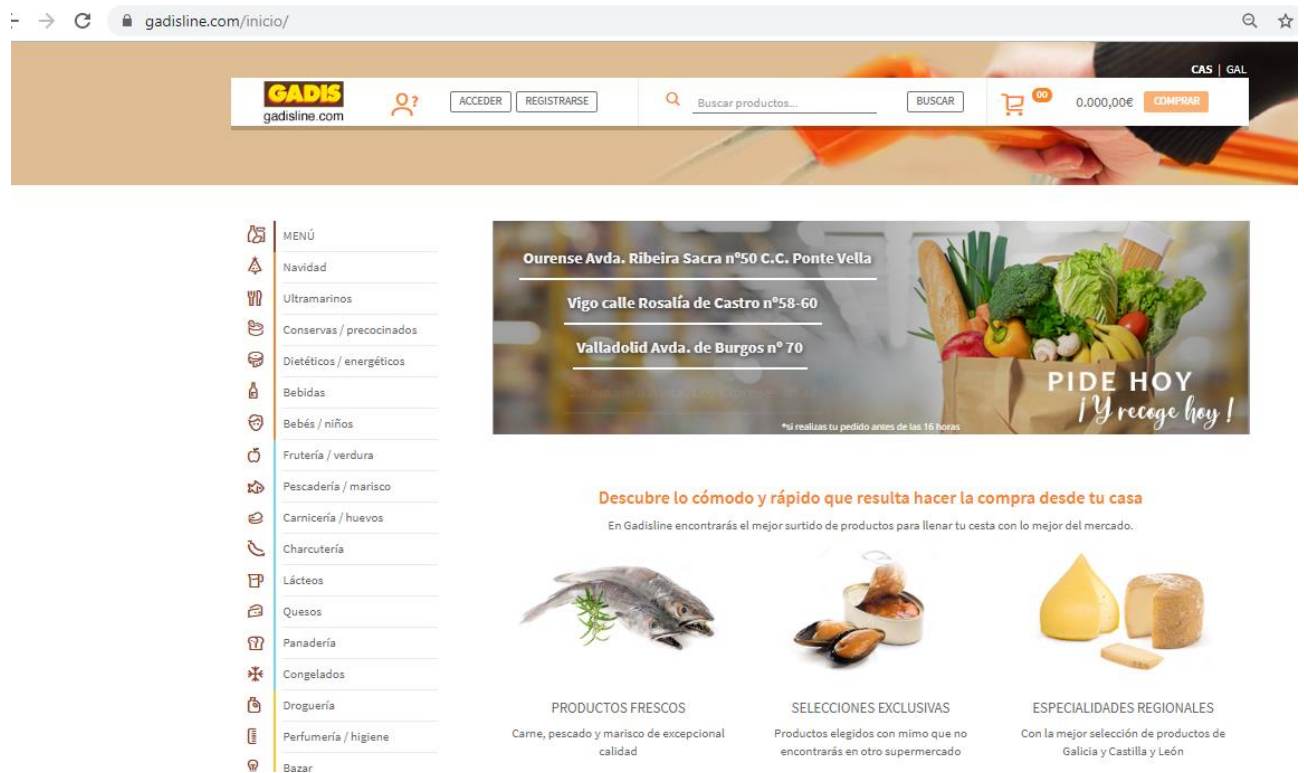
```
#Un aspecto importante es poder ver que user-agent estamos utilizando y ver si es adecuado el cambio
#Realmente al usar el chromedriver estamos simulando como que una persona está usando un navegador Chrome visitando páginas
agent = driver.execute_script("return navigator.userAgent")
print("=====")
print("El user-agent utilizado es:\n",agent,sep='')
print("=====")

=====
El user-agent utilizado es:
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.70 Safari/537.36
=====
```

Una vez solventado el inconveniente de la página inicial, llegamos a la página donde realmente podremos encontrar lo que buscamos: los productos.

## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

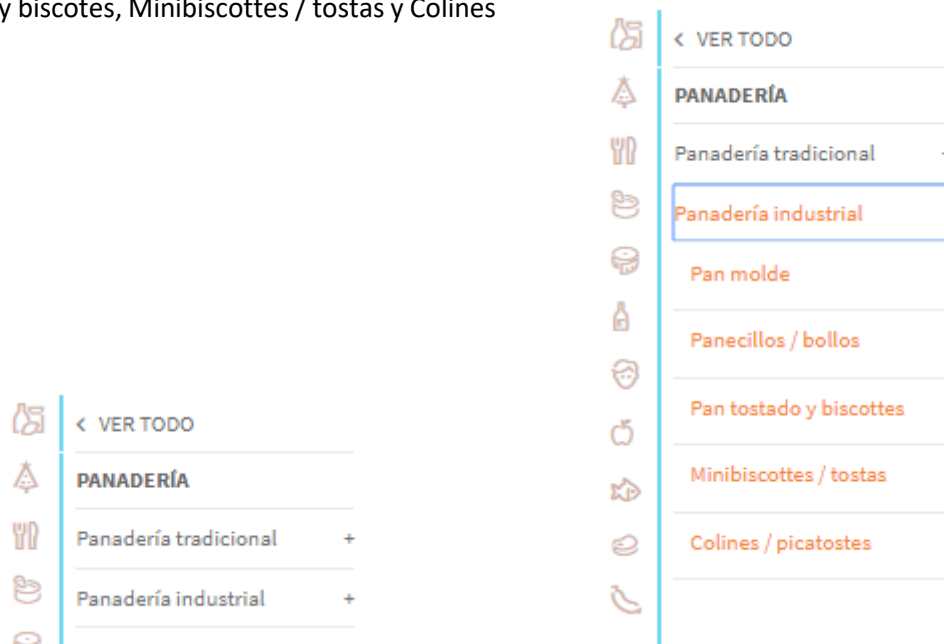
Javier Fernández y Rosa M. Suárez



De esta página lo que nos interesa realmente es poder recorrer ese menú que existe a la izquierda, donde se muestran a través de categorías de productos.

El menú realmente consta de 3 niveles. Por ejemplo, si seleccionamos en el Nivel 1 "Panadería", entonces aparece un nuevo menú que nos permite seleccionar entre "Panadería tradicional" y "Panadería industrial".

Y si elegimos "Panadería industrial" (ya estamos en un Nivel2), nos aparecen una serie de ítems elegibles (Pan de molde, Panecillos / bollos, Pan tostado y biscotes, Minibiscotes / tostas y Colines / picatostes)



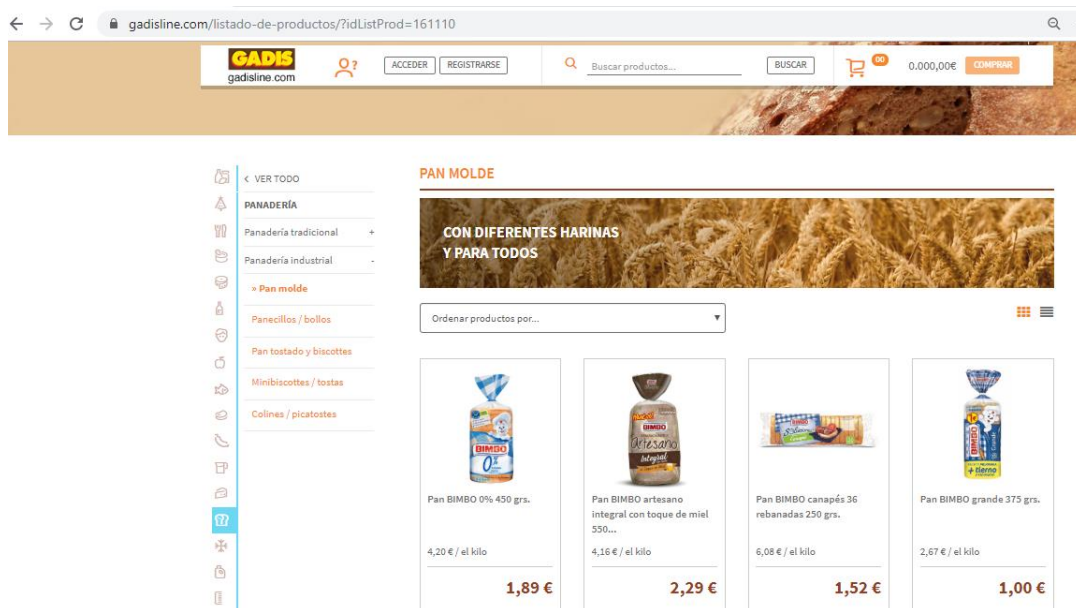
## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

A su vez esos ítems son elegibles (Nivel3) y al seleccionar cualquiera de ellos nos lleva directamente a la página donde se encuentran los productos de ese nivel3.

Es decir, la estructura de los artículos lo tienen repartido en 3 niveles, y cuando se llega al nivel 3 es cuando realmente podemos ver los artículos.

Si por ejemplo elegimos “Pan de molde” (recordemos Nivel1=“Panadería”, Nivel2=“Panadería industrial” y Nivel3=“Pan de molde” nos lleva a una nueva página web (concretamente <https://www.gadiline.com/listado-de-productos/?idListProd=161110>) donde se encuentran los productos que están en esa categoría concreta.



Lo que nos interesa realmente es poder tener disponible todo el menú de categorías, porque además es algo dinámico. Aparecen y desaparecen categorías y nos interesa siempre poder acceder a todas las que se encuentren en cada momento. Por ejemplo, durante la realización de esta práctica apareció la categoría a Nivel1 = “Navidad” y seguramente esa categoría en pocos meses desaparecerá.

Para obtener las categorías completas, lo que hicimos fue acceder en la página al elemento “Menú” y ver como estaba estructurado, a ver si había alguna forma de poder obtener todo el árbol de categorías sin tener que estar simulando pulsaciones constantes de ratón, lo que haría además mucho más lento la obtención de los datos que queremos.

Inspeccionando los elementos de la página hemos encontrado que el “div” con “class” = 'acc-menu col-xs-10 col-sm-10 labels' que es el que contiene todo el bloque Nivel1 de los diferentes productos. No hemos cogido el nivel superior porque esta página tiene una versión para móvil por lo visto, y entonces dependiendo de uno u otro, tiene 2 menús completos. Por tanto vamos a localizar y recorrer ese “div” para poder obtener todos los niveles del árbol de categorías.

## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

```
▼<div class="menu_category_container">
  ▶<div class="col-xs-2 col-sm-2 icons">...</div>
  ▼<div aria-activedescendant="true" class="acc-menu col-xs-10 col-sm-10 labels"> == $0
    ▶<div class="row-10 label-row" data-category-color="#f7862a">...
    </div>
    ▶<div class="row-11 label-row" data-category-color="#f7862a">...
    </div>
    ▶<div class="row-12 label-row" data-category-color="#f7862a">...
    </div>
    ▶<div class="row-22 label-row" data-category-color="#f7862a">...
    </div>
    ▶<div class="row-21 label-row" data-category-color="#f7862a">...
    </div>
    ▶<div class="row-23 label-row" data-category-color="#f7862a">...
    </div>
```

Adjuntamos parte del código:

```
#Accedemos al menú, a través del elemento padre:
# - Podemos hacerlo con: driver.find_element_by_class_name("acc-menu.col-xs-10.col-sm-10.labels")
# - Podemos hacerlo con: driver.find_element_by_xpath("//div[@class='acc-menu col-xs-10 col-sm-10 labels']")

#parentElement = driver.find_element_by_class_name("acc-menu.col-xs-10.col-sm-10.labels")
#parentElement = driver.find_element_by_class_name("menu_category_container")
elementoPadreMenu = driver.find_element_by_xpath("//div[@class='acc-menu col-xs-10 col-sm-10 labels']")

#Una vez localizado el elemento padre del menú, buscamos los tag "div" que cuelgan de él
elementList_div = elementoPadreMenu.find_elements_by_tag_name("div")

#Recorremos los elementos "div" que cuelgan del MenúPadre
for elemento_div in elementList_div:
    clase_div = elemento_div.get_attribute('class')

    #Nos interesan los elementos div cuya clase "col-xs-12 category-wrapper"
    if (clase_div == 'col-xs-12 category-wrapper'):
        #Si solo quiero un nivel
        #elementList_a = elemento_div.find_element_by_tag_name("a")
        #print (elementList_a.get_attribute('text'))

        #Una vez encontrado cada elemento "div" de clase = "col-xs-12 category-wrapper" buscamos
        #los tag <a> ya que en ellos se encuentran los diferentes niveles (del 1 al 3)

        elementList_a = elemento_div.find_elements_by_tag_name("a")
        for elemento_a in elementList_a:
            #El nivel se obtiene leyendo el atributo "data-ga_action" de los tags encontrados
            nivel = elemento_a.get_attribute('data-ga_action')
            url link = ''
```



## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

```
url_link = ''
if (nivel == 'nivel 1'):
    print('(L1) +--->', end='')
    texto = elemento_a.get_attribute('text')
    txtN1 = texto
    N1 += 1
elif (nivel == 'nivel 2'):
    print('  (L2) +--->', end='')
    texto = elemento_a.get_attribute('text')
    txtN2 = texto
    N2 += 1
elif (nivel == 'nivel 3'):
    # En el nivel 3 aparte del texto que obtenemos en los otros niveles, tenemos un link a esa categoría
    print('    (L3) +--->', end='')
    texto = elemento_a.get_attribute('aria-label')
    txtN3 = texto
    url_link = elemento_a.get_attribute('href')
    texto = texto + ' [' + url_link + ' ]'
    N3 += 1
    df.loc[elementos_tabla] = N1, N2, N3, txtN1, txtN2, txtN3, url_link, 0, \
        datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    elementos_tabla += 1
else:
    print('-----')
    print(texto)

# Marcamos el tiempo de fin
tFin = datetime.now()

texto = 'Hemos invertido {:.2f} segundos en obtener {:d} categorías (de 3 niveles)' \
    .format((tFin-tInicio).total_seconds(), df.shape[0])
print(texto)
```

Una vez que localizamos el div que buscábamos donde estaba todo el menú, vamos recorriendo los elementos de ese div y nos interesan los elementos linkables, es decir cuyo tag es <a ...> y además con el atributo “data-ga\_action” ya que ese atributo nos va a indicar si es “nivel 1”, “nivel 2” o “nivel 3”. Además, en ese tag tenemos el nombre de la categoría (en el nivel que corresponda).

Y ya solamente cuando nos encontramos en un elemento de nivel 3, es cuando además del nombre de la categoría, podemos obtener la página web donde se encuentran los artículos de esa categoría de nivel 3.

Por ejemplo, para el nivel 3 del Pan de molde que comentábamos anteriormente:

```
"nivel 2" class="plus-1611 plus-a open" onclick=
"displayLevel2('1611');"Panadería industrial</a>
<ul aria-labelledby="level2_1" role="menu" aria-hidden=
"false" aria-expanded="true" data-id="lv2-1611" class=
"lv2-1611 level2list" data-category-color="#6CD8F3">
  <li role="presentation">
    <a id="child-161110" role="menuitem" aria-label="Pan
molde" data-ga_action="nivel 3" class="child-161110
col-xs-12 category_menu_link current" href="https://
www.gadisline.com/listado-de-productos/?
idlistProd=161110&nofilters"> == $0
    <span class="category_menu_name">Pan molde</span>
  </a>
</li>
<li role="presentation">...</li>
<li role="presentation">...</li>
<li role="presentation">...</li>
```

## **PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.**

*Javier Fernández y Rosa M. Suárez*

De esta forma somos capaces de obtener todo el árbol de categorías (nivel 1 – 2 – 3) y además obtener para el último nivel, la página web asociada.

Si todos esos datos los almacenamos en un pandas DataFrame luego podremos hacer búsquedas por niveles de categorías si en algún momento queremos consultar precios de productos de alguna categoría concreta.

### **Accediendo a las páginas con los productos.**

Una vez que ya tenemos disponible todo el menú completo (que son aproximadamente unas 500 categorías) ahora ya podemos acceder a las páginas donde realmente están esos productos porque tenemos la dirección de la página web de cada uno de los niveles 3.

Una de las dificultades añadidas que nos hemos encontrado es que cuando se carga una página de productos, realmente la página solamente carga hasta 24 productos. El problema es que si hay más productos, esos solamente se cargan si alguien hace un scroll down de la página. Es decir “se necesita un usuario” que sea el que está “visitando” la página y al ir bajando sobre ella, los artículos se van cargando dinámicamente.

Entonces aquí volvemos a tener que hacer uso de alguna herramienta como la que hemos escogido, Selenium webdriver, ya que lo que vamos a hacer realmente cuando carguemos una página de artículos es simular que “alguien” está haciendo scroll-down de forma continua hasta el final (ese final se obtiene cuando la página deja de crecer, es decir cada vez que hacemos scroll-down, si aparecen nuevos artículos, la página crece al alto, de hecho se puede comprobar muy bien con la barra del scroll como cambia). Entonces la página va recargado más y más artículos hasta que realmente no hay más que mostrar, y entonces deja de crecer.

Lo hemos realizado implementando 2 formas posibles, ejecutando “`window.scrollTo(0, document.body.scrollHeight);`” y también le hemos añadido el envío de la tecla END (`driver.find_element_by_tag_name('body').send_keys(Keys.END)`).

Se comprueba el si la página cambia de tamaño a lo alto, y si no cambia, entonces ya no hay que seguir haciendo más scroll. Entre scroll y scroll hacemos una breve pausa, para ir dejando cargar los nuevos artículos.



## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

```
#Para cada URL cargada, vamos a hacer scroll hasta abajo de todo, para que se carguen todos los productos
tiempo_pausa_scroll = 1.5

# Get scroll height
ultimo_height = driver.execute_script("return document.body.scrollHeight")

while True:
    # Hacemos scroll hacia abajo
    driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")

    # Esperamos para que se cargue la página
    time.sleep(tiempo_pausa_scroll)

    scroll = driver.find_element_by_tag_name('body').send_keys(Keys.END)

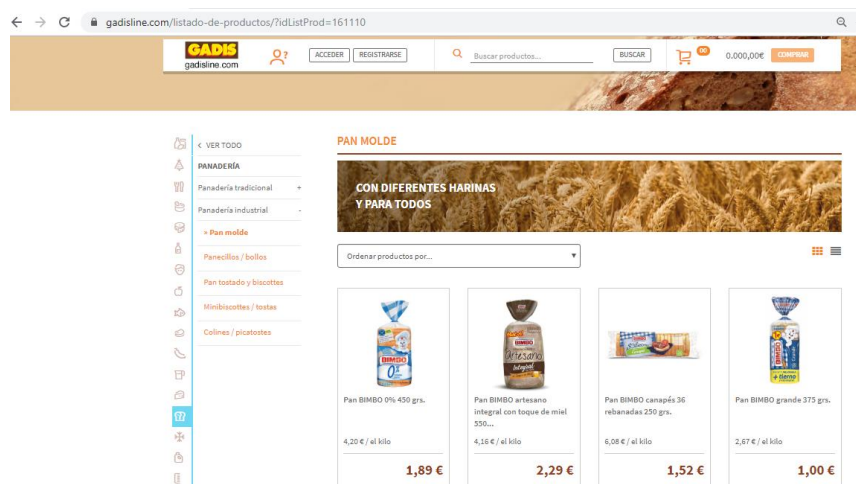
    # Esperamos para que se cargue la página
    time.sleep(tiempo_pausa_scroll)

    # Volvemos a calcular la altura de la página después de haber ejecutado el scroll
    # Si ha cambiado, continuamos, sino hemos acabado de hacer scroll
    nuevo_height = driver.execute_script("return document.body.scrollHeight")
    if nuevo_height == ultimo_height:
        break
    ultimo_height = nuevo_height
```

Una página donde se puede ver con mucha claridad este efecto es la página donde están las verduras (<https://www.gadisline.com/listado-de-productos/?idListProd=191110>) ya que en esa hay más de 150 artículos y por tanto necesitaba varios scroll-down para cargarlos todos.

Una vez que ya estamos en la página de los artículos y se supone que con todos los artículos de ese nivel 3 cargados (porque hemos hecho scroll-down hasta que no vengan más artículos). Nos queda ir a buscar realmente los datos propios de cada artículo que sale en la página.

Centrados por ejemplo ya en el nivel “Pan Molde”:



Queremos recuperar todos los artículos de esa categoría (nivel3), y con el siguiente nivel de detalle:

- El nombre
- El id (con el que luego vinculamos todos los datos)
- El precio

*Javier Fernández y Rosa M. Suárez*

- Cada producto está en un “div” de class=’product\_container’



Y para cada uno de los elementos que obtenemos:

Localizamos el elemento `<img class='img-responsive product_img opacity_hover'>` con el que obtenemos la url de la imagen (atributo `'src'`)

Del elemento `<a class='gtmProductClick'>` nos quedamos con el ID, con el nombre del producto y con la URL propia del producto

Del elemento `<p class='product_price'>` nos quedamos con el precio de venta.

## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

Del elemento `<p class='producto_unity_price'>` nos quedamos con el precio por unidad de medida.

Con todos estos datos recorriendo los diferentes niveles podremos construir un catálogo de productos actualizado (artículo, precio, precio por unidad de medida) y además podremos si queremos empezar a construirnos un histórico de precios, de forma que luego podamos hacer un análisis de la evolución de los precios de los artículos. Si vamos guardando las lecturas por fecha, tendremos ese histórico.

Dentro de los datos obtenidos en la lista de productos del nivel 1 dijimos que estaba la URL propia del producto.

The screenshot shows the product page for 'Pan BIMBO 0% 450 grs.' on the website gadisline.com. The page includes a navigation bar with the Gadisline logo, user account options (ACCEDER, REGISTRARSE), a search bar, and a shopping cart icon showing 0.000,00€. The product image is a loaf of bread in a blue and white bag. The price is 1,89 €, with a unit price of 4,20 € / el kilo. There is a quantity selector set to 01 and an 'AÑADIR' button. Below the product image is a text area for a note, with an example: 'Por ejemplo: caducidad superior a cuatro días'. At the bottom, there are two sections: 'ANÁLISIS NUTRICIONAL' and 'INGREDIENTES'. The nutritional analysis table lists values for 100g: Energy (251 kcal), Total fat (10,4 g), Monounsaturated fat (0,7 g), Polyunsaturated fat (1,5 g), Carbohydrates (45 g), and Sugars (3,5 g). The ingredients list includes wheat flour, water, yeast, gluten, vegetable oil, salt, preservatives (E-202, E-200, E-202), wine vinegar, emulsifiers (E-472e, E-481), and wheat treatment agent (E-300, E-341). It also mentions the presence of sesame seeds and natural sugars.

gadisline.com/producto/?productID=6082544

**GADIS**  
gadisline.com

ACCEDER REGISTRARSE

Buscar productos... BUSCAR

0.000,00€ COMPRAR

[VOLVER A PAN MOLDE](#)

Pan BIMBO 0% 450 grs.

**1,89 €**

4,20 € / el kilo

01

**AÑADIR**

Si quieres, puedes incluir una nota sobre este producto para tenerla en cuenta al servir tu pedido. ⓘ

Por ejemplo: caducidad superior a cuatro días

**ANÁLISIS NUTRICIONAL**

La información del nutriente se aplica cuando el producto está preparado

Valor medio en 100g
Energía: (251 kcal)
Grasas: (10,4 g)
Ácidos grasos saturados: (0,7 g)
Ácidos grasos monoinsaturados: (1,5 g)
Hidratos de carbono: (45 g)
Azúcares: (3,5 g)

**INGREDIENTES**

Harina de trigo, agua, levadura, gluten de trigo, aceite vegetal (girasol), sal, conservadores (E-202, E-200, E-202), vinagre de vino, emulgentes (E-472e, E-481), agente de tratamiento de la harina (E-300, E-341). Puede contener soja y/o semillas de sésamo. Contiene azúcares naturalmente presentes.

**CONSEJOS DE CONSERVACIÓN**

Esta página completamente estática la hemos tratado con BeautifulSoup.

En este momento la dificultad con la que nos hemos encontrado es con la falta de estructura en los datos. Realmente los datos están metidos en un elemento div, en los que todo el texto es continuo, sin separar los diferentes valores.

## ANÁLISIS NUTRICIONAL

La información del nutriente se aplica cuando el producto está preparado

Valor medio en 100g

Energía: (251 kcal)

Grasas: (2,7 g)

Ácidos grasos saturados: (0,4 g)

Ácidos grasos monoinsaturados: (0,7 g)

Ácidos grasos poliinsaturados: (1,5 g)

Hidratos de carbono: (45 g)

Azúcares: (3,5 g)

Almidón: (41 g)

Fibra alimentaria: (3 g)

Proteínas: (11 g)

Sal: (1,1 g)

### Nombre Legal Producto

PAN DE MOLDE

### Cantidad neta del alimento:

450 g

### Nombre Operador de la empresa alimentaria o

#### importador:

BIMBO DONUTS IBERIA, S.A.U.

**Dirección del operador o importador:** Calle Cigoitia 1,

Polígono Industrial Las Mercedes 28022 Madrid (España)

## INGREDIENTES

Harina de **trigo**, agua, levadura, gluten de **trigo**, aceite vegetal (girasol), sal, conservadores (E-282, E-200, E-202), vinagre de vino, emulgentes (E-472e, E-481), agente de tratamiento de la harina (E-300, E-341). Puede contener **soja** y/o semillas de **sésamo**. Contiene azúcares naturalmente presentes.

## CONSEJOS DE CONSERVACIÓN

Mantener en lugar fresco y seco.

En esta sección lo que nos interesaba obtener es lo siguiente:

- Análisis nutricional
- Ingredientes
- Cantidad neta del alimento

Pero no fue tan sencillo:

```
▼<div role="contentinfo" tabindex="0" class="col-sm-4">
  <h2>ANÁLISIS NUTRICIONAL</h2> == $0
  ▶<p itemprop="description">...</p>
  <!--?php/*Campo libre 1*/?-->
  <strong>Nombre Legal Producto</strong>
  <br>
  "
  PAN DE MOLDE"
  <br>
  <br>
  <strong>Cantidad neta del alimento:</strong>
  <br>
  "450 g"
  <br>
  <br>
  <strong>Nombre Operador de la empresa alimentaria o importador:</strong>
  <br>
```

## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

Los textos están como una secuencia de textos simplemente separados por saltos de línea <br> sin ningún identificativo propio.

Lo primero que hemos hecho es localizar los “div” con el atributo “role”=‘contentinfo’  
Luego dentro de ese los elementos, solamente aquellos cuya “class”=‘col-sm-4’  
Y para obtener la “Cantidad neta” recorreremos todos los elementos <br> intentando localizar el texto “Cantidad neta”, y el siguiente elemento es el valor que estamos intentando encontrar.

```
driver.get('https://www.gadisline.com/producto/?productID=6082544')
soup=BeautifulSoup(driver.page_source,"lxml")
#mydivs = soup.findAll("div", {"class": ["col-sm-4"]})
mydivs = soup.findAll("div", {"role": "contentinfo"})
for elementoDiv in mydivs:
    clase=''.join(elementoDiv.get('class'))

    if (clase=='col-sm-4'):
        html_content = elementoDiv.prettify()
        html_content = html_content.replace("\r", "")
        html_content = html_content.replace("\n", "")
        #print(elementoDiv.get_text())
        sig_tag_ok = False
        for br in elementoDiv.findAll('br'):
            next_s = br.nextSibling
            texto = str(next_s).strip()
            if sig_tag_ok:
                print("La cantidad buscada es:", texto)
                break
            pos = texto.find("Cantidad neta")
            #print(pos, next_s, "::::", texto)
            if (pos != -1):
                sig_tag_ok = True
```

Para la parte de Ingredientes y Análisis nutricional buscamos dentro del <div> que estamos tratando los elementos H2, y buscamos si corresponden a los textos que esperamos “ANÁLISIS NUTRICIONAL” e “INGREDIENTES”

```
bloque_nutricional = False
bloque_ingredientes = False
for h2 in elementoDiv.findAll('h2'):
    txth2 = h2.text
    if (txth2.find('ANÁLISIS NUTRICIONAL') != -1):
        bloque_nutricional = True
    if (txth2.find('INGREDIENTES') != -1):
        bloque_ingredientes = True
```

Y luego ya específicamente para cada uno de esos bloques hacemos un tratamiento de texto específico, ya que como comentábamos antes, la información no está muy estructurada, por tanto lo que hicimos fue obtener el texto, y luego ir buscando diferentes cadenas, para quedarnos con lo que realmente nos interesaba.

## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

```
if (bloque_nutricional):

    texto=elementoDiv.get_text()
    #print(texto)

    #Borramos el texto ANALISIS NUTRICIONAL
    texto=texto.replace("ANÁLISIS NUTRICIONAL","")
    #Localizamos los cambios de parrafo
    pos=texto.find("\n")

    texto_simp=texto[:pos]
    #print(texto_simp)
    #Ahora una vez que hemos detectado el corte, quitamos todos los saltos de línea.
    texto_simp = texto_simp.replace("\r",". ")
    texto_simp = texto_simp.replace("\n",". ")

    #Localizamos "Energía" que es donde comienza la info nutricional
    pos_energia=texto_simp.find("Energía")

    #Localizamos "tamaño" o "cantidad" puesto que viene con estas dos nomenclaturas posibles
    pos_tamanno=texto_simp.find("Tamaño")
    if (pos_tamanno!=-1):
        pos_tamanno=texto_simp.find("Cantidad")

    if (pos_tamanno != -1):
        #Hemos localizado el texto "Tamaño"
        if (pos_energia!=-1):
            #Para los casos en que localice "Tamaño" y "Energía"(habitualmente primero info de tamaño y despues energía)
            #Imprimimos como tamaño en texto entre la etiqueta "Tamaño" y "Energía"
            #Imprimimos como análisis nutricional la info que comienza en etiqueta "Energía" hasta el final
            #print ("->Tamaño: ", texto_simp[pos_tamanno:pos_energia])
            #print ("->Tamaño: ", texto_simp[pos_tamanno:pos_energia])
            txt_nutricional = texto_simp[pos_tamanno:pos_energia].strip()
            if (len(txt_nutricional) > 0):
                #Comprobar si acaba con punto
                if (txt_nutricional[-1:] != '.'):
                    txt_nutricional = txt_nutricional + '.'
            txt_nutricional = txt_nutricional + ' ' + texto_simp[pos_energia:pos]
            #print ("->Análisis nutricional: ", texto_simp[pos_tamanno:pos_energia] + '.' + texto_simp[pos_energia:pos])
            print(">Análisis nutricional:", txt_nutricional)
        else:
            #Si localizamos "Tamaño" pero no "Energía", la info de tamaño es la que comienza en la etiqueta hasta el fin
            print ("->Análisis nutricional: ", texto_simp[pos_tamanno:pos])

    else:
        #No hemos localizado el texto "Tamaño"
        if (pos_energia!=-1):
            #Si no localizamos "Tamaño" pero sí "Energía": La info nutricional comenzara en etiq "Energía" hasta el fin
            print ("->Análisis nutricional: ", texto_simp[pos_energia:pos])
        else:
            #Nos quedamos con todo el párrafo
            print (texto_simp)

#Bloque de ingredientes
```

Y para los INGREDIENTES leemos todo o hasta que encontramos "CONSEJOS":

```
#Bloque de ingredientes
if (bloque_ingredientes):
    #Localizamos "Ingredientes" y "consejos"
    texto=elementoDiv.get_text()
    #print(texto)
    pos_ingredientes=texto.find("INGREDIENTES")
    pos_consejos=texto.find("CONSEJOS")

    #Quitamos todos los saltos de línea.
    texto_simp = texto.replace("\r",". ")
    texto_simp = texto_simp.replace("\n",". ")

    if (pos_consejos!=-1):
        #Si existe etiqueta "Consejos", los ingredientes irán desde etiq "Ingredientes" hasta "Consejos"
        print ("->Ingredientes: ", texto_simp[pos_ingredientes:pos_consejos].replace('INGREDIENTES',''))
    else:
        #Si no existe etiqueta "consejos" los ingredientes irán hasta el final del textto.
        print ("->Ingredientes: ", texto_simp[pos_ingredientes:].replace('INGREDIENTES',''))
```

## **PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.**

*Javier Fernández y Rosa M. Suárez*

Si todo va bien, de una página de este estilo habremos recuperado:

- La cantidad neta del producto
- El análisis nutricional
- Los ingredientes.

Y estos datos son muy útiles para complementar los que ya teníamos a nivel de producto. Con lo cual, finalmente de cada producto hemos obtenido:

- Id (para relacionar la información)
- Nombre del producto
- Precio del producto
- Precio por unidad de medida
- URL de la imagen
- URL de características del producto
- Cantidad neta del producto
- Ingredientes
- Análisis nutricional.

## 4. Agradecimientos

Los datos se han obtenido de la web de compra on line del retailer alimentario Gadisa. Se trata de un líder en Galicia en su sector y que también ha comenzado a vender ya en Castilla León. Aproximadamente desde el año 2012 cuentan con la web actual.

## 5. Inspiración

La elección de este conjunto de datos viene dada por el origen del mundo del retail alimentario de los dos autores de la práctica (Javier Martínez y Rosa M. Suárez). La puesta en marcha de las webs de venta on line en este sector ha supuesto una mejora muy importante en la tarea de chequeo de precios en la competencia, pasando de ser un proceso meramente presencial con exigencia de personal dedicado a ello, a que pueda ser sustituido por un proceso automático que resuelve el problema en unas horas. Al scraping de precios que aporta nuestra solución habría que añadirle un paso más, que sería el enlace de los precios propios con los extraídos. Es decir, el enlace de productos, ¿de qué manera los relacionamos? Este es ya otro problema, de magnitud propia que merecería un nuevo proyecto.

Otras aplicaciones del scraping desarrollado podrían ser dar respuesta a preguntas del tipo: Productos más caros por tipología, en función de su precio/kilo, top de productos más calóricos, top de productos más insanos (por presencia de aceite de palma u otros contenidos desaconsejados)... etc.

## 6. Licencia

La licencia elegida en nuestro caso ha sido CC BY-SA 4.0 License, los motivos son los siguientes:

Para copiar y redistribuir nuestros datos se podrán hacer en cualquier medio o formato. Otra ventaja es que estos datos o material pueden tener un uso comercial a diferencia de otro tipo de licencias que esto no lo permitirían.

Otra característica de este tipo de licencias es que, si se modifica o se crea otro material a partir de nuestros datos, este tiene que estar guardado con la misma licencia de los datos originales.



## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

### 7. Código

El código puede encontrarse en el siguiente enlace a GitHub:

[https://github.com/javifermar/Retail\\_Scraping](https://github.com/javifermar/Retail_Scraping)

Indicaciones a la ejecución:

- Es preciso para la correcta ejecución de este proyecto, localizar el archivo chromedriver.exe que se puede encontrar en la carpeta Chromedriver de este proyecto en la ruta local c:\Chromedriver
- Verificar que se tienen instaladas las librerías indicadas anteriormente.
- Ejecutar python Retail\_Scraping.py

NOTA: Se recomienda inicialmente la ejecución sobre una categoría con un número moderado de elementos. Por ejemplo, Panadería (N1=13. N2=0. N3=0), o Dietéticos/energéticos-Dietéticos (N1=4 N2=24 N3=0)

```

Categorías que han cambiado de URL: 0
Total Páginas
N1 Nivel1
1 Navidad 10
2 Ultramarinos 101
3 Conservas / precocinados 30
4 Dietéticos / energéticos 5
5 Bebidas 74
6 Bebés / niños 14
7 Frutería / verdura 2
8 Pescadería / marisco 1
9 Carnicería / huevos 5
10 Charcutería 30
11 Lácteos 29
12 Quesos 26
13 Panadería 6
14 Congelados 22
15 Drogueria 56
16 Perfumería / higiene 71
17 Bazar 11
18 Mascotas 8
Selecciona una categoría M1. Escriba SALIR si lo desea: 4
Total Páginas
N2 Nivel2
24 Dietéticos 5
Selecciona una categoría N2. Puede poner 0 para TODAS. Escriba SALIR si lo desea: 24
Total Páginas
N3 Nivel3
142 Galletas,barritas y tortitas 1
143 Panificados 1
144 Mermeladas 1
145 Infusiones / líquidos dietéticos 1
146 Otros dietéticos 1
Selecciona una categoría N3. Puede poner 0 para TODAS. Escriba SALIR si lo desea: 0
=====
Productos N1Dietéticos / energéticos N2Dietéticos N3:
=====
Aún no existe un dataframe guardado con toda la información de los de productos.
Dietéticos / energéticos->Dietéticos->Galletas,barritas y tortitas->https://www.gadisline.com/listado-de-productos/?idListProd=221010
```

## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

## 8. Dataset

### CATEGORIAS

Nivel1	Nivel2	Nivel3	URL	NumArticulos	UltActualizacion
Navidad	Navidad	Turrón blando	https://www.gadisline.com/listado-de-productos/?idListProd=101005	0	05/11/2019 1:51
Navidad	Navidad	Turrón duro/tortas	https://www.gadisline.com/listado-de-productos/?idListProd=101011	0	05/11/2019 1:51
Navidad	Navidad	Turrón chocolate/trufados	https://www.gadisline.com/listado-de-productos/?idListProd=101012	0	05/11/2019 1:51
Navidad	Navidad	Trufas chocolate	https://www.gadisline.com/listado-de-productos/?idListProd=101013	0	05/11/2019 1:51
Navidad	Navidad	Turrón fruta/yema/guirlache	https://www.gadisline.com/listado-de-productos/?idListProd=101014	0	05/11/2019 1:51
Navidad	Navidad	Especialidades/mazapán	https://www.gadisline.com/listado-de-productos/?idListProd=101015	0	05/11/2019 1:51
Navidad	Navidad	Mantecados/polvor./hojald.	https://www.gadisline.com/listado-de-productos/?idListProd=101016	0	05/11/2019 1:51
Navidad	Navidad	Peladillas/piñones/almend.rellenas	https://www.gadisline.com/listado-de-productos/?idListProd=101017	0	05/11/2019 1:51
Navidad	Navidad	Panettones	https://www.gadisline.com/listado-de-productos/?idListProd=101010	0	05/11/2019 1:51

### PRODUCTOS

Id	Nombre	PVP	PVP_Unidad_Medida	Cantidad_Net.	URL_Producto	URL_Imagen
8E+06	Acelgas manojó 1º peso mínimo 500 grs. unidad	1,12	2,24 € / el kilo		https://www.gadisline.com/producto/?productID=7801010	https://www.gadisline.com/resources/images/7801010_F_0
8E+06	Espinacas manojó categoría 1ª peso mínimo 300...	1,35	4,50 € / el kilo		https://www.gadisline.com/producto/?productID=7841010	https://www.gadisline.com/resources/images/7841010_F_0
8E+06	Nabizas manojó 1º (peso mínimo 800 grs.) unidad	1,69	2,11 € / el kilo		https://www.gadisline.com/producto/?productID=7855500	https://www.gadisline.com/resources/images/7855500_F_0
8E+06	Verdura rizada manojó categoría 1ª (peso mínimo...	1,65	3,30 € / el kilo		https://www.gadisline.com/producto/?productID=7891010	https://www.gadisline.com/resources/images/7891010_F_0
8E+06	Xenos repollo sin acopillar manojó categoría 1ª...	1,5	3,75 € / el kilo		https://www.gadisline.com/producto/?productID=7887910	https://www.gadisline.com/resources/images/7887910_F_0
8E+06	Ajo blanco elefante 1º calibre 68 (1 cabeza peso...	0,39	4,33 € / el kilo		https://www.gadisline.com/producto/?productID=7803193	https://www.gadisline.com/resources/images/7803193_F_0
1E+07	Ajo blanco extra calibre 55/62 mm. butti 4 unidades...	1,2	4,80 € / el kilo		https://www.gadisline.com/producto/?productID=9856060	https://www.gadisline.com/resources/images/9856060_F_0
8E+06	Ajo blanco yumbo calibre 62/68 mm. kilo	3,49	3,49 €		https://www.gadisline.com/producto/?productID=7803129	https://www.gadisline.com/resources/images/7803129_F_0
8E+06	Ajo ecológico calibre 0/62 malla 4 unidades...	1,25	6,25 € / el kilo		https://www.gadisline.com/producto/?productID=7803100	https://www.gadisline.com/resources/images/7803100_F_0
8E+06	Ajo morado calibre 45/55 mm. butti 5 unidades 200...	0,9	4,50 € / el kilo		https://www.gadisline.com/producto/?productID=7803212	https://www.gadisline.com/resources/images/7803212_F_0
8E+06	Ajo morado diente gordo calibre 55mm+ malla 500...	2,19	4,38 € / el kilo		https://www.gadisline.com/producto/?productID=7803189	https://www.gadisline.com/resources/images/7803189_F_0
8E+06	Ajo morado super calibre 50/60 mm. kilo	4,49	4,49 €		https://www.gadisline.com/producto/?productID=7803229	https://www.gadisline.com/resources/images/7803229_F_0
8E+06	Ajo morado Yumbo C-62+ (1 cabeza peso mínimo 75...	0,39	5,20 € / el kilo		https://www.gadisline.com/producto/?productID=7803191	https://www.gadisline.com/resources/images/7803191_F_0

### HISTORICO PRECIOS

Id	DiaCaptura	PVP	PVP_Unidad_Medida	UltActualizacion
5325068	30/10/2019	2,69	26,90 € / el kilo	30/10/2019 1:50
5325086	29/10/2019	3,2	4,92 € / el kilo	30/10/2019 1:48
5325093	30/10/2019	3,99	7,98 € / el kilo	30/10/2019 1:50
5325091	30/10/2019	3,09	7,73 € / el kilo	30/10/2019 1:50
5325094	30/10/2019	2,85	21,92 € / el kilo	30/10/2019 1:50
5325090	30/10/2019	2,85	14,25 € / el kilo	30/10/2019 1:50
5325082	30/10/2019	2,79	13,95 € / el kilo	30/10/2019 1:50
5325069	30/10/2019	2,4	16,00 € / el kilo	30/10/2019 1:50
5325072	30/10/2019	2,2	22,00 € / el kilo	30/10/2019 1:50

### INFO NUTRICIONAL

Id	AnalisisNutricional	UltActualizacion
6E+06	Energía: (251 kcal)Grasas: (2,7 g)Ácidos grasos saturados: (0,4 g)Ácidos grasos monoinsaturados: (0,3 g)Ácidos grasos poliinsaturados: (0,0 g)Hidratos de carbono: (52 g)Azúcares: (0,0 g)Fibra: (0,0 g)Proteínas: (2,7 g)Sal: (0,0 g)	09/11/2019 2:01
6E+06	Energía: (259 kcal)Grasas: (2,7 g)Ácidos grasos saturados: (0,5 g)Ácidos grasos monoinsaturados: (0,2 g)Ácidos grasos poliinsaturados: (0,0 g)Hidratos de carbono: (45 g)Azúcares: (0,0 g)Fibra: (0,0 g)Proteínas: (2,7 g)Sal: (0,0 g)	09/11/2019 2:01
6E+06	Energía: (299 kcal)Grasas: (6 g)Ácidos grasos saturados: (0,8 g)Hidratos de carbono: (52 g)Azúcares: (0,0 g)Fibra: (0,0 g)Proteínas: (2,7 g)Sal: (0,0 g)	09/11/2019 2:01
6E+06	Energía: (252 kcal)Grasas: (2,7 g)Ácidos grasos saturados: (0,5 g)Ácidos grasos monoinsaturados: (0,2 g)Ácidos grasos poliinsaturados: (0,0 g)Hidratos de carbono: (45 g)Azúcares: (0,0 g)Fibra: (0,0 g)Proteínas: (2,7 g)Sal: (0,0 g)	09/11/2019 2:01
6E+06	Energía: (262 kcal)Grasas: (2,2 g)Ácidos grasos saturados: (0,4 g)Ácidos grasos monoinsaturados: (0,2 g)Ácidos grasos poliinsaturados: (0,0 g)Hidratos de carbono: (45 g)Azúcares: (0,0 g)Fibra: (0,0 g)Proteínas: (2,7 g)Sal: (0,0 g)	09/11/2019 2:01
6E+06	Tamaño de referencia para 100g. Energía: (259 kcal)Grasas: (3 g)Ácidos grasos saturados: (0,7 g)Ácidos grasos monoinsaturados: (0,2 g)Ácidos grasos poliinsaturados: (0,0 g)Hidratos de carbono: (45 g)Azúcares: (0,0 g)Fibra: (0,0 g)Proteínas: (2,7 g)Sal: (0,0 g)	09/11/2019 2:01
6E+06	Energía: (239 kcal)Grasas: (3,4 g)Ácidos grasos saturados: (0,5 g)Hidratos de carbono: (45 g)Azúcares: (0,0 g)Fibra: (0,0 g)Proteínas: (2,7 g)Sal: (0,0 g)	09/11/2019 2:01
6E+06	Energía: (263 kcal)Grasas: (2,2 g)Ácidos grasos saturados: (0,6 g)Ácidos grasos monoinsaturados: (0,2 g)Ácidos grasos poliinsaturados: (0,0 g)Hidratos de carbono: (45 g)Azúcares: (0,0 g)Fibra: (0,0 g)Proteínas: (2,7 g)Sal: (0,0 g)	09/11/2019 2:01
6E+06	Tamaño de referencia para 100g. Energía: (256 kcal)Grasas: (4,4 g)Ácidos grasos saturados: (1 g)Ácidos grasos monoinsaturados: (0,3 g)Ácidos grasos poliinsaturados: (0,0 g)Hidratos de carbono: (45 g)Azúcares: (0,0 g)Fibra: (0,0 g)Proteínas: (2,7 g)Sal: (0,0 g)	09/11/2019 2:02
6E+06	Energía: (262 kcal)Grasas: (3,1 g)Ácidos grasos saturados: (0,5 g)Hidratos de carbono: (51 g)Azúcares: (0,0 g)Fibra: (0,0 g)Proteínas: (2,7 g)Sal: (0,0 g)	09/11/2019 2:02
6E+06	Energía: (252 kcal)Grasas: (2,8 g)Ácidos grasos saturados: (0,5 g)Ácidos grasos monoinsaturados: (0,2 g)Ácidos grasos poliinsaturados: (0,0 g)Hidratos de carbono: (45 g)Azúcares: (0,0 g)Fibra: (0,0 g)Proteínas: (2,7 g)Sal: (0,0 g)	09/11/2019 2:02
6E+06	Energía: (248 kcal)Grasas: (2,9 g)Ácidos grasos saturados: (0,7 g)Ácidos grasos monoinsaturados: (0,2 g)Ácidos grasos poliinsaturados: (0,0 g)Hidratos de carbono: (45 g)Azúcares: (0,0 g)Fibra: (0,0 g)Proteínas: (2,7 g)Sal: (0,0 g)	09/11/2019 2:02

### INGREDIENTES

## PRA1: Web Scraping. Tipología y Ciclo de vida de los datos.

Javier Fernández y Rosa M. Suárez

Id	Ingredientes	UltActualizacion
6E+06	Harina de trigo, agua, levadura, gluten de trigo, aceite vegetal (girasol), sal, conservadores (E-282,	09/11/2019 2:01
6E+06	Harina integral de trigo, agua, harina de trigo, levadura, miel (2,7%), harina integral de trigo espelti	09/11/2019 2:01
6E+06	Harina de trigo, agua, aceite vegetal (colza), azúcar, aroma, levadura, sal, gluten de trigo, emulgen	09/11/2019 2:01
6E+06	Harina de trigo, agua, levadura, aceite vegetal (girasol), sal, conservadores (E 282, E 200), emulger	09/11/2019 2:01
6E+06	Harina de trigo, agua, levadura, aceite vegetal (oliva refinado 1.3%), gluten de trigo, sal, azúcar, m	09/11/2019 2:01
6E+06	Harina de trigo, agua, levadura, aceite de girasol (1%), azúcar, fibra de avena, sal, emulgentes (E-4	09/11/2019 2:01
6E+06	Almidon de maíz, agua, masa madre 14% (harina de arroz, agua), almidon de arroz, jarabe de arro	09/11/2019 2:01
6E+06	Harina de TRIGO, agua, harina de CENTENO, masa madre (3,2%)(contiene TRIGO), levadura, aceit	09/11/2019 2:01
6E+06	Harina de trigo, agua, aceite de oliva virgen extra (2%), azúcar, levadura, sal, aceite de girasol, fibr	09/11/2019 2:02
6E+06	Almidón de maíz, agua, masa madre 14% (harina de arroz, agua), harina de alforfón 8%, harina de	09/11/2019 2:02
6E+06	Harina de trigo, agua, levadura, aceite de oliva refinado (1.1%), sal, masa madre 1% (harina de ma	09/11/2019 2:02
6E+06	Harina de trigo, agua, levadura, aceite vegetal (girasol), azúcar, sal, gluten de trigo, almidón de ma	09/11/2019 2:02
6E+06	Harina de trigo, agua, azúcar, levadura, sal, aceite vegetal (girasol), gluten de trigo, emulgentes (E-	09/11/2019 2:02
6E+06	Harina de trigo, agua, levadura, aceite vegetal (girasol), sal, conservadores (E 282, E 200), emulger	09/11/2019 2:02

Adjuntamos un archivo comprimido donde encontrar los 5 datasets de salida. Para su correcta visualización en Excel recomendamos importar en formato UTF-8.



datasets.rar