TRABAJO FINAL DE GRADO

TÍTULO: Analizando a la competencia

AUTOR: Joel Sánchez López

TITULACIÓN: Grado en Ingeniería Informática

PONENT: Neus Català Roig

DEPARTAMENTO: Ciencias de la computación

FECHA: 31 de enero de 2018

TÍTULO:

ANALIZANDO A LA COMPETENCIA

AUTOR: Joel Sánchez López

TITULACIÓN: Grado en Ingeniería Informática

ESPECIALIDAD: BBDD

PLAN : 2010

PONENT: Neus Català Roig

DEPARTAMENTO: Ciencias de la computación

NOTA TFG

TRIBUNAL

PRESIDENTA	VOCAL	SECRETARIO
HERNANDEZ GOMEZ, M. ANGELES	MASSIP BRUIN, JAVIER	ESTEVE CUSINE, JORDI

FECHA DE LECTURA:

07 de febrero de 2018

RESUMEN

El proyecto presentado a continuación tiene como objetivo la mejora del plan comercial de la empresa Venca, ayudando así a la toma de decisiones con la finalidad de mejorar la estrategia de mercado y conseguir llegar a los objetivos marcados.

Para ello crearemos un nuevo sistema de análisis de la competencia en el que incluiremos nuevas bases de datos, diversos informes y el programario necesario para rellenarlos.

Las herramientas principales en este proyecto serán Python (utilizando el entorno de desarrollo Spyder de Anaconda), SQL Server Management y Excel (en el cual también se incluye programación en VBA para macros). Más secundarias serán Filezilla y Pentaho, las cuales utilizaremos para hacer llegar los informes de manera automática a cualquier ordenador que los necesite.

PALABRAS CLAVE

Análisis – Estudio – Competencia – Programación –

Script – Estrategia de mercado –

Business Intelligence – Generación automático de informes

- Este proyecto tiene en cuenta aspectos medioambientales: ■ Sí ■ No

Analizando a la competencia

"Torture the data, and it will confess to anything."

Ronald H. Coase

Índice de contenido

1. INTRODUCCIÓN

1.1 Motivaciones

Actualmente nos encontramos en un mundo casi digital por completo en el cual, sin salir de casa y con tan solo dos clicks, puedes comprar ropa, comida o cualquier necesidad que tengas con apenas disponer de un teléfono móvil, Tablet u ordenador. Todas las empresas se han tenido que adaptar a este mundo y aún hoy lo siguen haciendo, ya que no para de cambiar y evolucionar. Es por ello por lo que cada vez necesitamos más información: para poder realizar estrategias empresariales que nos permitan llegar a los objetivos.

Hace ya poco más de dos años entré a formar parte de la empresa Venca como becario en el equipo de Business Intelligence. Mi objetivo era desarrollar una estructura que incluyera programas, bases de datos e informes para ayudar a la empresa a analizar a su competencia. En el equipo nos dedicamos a esto: a analizar datos, datos y más datos; toda la información de nuestras ventas, clientes, tráfico en la web, emails, SMS, envío de catálogos, etc. Así, facilitamos toda la información posible para ayudar a la toma de decisiones o bien, éstas pasan directamente por nosotros mismos. Siguiendo esta línea, en la evolución de los e-commerce y del mercado en general tomando un claro rumbo hacia el online, en la empresa faltaba algo: una herramienta rápida y sencilla que permitiera, en cuestión de horas, saber cómo estaba el mercado en ese momento, cuáles eran los precios de su competencia, ofertas, novedades, etc.

Y esta fue mi motivación: poder ayudar a la empresa a conocer esos datos, que, a pesar de ser públicos, requiere mucho esfuerzo poder extraerlos si no se hace de manera automática.

En ese momento, el departamento de compras se dedicaba a entrar página por página y mirar manualmente los precios y productos de cada marca. Mi objetivo fue hacer cambiar eso.

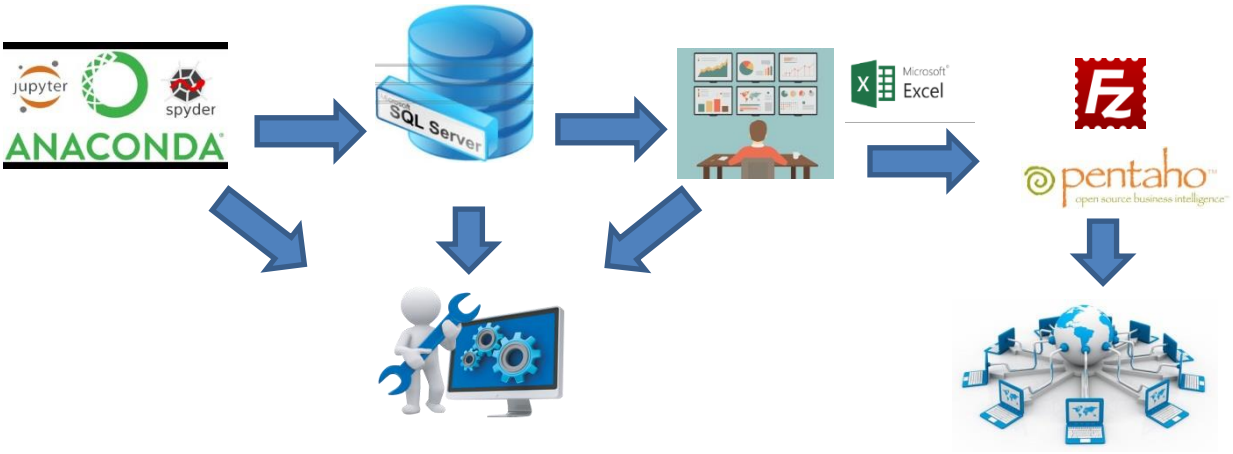
Una vez tuviéramos estas bases de datos, podríamos disponer diariamente, semanalmente o a gusto del cliente, en este caso Venca, de los datos que nos permitieran analizar el mercado. Con ello, sabríamos si estamos muy lejos o nos pasamos a nivel de precios, ofertas o cualquier otra información que el cliente creyera de interés. También le libraríamos, por supuesto, del trabajo manual que tenía que ejercer para analizar los datos, disponiendo así de muchas más horas para mejorar en otros aspectos.

1.2 Objetivos del proyecto

La finalidad de este proyecto consiste en poder analizar el comportamiento de otras marcas en el mercado a través de programas que analicen sus páginas web y de un histórico guardado en bases de datos que nos permitan analizar tanto precios, como ofertas, novedades e incluso predecir campañas o cualquier patrón que encontremos en ellas con tal de mejorar nuestra oferta de precios, adecuarnos al mercado y tener una idea de cómo estamos versus otras marcas.

Para ello necesitamos trabajar con diversos entornos y dividir el proyecto en cuatro fases esenciales:

1. **Programación:**
   - Conseguir los datos necesarios de las páginas web de la competencia a través de scripts.
2. **Incorporación de los datos:**
   - Incorporar los datos en la empresa Venca a través de una nueva estructura de bases de datos dedicada al análisis de la competencia.
3. **Informes:**
   - Generar informes para hacer los datos visibles a cualquier departamento y así permitirles hacer su seguimiento y estudio necesario, además de hacer llegar estos informes mediante email o FTP y poderlo hacer de manera automática gracias a Pentaho.
4. **Mantenimiento:**
   - Tanto la base de datos, como los scripts necesitan de un mantenimiento diario, a nivel de espacio y a nivel de código. Hay que estar alerta que ninguna marca que analizas cambie su código fuente para que tus scripts sigan funcionando correctamente.



1.3 Descripción de la empresa

Venca es una empresa multinacional del sector de la moda especializada en la venta a distancia y con una trayectoria de 48 años de experiencia en el sector. Comenzó su actividad económica en Cataluña en el año 1979 y actualmente las instalaciones de 41 000 m² están situadas en Vilanova i la Geltrú, provincia de Barcelona.

En el año 1988, el grupo francés 3 Suisses International* adquiere la mayoría accionarial de Venca, lo que impulsa su expansión comercial en el territorio nacional. Siguiendo esta línea, en el 1997 lanzó su página Web como nueva plataforma de venta a distancia, convirtiéndose en pionera en la venta mediante Internet, también conocido como e-commerce.

Más allá de la venta a distancia, Venca cuenta con dos tiendas físicas enfocadas a la venta de artículos de temporadas pasadas u outlets, además de ofrecer a los clientes servicios de recogida y devolución de sus pedidos.

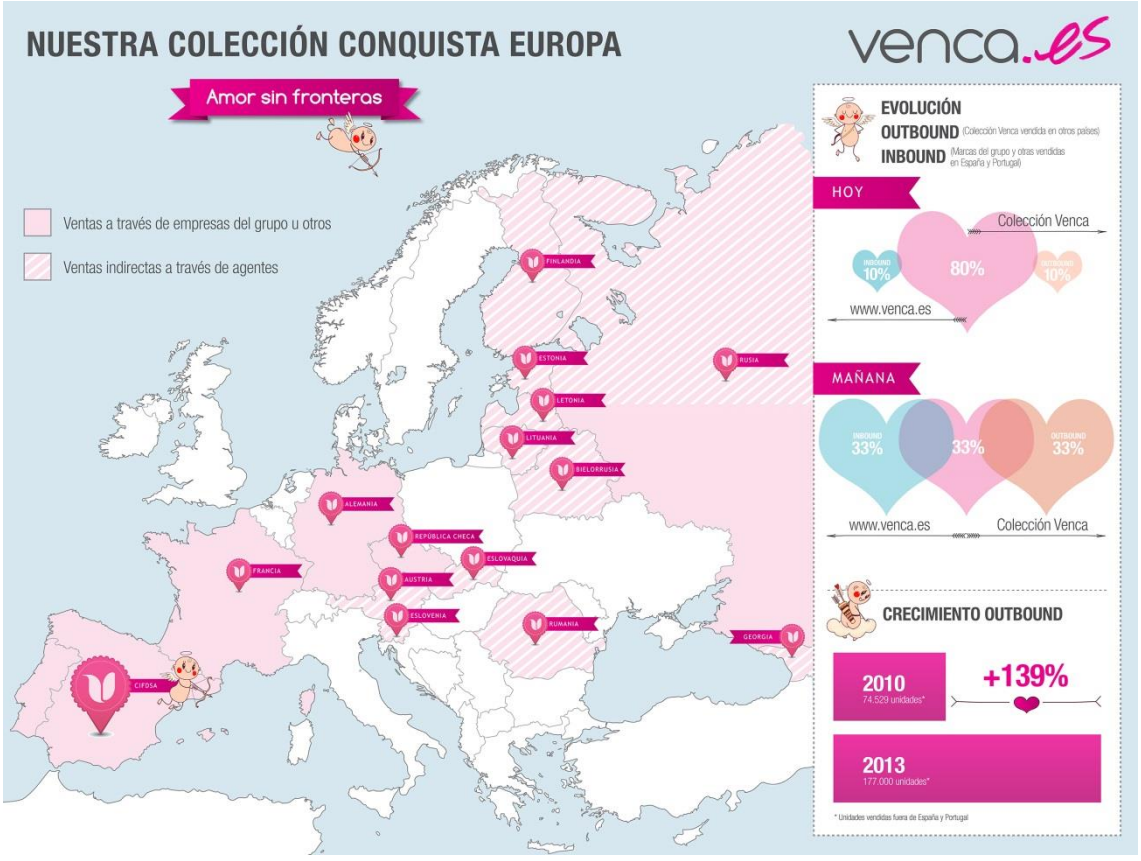
En la búsqueda de ampliación de la cuota de mercado, en el año 2011 inicia la venta de la colección a través de las plataformas de ventas de 3 Suisses International. De igual forma, Venca pasa a ofrecer y distribuir los productos de la compañía francesa mediante su plataforma Web.

Con la ampliación del mercado, la empresa busca combatir el descenso de la cuota en el terreno nacional. La facturación en el año 2011 fue de 36 601 860 €, mientras que, en el año 2013, ésta presentó un descenso del 21,65%, situándose en 28 676 850 €. Todo ello propició, en el año 2012, el Expediente de Regulación de Empleo (ERE).

El Grupo 3 Suisses International, corporación de origen francés considerada una de los más importantes del mundo en venta a distancia. En total agrupa a 22 empresas, entre las que se encuentra Venca, con más de 8.300 personas en un total de 16 países. (INDISA, 2009).

En el año 2013, la empresa vivió una nueva expansión de la inversión extranjera cuando la compañía alemana Otto adquirió el poder accionarial de 3 Suisses International, quedando Venca ligada al grupo alemán. Tras esta última adquisición, la compañía amplió la cuota de mercado a nivel europeo a través del sistema de distribución alemán. En concreto, pasó a comercializar su colección en 15 países.

El objetivo de dicha expansión reside en ampliar paulatinamente el modelo de negocio electrónico por encima del generado con el catálogo de papel tradicional. Para ello, la compañía ha trabajado en la creación de un nuevo portal Web adaptado a los clientes y que permita ampliar el nivel de facturación.



Es por este motivo por el cual la empresa necesita una herramienta capaz de aplicar métodos estadísticos para la búsqueda de comportamientos homogéneos dentro de todo el mercado Web que permitan predecir su evolución y, de esta manera, poder aumentar los ingresos de esta.

Esta herramienta se desarrollará dentro del equipo de Business Intelligence. El equipo está compuesto por un total de cuatro personas, de entre las cuales yo mismo seré el único encargado de crearla y mantenerla. Por tanto, mi rol principal dentro del equipo será analizar los datos de la competencia que yo mismo generaré. Los otros 3 roles se basarán, al fin y al cabo y de una manera u otra, en analizar los datos de la empresa, generar informes para cada departamento y aportar nuestras ideas y conocimientos extraídos de los análisis, entre los cuales pueden encontrarse desde el análisis de ventas de cualquier acción, pasando por el análisis del tráfico web/mobile y hasta el análisis de emails, entre otros.

1.4 La venta a distancia

Se consideran ventas a distancia "*las celebradas sin la presencia física simultánea del comprador y del vendedor*" (LOCM, 1996), transmitiéndose la propuesta de contratación del vendedor y la aceptación del comprador por un medio de comunicación a distancia de cualquier naturaleza (correo, catálogo, televisión, radio, teléfono, etc.).

La venta por catálogo, por teléfono o por internet son tres modalidades de venta a distancia que se caracterizan porque la oferta del vendedor y la aceptación del comprador se realizan por correo o catálogo, televisión, radio, teléfono o internet.

Las ventas a distancia junto con la venta ambulante, las ventas automáticas y las ventas en pública subasta forman parte de las ventas especiales que regula la Ley 17 / 1996 de Comercio.

1.4.1 E-Commerce

El e-commerce o Comercio Electrónico, consiste en la distribución, compra, venta, marketing y suministro de información de productos o servicios a través de Internet. La incorporación de este nuevo método de ventas permite que los clientes accedan de manera simple y desde cualquier parte del mundo a los productos y servicios que una empresa ofrece.

Tiene una serie de ventajas que han permitido el crecimiento de manera significativa en los últimos años:

- Expandir la base de clientes al entrar a un mercado más amplio.
- Extender el horario de venta hasta las 24h del día los siete días de la semana, 365 días del año.
- Crear una ventaja competitiva.
- Reducir costos de producción, capital y administración, entre otros.
- Mejorar la comunicación con los clientes y la efectividad de campañas publicitarias.

También tiene una serie de inconvenientes a los que cualquier empresa e-commerce tiene que enfrentarse y que no podemos dejar de mencionar:

- Inseguridad a la hora de comprar online (p.e. tener que dar información confidencial como las tarjetas de crédito).
- Inseguridad en el envío de artículos.
- No tener el tacto y la posibilidad de probar el articulo antes de comprarlo.

Estos inconvenientes vemos que cada año se van dejando atrás y que la gente va perdiendo el miedo a la compra online. Cada día que pasa el mundo offline se va fusionando con el online, y empresas como Venca también hacen todo lo posible para que así ocurra mediante la creación de acciones como “pop-ups”, tiendas físicas en las cuales dar a conocer la marca y dar la posibilidad a la gente de probarse la ropa en la tienda y pedirla por internet desde allí mismo.

El crecimiento de las cifras de negocio en el E-commerce durante los últimos años le ha situado como el modelo de referencia en la venta a distancia, situación apetecible para las empresas. Hay una serie de factores que apuntan a su viabilidad:

- *Conexiones a internet cada vez más rápidas.*

Las mejoras en las infraestructuras y la tecnología utilizada en las telecomunicaciones han facilitado el acceso al nuevo entono de compras.

- *Comodidad*.

El cliente puede acceder al catálogo de productos en cualquier momento sin necesidad de desplazarse al punto de venta.

- *Expansión de los dispositivos móviles.*

Hoy día el consumidor cuenta con un mayor abanico de dispositivos, como el Smartphone o la Tablet, con los que acceder a Internet y a entornos de compras online.

- *Ahorro*.

Al eliminar costes respecto a las tiendas físicas, el cliente puede encontrar productos a precios más competitivos.

- *Costumbres*

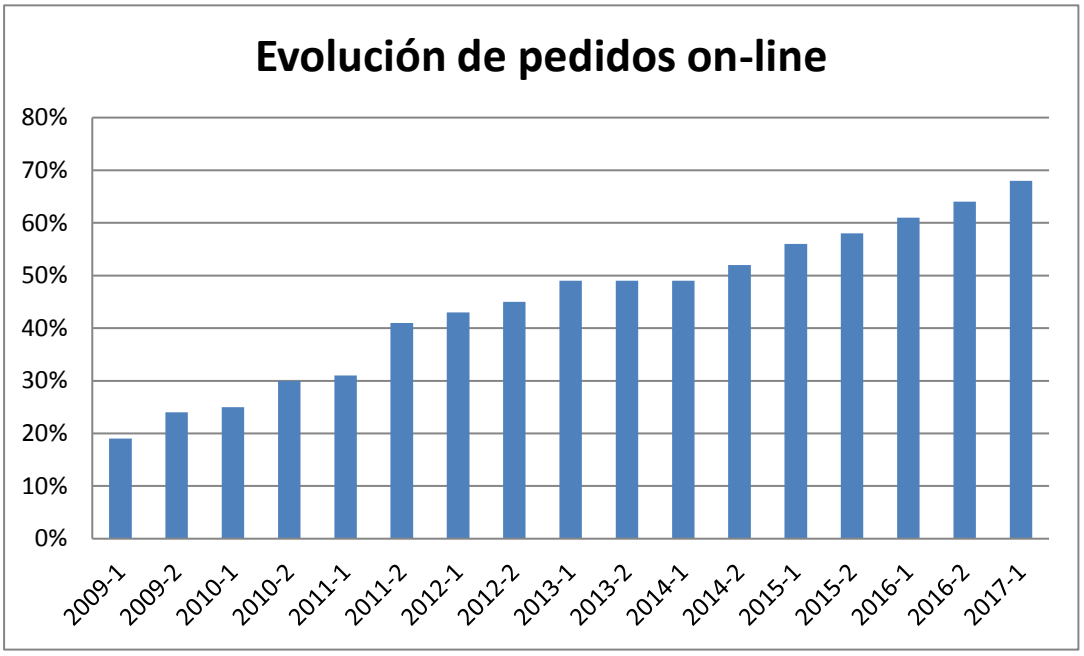
A medida que la sociedad se familiariza con Internet, desaparece el temor a realizar las compras online.

1.4.2 La venta a distancia en Venca

Venca se ha caracterizado por la venta a través del catálogo desde hace más de 30 años, tal y como se ha explicado antes. Este tipo de venta a distancia se denomina Off-line, ya que el cliente recibe en su casa un catálogo y para realizar un pedido envía su hoja de petición por correo o fax, o llama por teléfono.

En los últimos seis años esto ha cambiado. Venca, a la vez que toda la sociedad, está cada vez más ligada al mundo on-line. Esto ha hecho que la evolución de la venta a distancia online dentro de Venca haya evolucionado considerablemente. Tanto es así, que actualmente más del 60% de los pedidos generados en Venca son hechos on-line.

En la siguiente gráfica podemos ver la evolución semestral de pedidos on-line desde el año 2009 hasta la actualidad:

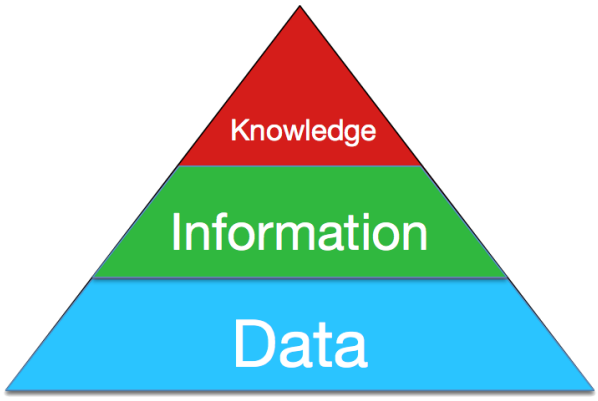


A raíz de este cambio dentro de la empresa, Venca necesita una nueva visión y forma de analizar a la competencia, ya que todo está on-line. Una herramienta importante y necesaria es un analizador para poder tratar todos los datos que obtienen a través de diferentes marcas y conseguir una mayor estrategia de mercado para la buena conversión dentro de la empresa.

2. ASPECTOS TEÓRICOS

2.1 ¿Qué es Business Intelligence?

Business Intelligence es la habilidad para transformar los datos en información y ésta en conocimiento, de forma que se pueda optimizar el proceso de toma de decisiones en los negocios.



Desde un punto de vista más pragmático y asociándolo directamente con las Tecnologías de la Información, podemos definir Business Intelligence como el conjunto de metodologías, aplicaciones y tecnologías que permiten reunir, depurar y transformar datos de los sistemas transaccionales e información desestructurada (interna y externa a la compañía) en información estructurada para su explotación directa (reporting, análisis OLTP / OLAP, alertas…) o para su análisis y conversión en conocimiento, dando así soporte a la toma de decisiones sobre el negocio.

La inteligencia de negocio actúa como un factor estratégico para una empresa u organización, generando una potencial ventaja competitiva, que no es otra que proporcionar información privilegiada para responder a los problemas de negocio: entrada de nuevos mercados, promociones u ofertas de productos, eliminación de islas de información, control financiero, optimización de costes, planificación de la producción, análisis de perfiles de clientes, rentabilidad de un producto concreto, etc.

Los principales productos de Business Intelligence que existen hoy en día son:

- Cuadros de mando Integrales (CMI)
- Sistemas de Soporte a la Decisión (DSS)
- Sistemas de Información Ejecutiva (EIS)

Por otro lado, los principales componentes de orígenes de datos en el Business Intelligence (en adelante BI) que existen en la actualidad son:

- Datamart
- DataWarehouse

Los sistemas y componentes del BI se diferencian de los sistemas operacionales en que están optimizados para preguntar y divulgar sobre datos. Esto significa típicamente que, en un dataWarehouse, los datos están desnormalizados para apoyar consultas de alto rendimiento, mientras que en los sistemas operacionales suelen encontrarse normalizados para apoyar operaciones continuas de inserción, modificación y borrado de datos. En este sentido, los procesos ETL (extracción, transformación y carga), que nutren los sistemas BI, tienen que traducir de uno o varios sistemas operacionales normalizados e independientes a un único sistema desnormalizado, cuyos datos estén completamente integrados.

En definitiva, una solución BI completa permite:

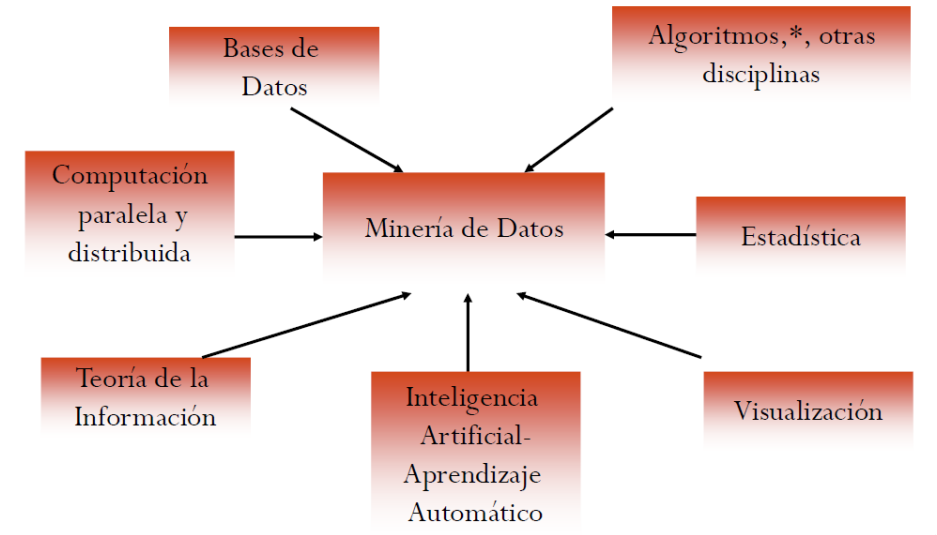
- ***Observar*** ¿Qué está ocurriendo?
- ***Comprender*** ¿Por qué ocurre?
- ***Predecir*** ¿Qué ocurrirá?
- ***Colaborar*** ¿Qué debería hacer el equipo?
- ***Decidir*** ¿Qué camino se debe seguir?

2.2 Minería de datos

En la parte más baja de la pirámide, tendremos quizás miles o millones de datos a los cuales habrá que hacer un procesamiento para extraer solo lo más importante y transformarlos a una estructura que sea comprensible para poder leerlos más tarde. Esto es exactamente para lo que utilizaremos minería de datos.

La Minería de Datos es un proceso que se tiene que centrar en las acciones derivadas del descubrimiento de conocimiento, no en el mecanismo de descubrimiento en sí mismo. La solución es más que un conjunto de técnicas y herramientas.

Este proceso está formado por diferentes ramas que hacen que llegues al resultado esperado:



El fin de este proceso es, principalmente, obtener los siguientes casos dentro la empresa:

- Conocimiento de los clientes
- Detección de segmentos
- Cálculo de perfiles
- Cross-selling
- Detección de buenos clientes
- Mejora de respuesta de mailings
- Campañas de adquisición de clientes.

Para ello, tal y como se menciona antes, se necesitan varios procesos antes de obtener el dato a estudiar.

2.2.1 Preprocesamiento de datos

Como primer paso es necesario un preproceso de datos, ya que los datos reales suelen estar “sucios”, es decir:

- Incompletos: se han perdido valores de atributos, atributos de interés o los datos están resumidos.
- Ruido: errores y “outliers”.
- Inconsistentes: hay discrepancias en los nombres y/o valores.

Por tanto, el “Preprocesamiento de datos” engloba a todas aquellas técnicas de análisis de datos que permiten mejorar la calidad de un conjunto de datos de modo que las técnicas de extracción de conocimiento/minería de datos puedan obtener mayor y mejor información, mejor porcentaje de clasificación, reglas con más completitud, etc.

Este paso es muy importante dado que sin calidad de los datos no habrá calidad en los resultados del proceso.

Las principales tareas del preprocesamiento son:

- Limpieza de datos: Rellenar los valores nulos, identificar y/o eliminar los outliers, resolver las inconsistencias, tratar los valores con ruido…
- Integración de datos: Combinar datos de diferentes fuentes.
- Transformación de los datos: Normalización y agregación.

2.2.2 Clustering o Agrupamiento

Un Clúster es un grupo o conjunto de objetos que:

- Son similares a cualquier otro incluido en el mismo clúster.
- Disimilares (distintos) a los objetos incluidos en otros grupos.

La definición de Clustering (análisis clúster) es segmentar una población heterogénea en un número de subgrupos homogéneos o clústeres. Con ello se pretende que dado un conjunto de datos el algoritmo dé como resultado una división de la población, de manera que se minimiza la distancia de los elementos de cada grupo o “clúster” y se maximiza la distancia inter-clase.

Un buen método de clustering producirá clusters de calidad:

- Alta: similitud dentro de cada clase (Inter-clase)
- Baja: similitud de elementos de distintas clases.

La calidad de los resultados depende de la medida de similitud que se utilice y en su implementación. También se mide por su capacidad para descubrir patrones ocultos. Si los datos tienen significados diferentes se pueden aplicar pesos sobre las variables.

Existen una serie de requerimientos de los métodos de clustering dentro de la Minería de datos:

- Escalables
- Tratar distintos tipos de variables
- Descubrimiento de cluster con formas arbitrarias
- Requisitos mínimos del dominio para determinar los parámetros
- Capaz de tratar datos con ruido
- Insensible al orden de los registros
- Resultados inteligibles (interpretables)

2.2.3 Técnicas de minería de datos

Es importante tener una visión de cómo trabajar (por dentro) para:

- Distinguir entre las distintas técnicas y conocer sus ventajas y desventajas.
- Entender qué técnica es más apropiada para cada tipo de problema.
- Familiarizarse con sus parámetros, entradas, salidas.

Distintas metas piden distintas técnicas (enfoque conceptual para extraer información):

- Prescriptiva: Su meta es automatizar el proceso de toma de decisión por medio de la construcción de un modelo que sea capaz de realizar una predicción, ya sea asignando un elemento a una clase o realizando una estimación de un valor.
- Descriptiva: Obtener una mejor comprensión de lo que ocurre en los datos y, como consecuencia, del mundo que reflejan.

También nos encontramos que distintos tipos de datos requieren distintos algoritmos (forma detallada paso a paso de implementar una técnica):

- Si la meta del problema es predecir un valor numérico como el valor de vida de un cliente o la carga de un vuelo, el algoritmo tendrá que ser capaz de producir un valor numérico.
- Si las variables son todas categóricas, o se busca un algoritmo que las admita o se tendrán que transformar.
- Tanto las variables de entrada como las de salida se tienen que tener en cuenta al elegir un algoritmo.

Para poder analizar un conjunto de datos y poder determinar las características de los mismos (creación de un módulo) es importante su clasificación.

Existen varios tipos de clasificación:

- Entrada de los algoritmos:
  - Atributos condición: Atributos usados para describir por medio del proceso de inducción las clases de equivalencia.
  - Atributos decisión o label: son atributos usados para construir las clases de equivalencia en los métodos supervisados (una clase para cada valor o combinación de valores de dichos atributos).

- Representación del error:
    - Matriz de confusión: Representación en forma de tabla del número de instancias clasificadas correctamente.
- Construcción del modelo (describir un conjunto de datos en base a una característica):
    - Cada tupla pertenece a una clase predefinida determinada por el atributo de decisión.
    - Se utiliza el conjunto de datos de entrenamiento.
    - El modelo se representa a través de reglas de clasificación, árboles de decisión o mediante fórmulas matemáticas.
- Utilización del modelo (para clasificar objetos nuevos de los que se desconoce su clase):
    - Determinación de la precisión del modelo.
    - Se utiliza el modelo para clasificar el conjunto de datos de entrenamiento y se compara el resultado con la etiqueta original.
    - La exactitud es el porcentaje de conjunto de datos de prueba que son clasificados correctamente por el modelo.
    - El conjunto de datos entrenamiento y el conjunto de datos de prueba deben ser disjuntos, para evitar el ajuste excesivo.

3. TECNOLOGÍAS UTILIZADAS

Para el correcto desarrollo del proyecto son necesarias una serie de herramientas con las cuales se trabaja en la empresa y que, si bien no han sido impuestas, he decidido utilizar por una mera cuestión de comodidad a la hora de utilizar todos las mismas.

A continuación, se definen palabras clave importantes para poder entender el funcionamiento de cada una de estas tecnologías:

- **Python**: es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.

- **Anaconda/Spyder**: es un potente entorno de desarrollo interactivo para el lenguaje Python con funciones avanzadas de edición, pruebas interactivas, depuración e introspección.

- **SQL**: es el lenguaje estándar ANSI/ISO de definición, manipulación y control de bases de datos relacionales. Es un lenguaje declarativo: sólo hay que indicar qué se quiere hacer. En cambio, en los lenguajes procedimentales es necesario especificar cómo hay que hacer cualquier acción sobre la base de datos. El SQL es un lenguaje muy parecido al lenguaje natural; concretamente, se parece al inglés, y es muy expresivo. Por estas razones, y como lenguaje estándar, el SQL es un lenguaje con el que se puede acceder a todos los sistemas relacionales comerciales.

- **Transact-SQL (TSQL):** es una extensión al SQL de Microsoft y Sybase. Es un lenguaje muy potente que nos permite definir casi cualquier tarea que queramos efectuar sobre la base de datos.

  Nos permite:

  - Crear interfaces de usuario.

  - Crear aplicaciones ejecutables, es decir, elementos que en algún momento llegarán al servidor de datos y serán ejecutados.

Incluye características propias de cualquier lenguaje de programación, características que nos permiten definir la lógica necesaria para el tratamiento de la información:

- Tipos de datos.
- Definición de variables.
- Estructuras de control de flujo.
- Gestión de excepciones.
- Funciones predefinidas.

Debido a estas restricciones, se emplea generalmente para crear procedimientos almacenados, triggers y funciones de usuario.

- **DML**: *Lenguaje de manipulación de datos (Data Manipulation Language)* es un lenguaje proporcionado por los sistemas gestores de bases de datos que permite a los usuarios de la misma llevar a cabo las tareas de consulta o modificación de los datos contenidos en las Bases de Datos del Sistema Gestor de Base de Datos. El lenguaje de manipulación de datos más popular hoy día es SQL, usado para recuperar y manipular datos en una base de datos relacional.

- **DDL**: *Lenguaje de definición de datos (Data Definition Language)* es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios de la misma llevar a cabo las tareas de definición de las estructuras que almacenarán los datos, así como de los procedimientos o funciones que permitan consultarlos.

  Un DDL es un lenguaje de programación para definir estructuras de datos. Actualmente se utiliza en un sentido genérico para referirse a cualquier lenguaje formal para describir datos o estructuras de información, como los esquemas XML.

- **Procedimiento almacenado** (Store Procedure-SP): es un programa almacenado físicamente en una base de datos. Su implementación varía de un gestor de bases de datos a otro.

  Usos típicos para procedimientos almacenados incluyen la validación de datos siendo integrados a la estructura de base de datos (los SP utilizados para este

propósito a menudo son llamados disparadores, *triggers* en inglés), o encapsular un proceso grande y complejo. El último ejemplo generalmente ejecutará más rápido como un procedimiento almacenado que de haber sido implementado como, por ejemplo, un programa corriendo en el sistema cliente y comunicándose con la base de datos mediante el envío de consultas SQL y recibiendo sus resultados.

Se usan a menudo, pero no siempre, para realizar consultas SQL sobre los objetos de la base de datos de una manera abstracta, desde el punto de vista del cliente de la aplicación, ya que permite agrupar en forma exclusiva parte de algo específico que se desee realizar o, mejor dicho, el SQL apropiado para dicha acción.

La ventaja de un procedimiento almacenado, en respuesta a una petición de usuario, está directamente bajo el control del motor del gestor de bases de datos, que corre generalmente en un servidor distinto del servidor web, aumentado con ello la rapidez de procesamiento de las peticiones del usuario. El servidor de la base de datos tiene acceso directo a los datos necesarios para manipular y sólo necesita enviar el resultado final al usuario. Pueden permitir que la lógica del negocio se encuentre como un API en la base de datos, que pueden simplificar la gestión de datos y reducir la necesidad de codificar la lógica en el resto de los programas cliente. Esto puede reducir la probabilidad de que los datos se corrompan por el uso de programas clientes defectuosos o erróneos. De este modo, el motor de base de datos puede asegurar la integridad de los datos y su consistencia con la ayuda de procedimientos almacenados. Algunos afirman que las bases de datos deber ser utilizadas para el almacenamiento de datos solamente, y que la lógica de negocio sólo debería aplicarse en la capa de negocio de código, a través de aplicaciones cliente que deban acceder a los datos. Sin embargo, el uso de procedimientos almacenados no se opone a la utilización de una capa de negocio.

- **Transacciones**: en un sistema de gestión de Bases de Datos (SGBD) es un conjunto de órdenes que se ejecutan formando una unidad de trabajo, es decir, en forma indivisible o atómica.

  Un SGBD se dice transaccional, si es capaz de mantener la integridad de los datos, haciendo que estas transacciones no puedan finalizar en un estado intermedio. Cuando por alguna causa el sistema debe cancelar la transacción empieza a deshacer las órdenes ejecutadas hasta dejar la base de datos en su

estado (llamado punto de integridad), como si la orden de la transacción nunca se hubiese realizado.

Para que un SDBD sea considerado transaccional debe cumplir los criterios de *atomicidad, consistencia, aislamiento* y *durabilidad*.

Para esto, el lenguaje de consulta de datos SQL provee los mecanismos para especificar que un conjunto de acciones debe constituir una transacción:

- BEGIN TRAN: Especifica que va a empezar una transacción.
- COMMIT TRAN: Le indica al motor que puede considerar la transacción completada con éxito.
- ROLLBACK TRAN: Indica que se ha alcanzado un fallo y que debe restablecer la base al punto de integridad.

En un sistema ideal, las transacciones deberían garantizar todas las propiedades, pero en la práctica, a veces alguna de estas propiedades se simplifica o debilita con vistas a obtener un mejor rendimiento.

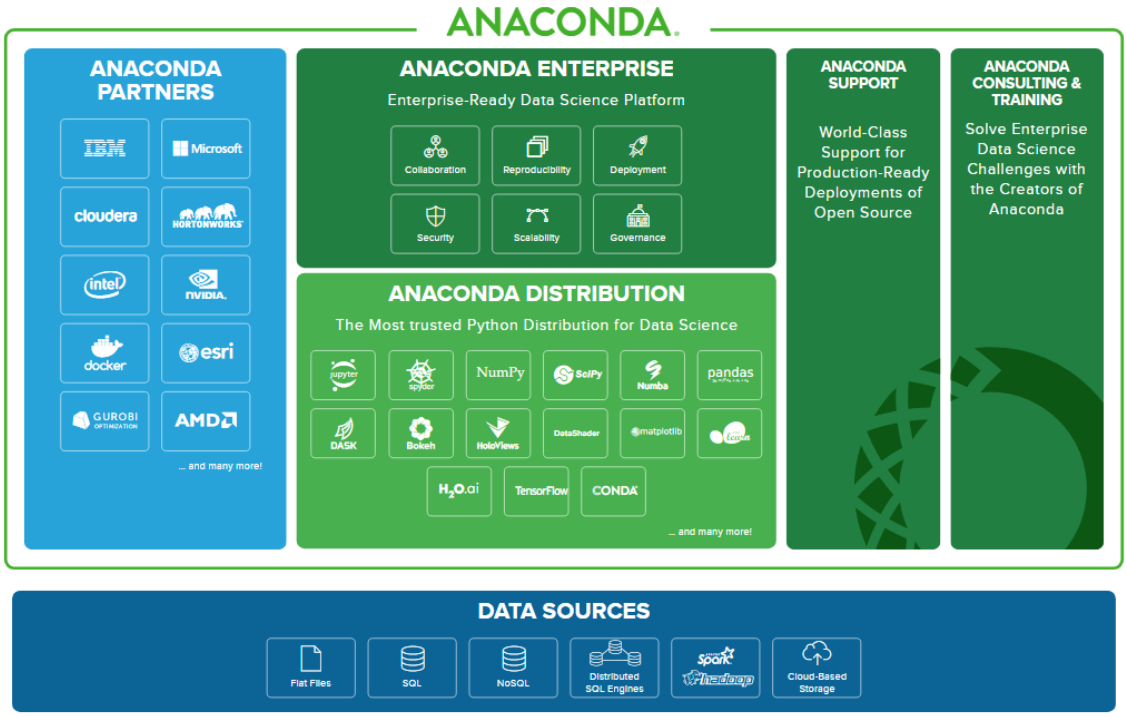
- **FTP** (File Transfer Protocol): es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (Transmission Control Protocol), basada en la arquitectura cliente-servidor. Desde un equipo cliente se puede conectar a un servidor para descargar archivos desde él o para enviarle archivos, independientemente del sistema operativo utilizado en cada equipo.

  El servicio FTP es ofrecido por la capa de aplicación del modelo de capas de red TCP/IP al usuario, utilizando normalmente el puerto de red 20 y el 21. Un problema básico es que está pensado para ofrecer la máxima velocidad en la conexión, pero no la máxima seguridad. Para solucionar este problema son de gran utilidad aplicaciones como **SFTP**, incluidos en el paquete SSH, que permiten transferir archivos, pero cifrando todo el tráfico.

- **VBA**: es el lenguaje de macros de Microsoft Visual Basic que se utiliza para programar aplicaciones Windows y que se incluye en varias aplicaciones Microsoft. **VBA** permite a usuarios y programadores ampliar la funcionalidad de programas de la suite Microsoft Office.

3.1 Anaconda

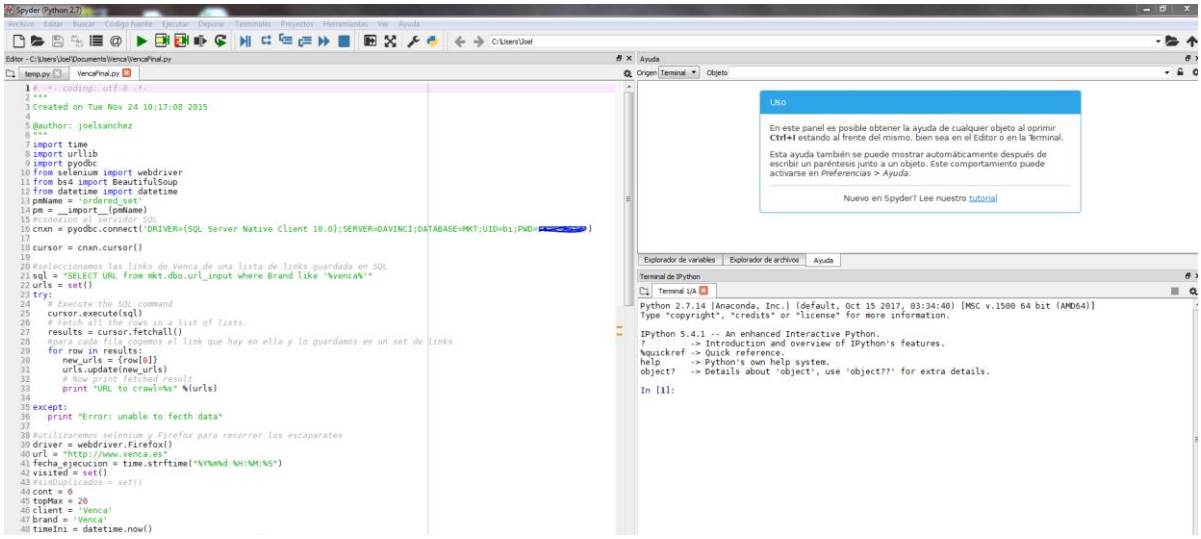
Con más de 4.5 millones de usuarios, Anaconda es la plataforma de ciencia de datos Python más popular del mundo. Anaconda, Inc. continúa liderando proyectos de código abierto como Anaconda, NumPy y SciPy, que forman la base de la ciencia de datos moderna. El producto estrella de Anaconda, Anaconda Enterprise, permite a las organizaciones asegurar, gobernar, escalar y ampliar Anaconda para ofrecer conocimientos accionables que impulsen a las empresas y las industrias.



Producto Anaconda

Dentro de su inmenso universo, hemos elegido Spyder, como herramienta principal para programar los scripts en Python.

Spyder: (anteriormente Pydee) es un entorno de desarrollo integrado y multiplataforma de código abierto (IDE) para programación científica en el lenguaje Python. Spyder integra NumPy, SciPy, Matplotlib e IPython, así como otro software de código abierto. Se lanzó bajo la licencia de MIT.



Interfaz de usuario de Spyder

Aquí podemos ver un ejemplo de script, escrito en Python y en la plataforma Spyder.

En resumen, gracias a Anaconda y en concreto a Spyder, podremos generar nuestros scripts de análisis y los podremos generar en un entorno amigable y fácil de utilizar.

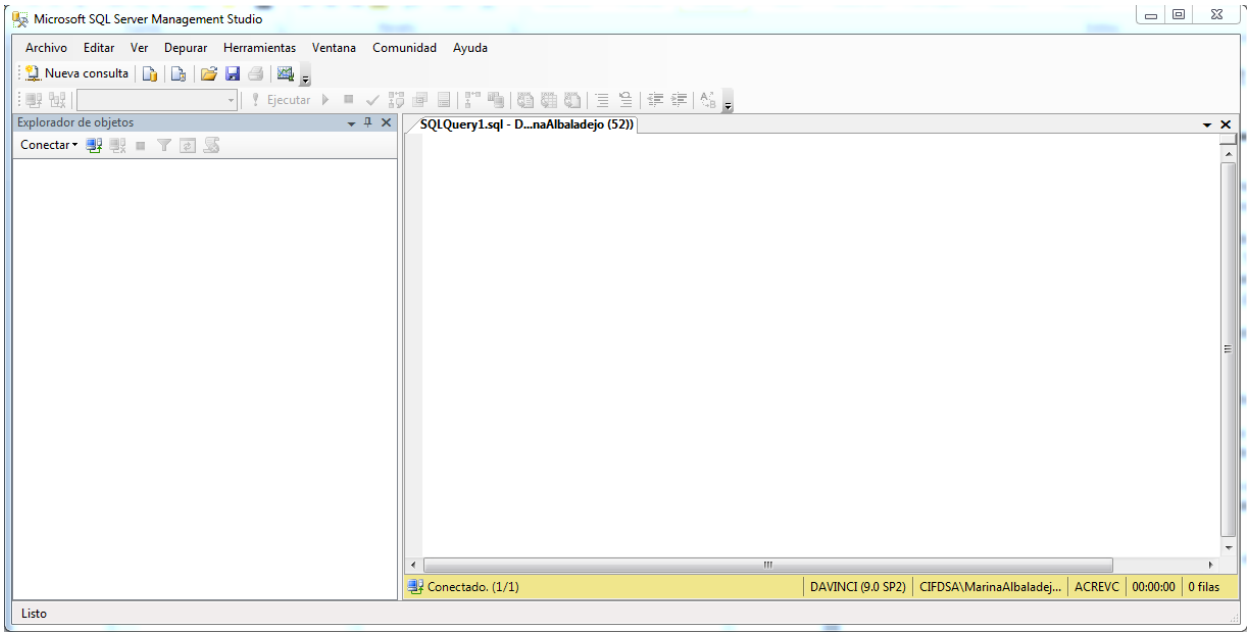
3.2 Microsoft SQL Server

Microsoft SQL Server es un sistema de manejo de bases de datos del modelo relacional desarrollado por la empresa Microsoft.

El lenguaje de desarrollo utilizado (por la línea de comandos o mediante la interfaz gráfica de management studio) es Transact-SQL (*TSQL*), una implementación del estándar ANSI del lenguaje *SQL*, utilizado para manipular y recuperar datos (*DML*), crear tablas y definir relaciones entre ellas (*DDL*).

Las características principales de esta herramienta son:

- Soporte de *transacciones*.
- Soporta *procedimientos almacenados*.
- Incluye también un entorno gráfico de administración, que permite el uso de comandos DDL y DML gráficamente.
- Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y los terminales o clientes de la red sólo acceden a la información.
- Permite administrar información de otros servidores de datos.



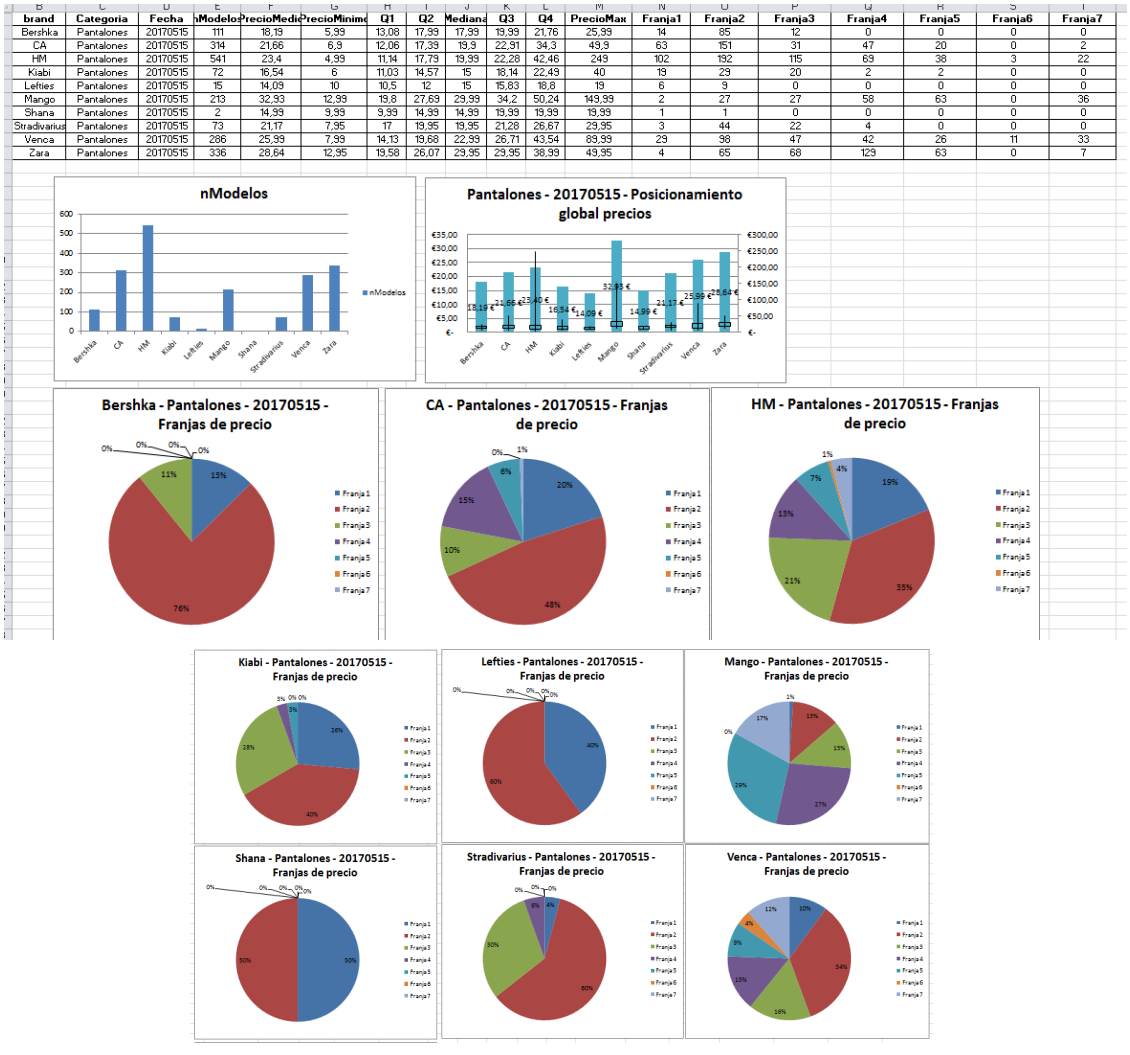
Interfaz de usuario de Microsoft SQL Server

3.3 Excel

Microsoft Excel es una aplicación de hojas de cálculo que forma parte de la suite de oficina Microsoft Office. Permite realizar tareas contables y financieras gracias a sus funciones, desarrolladas específicamente para ayudar a crear y trabajar con hojas de cálculo.

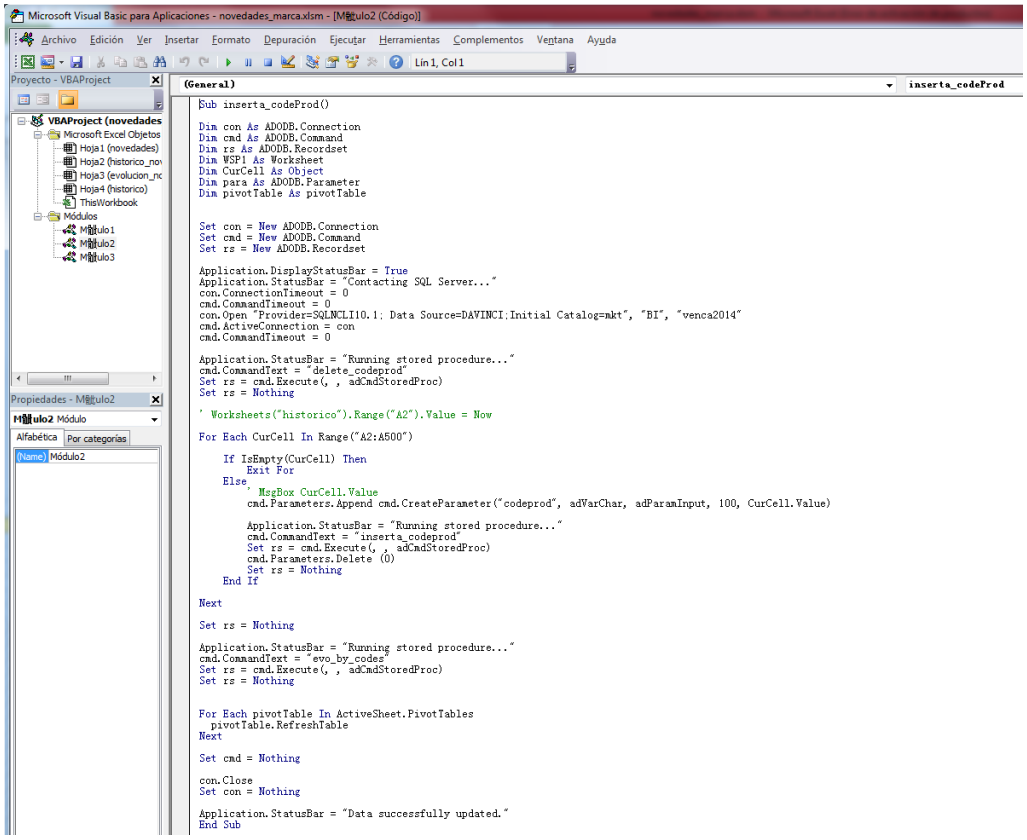
Esta aplicación nos permitirá dar forma a nuestros datos, organizándolos ya sea en números o texto y en diferentes hojas de cálculo.

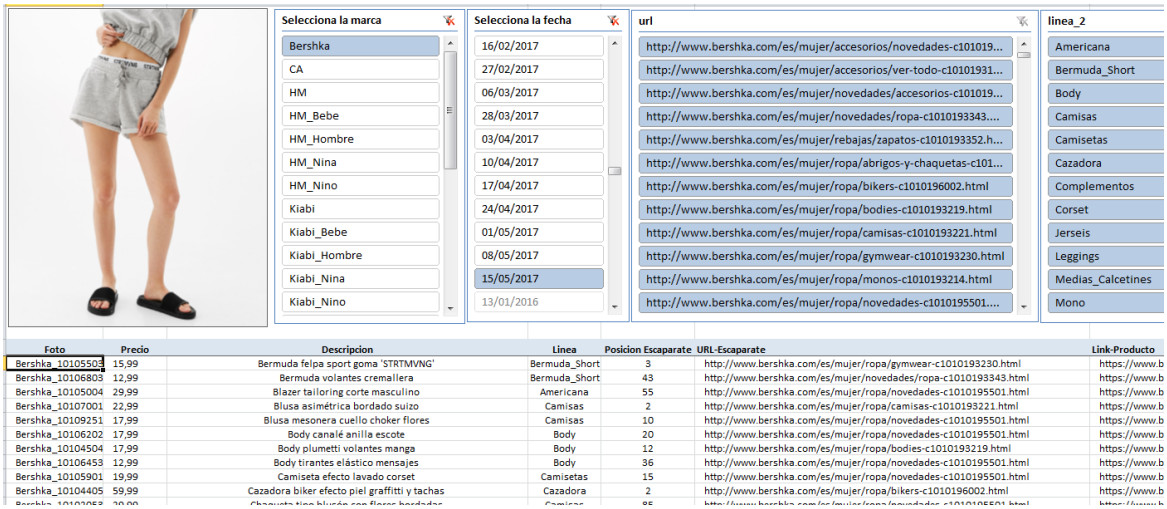
También nos permitirá su análisis, gracias a gráficos, tablas dinámicas e infinidad de herramientas que nos ofrece.



VBA: Microsoft VBA viene integrado en aplicaciones de Microsoft Office, como Outlook, Word, Excel, Access y Powerpoint. Es el lenguaje de macros de Microsoft Visual Basic que se utiliza para programar aplicaciones Windows.

VBA nos ha permitido programar muchas funcionalidades que Excel no te ofrecía, como por ejemplo la generación automática de los informes, o incluso llegar a poner imágenes a los productos de forma dinámica.





3.4 Pentaho

Pentaho se define a sí mismo como una plataforma de BI “orientada a la solución” y “centrada en procesos” que incluyen todos los principales componentes requeridos para implementar soluciones basadas en procesos y ha sido concebido desde el principio para estar basada en procesos.

Las soluciones se componen fundamentalmente de una infraestructura de herramientas de análisis e informes integrados con un motor de workflow de procesos de negocio. La plataforma será capaz de ejecutar las reglas de negocio necesarias, expresadas en forma de procesos y actividades y de presentar y entregar la información adecuada en el momento adecuado.

De los diferentes paquetes que ofrece Pentaho, para el proyecto hemos utilizado Pentaho Data Integration (PDI).

PDI está formado por un conjunto de herramientas, cada una con un propósito específico.

- **Spoon / Kettle**: Spoon es la herramienta gráfica que nos permite el diseño de las transformaciones y trabajos del sistema de ETTL’s de Pentaho Data Integration (PDI), también conocido como Kettle (acrónimo recursivo: “Kettle Extraction, Transformation, Transportation and Load Environment).

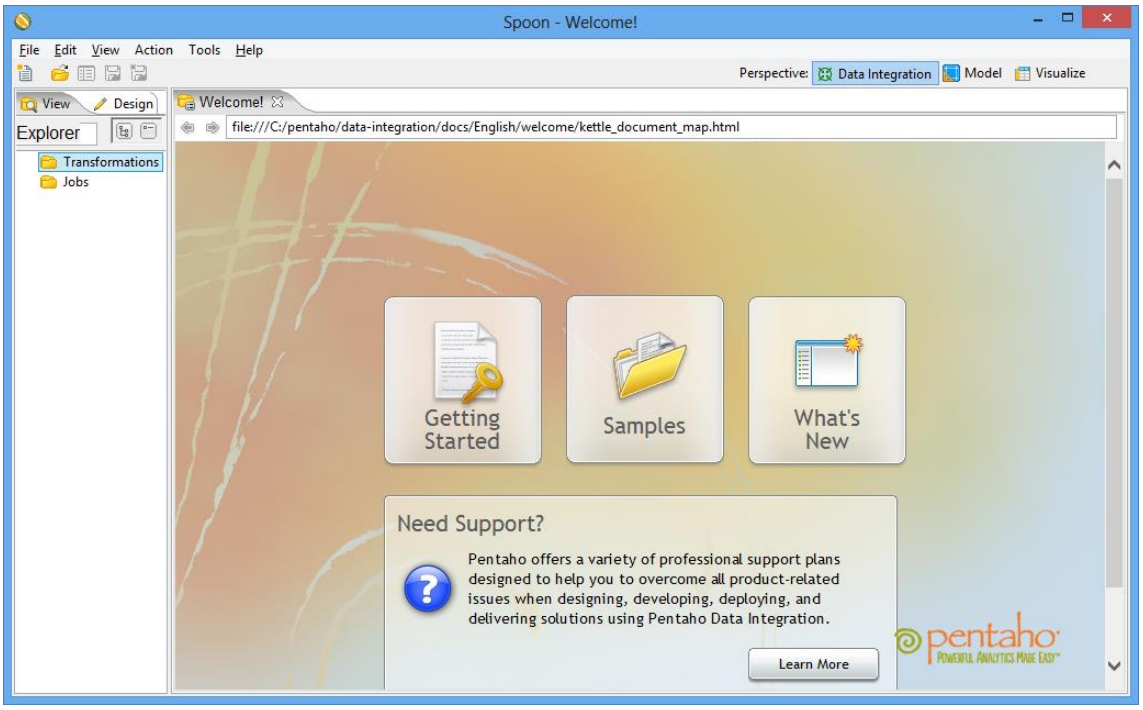
  

  Está diseñado para ayudar en los procesos ETTL’s, que incluyen Extracción, Transformación, Transporte y Carga de datos.

  Es una Interfaz Gráfica de Usuario (GUI) que permite diseñar transformaciones y trabajos que se pueden ejecutar con las herramientas de Kettle (Pan y Kitchen).

  Incluye opciones para previsualizar y testear los elementos desarrollados. Es la principal herramienta de trabajo PDI y con la que construiremos y validaremos nuestros procesos ETL (**E**xtrac**t** **T**ransform **L**oad).

Las transformaciones y trabajos se pueden describir usando un archivo XML o se pueden colocar en un catálogo de base de datos Kettle.



Interfaz de usuario de Spoon

- **Pan**: es la herramienta que nos permite la ejecución de las transformaciones diseñadas en Spoon (bien desde un fichero o desde el repositorio). Nos permite, desde la línea de comandos, preparar la ejecución mediante scripts.
- **Kitchen**: similar a Pan, pero para ejecutar los trabajos o Jobs realizados con Spoon.
- **Carte**: es un pequeño servidor web que permite la ejecución remota de transformaciones y Jobs.

En resumen, PDI facilita la construcción, actualización y mantenimiento de Data WareHouses.

3.5 Filezilla

FileZilla Client es un cliente multiplataforma rápido y fiable de *FTP, FTPS y SFTP* con muchas funcionalidades útiles y una intuitiva interfaz gráfica de usuario.

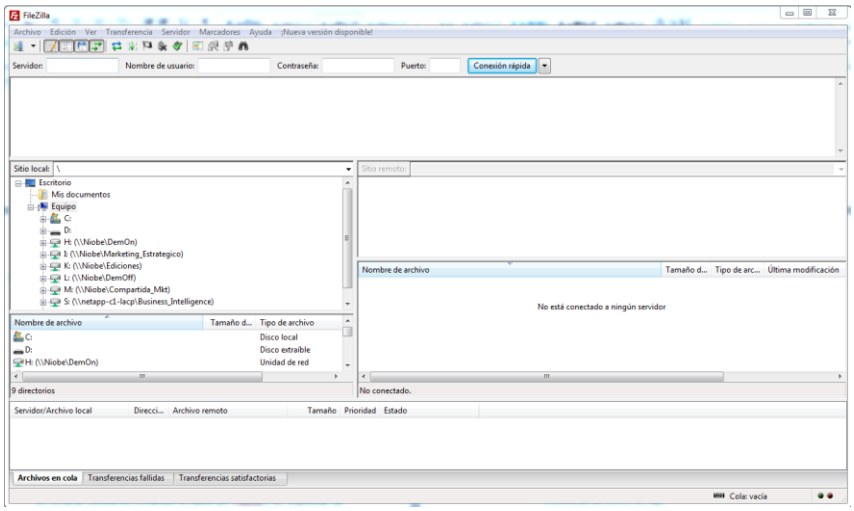
Las características principales de esta herramienta son las siguientes:

- Administrador de sitios: permite a un usuario crear una lista de sitios FTP o SFTP (estableciendo una conexión cifrada que utiliza el protocolo SSH) con sus datos de conexión, como el número de puerto a usar, o si se utiliza inicio de sesión normal o anónima. Para el inicio normal, se guarda el usuario y, opcionalmente, la contraseña.

- Registro de mensajes: se muestra en la parte superior de la ventana. Muestra en forma de consola los comandos enviados por FileZilla y las respuestas del servidor remoto.

- Vista de archivo y carpeta: situada en la parte central de la ventana, proporciona una interfaz gráfica para FTP. Los usuarios pueden navegar por las carpetas, ver y alterar sus contenidos tanto en la máquina local como en la remota utilizando una interfaz de tipo árbol de exploración. Los usuarios pueden arrastrar y soltar archivos entre los ordenadores local y remoto.

- Cola de transferencia: situada en la parte inferior de la ventana, muestra en tiempo real el estado de cada transferencia activa o en cola.



Interfaz de usuario de FileZilla

4. IMPLEMENTACIÓN DEL PROYECTO

4.1 Introducción

Venca, al igual que todas las empresas dedicadas a la venta online, está sufriendo la evolución de los e-commerce, la forma de comprar de sus clientes y la forma de vender de sus competidores. Es por ello por lo que necesita de herramientas para analizar todo este cambio en la medida de lo posible.

Una de estas herramientas será un analizador de la competencia automático que permita al departamento de compras, que actualmente analiza a la competencia entrando manualmente en cada producto de cada marca, disponer de esta información diariamente o en la frecuencia que pidan de manera automática y sin tener que hacer nada.

Este analizador tendrá como objetivo rastrear las páginas webs que la empresa vea convenientes (p.e. Zara, Mango, Kiabi, etc.) con el objetivo de extraer toda la información posible para poder analizar a las distintas empresas de la competencia y sus movimientos en el mercado online, desde extraer precios, novedades, escaparates, posición de los artículos en los escaparates, etc. Con todos estos datos, almacenados en nuestras bases de datos, se generarán una serie de informes para hacerlos visibles y entendibles a cualquier departamento que los solicite.

Para poder llevar a cabo este proyecto se ha decidido dividirlo en 4 fases claves:

- **Programación de los scripts**: Aprender Python y conseguir desarrollar programas que automáticamente entren en X páginas web y logren extraer los datos solicitados.

- **Incorporación de los datos**: Incorporar los datos extraídos en las bases de datos de la empresa.

- **Generación de informes**: Lograr generar informes entendibles y con los datos solicitados de manera casi automática para el posterior análisis por parte del departamento que los solicite, hacer llegar estos análisis mediante emails o automatizarlos gracias a tareas diarias de Windows, Pentaho y Filezilla.

- **Depuración y mantenimiento**: Tanto los scripts como las bases de datos y los informes, necesitaran de una supervisión diaria por cualquier cambio o error que pueda suceder en su ejecución o en el entorno web de la competencia.

Gracias a la información obtenida y a los informes realizados a través de la misma, la empresa ha realizado unos planes estratégicos distintos a los que ejecutaba anteriormente que han supuesto un gran cambio de visión dentro de Venca y que ha hecho que la empresa obtenga los objetivos a los que necesitaba llegar para su buen funcionamiento.

Dispone de dos servidores para poder tratar los datos.

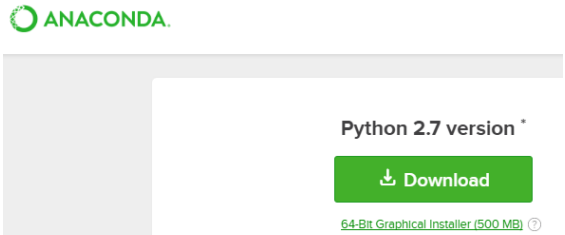
- KOSMOS: Servidor de back-up y con más especio libre donde crearemos nuestra base de datos.
- DAVINCI: Servidor en el que se incorporan todos los datos recibidos de la empresa.

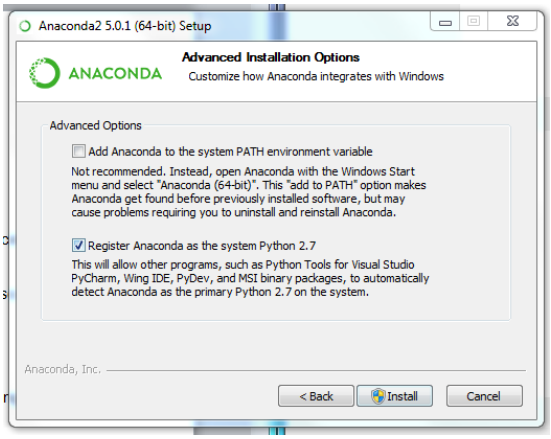
4.2 Programación de los scripts

4.2.1 Instalación de Anaconda

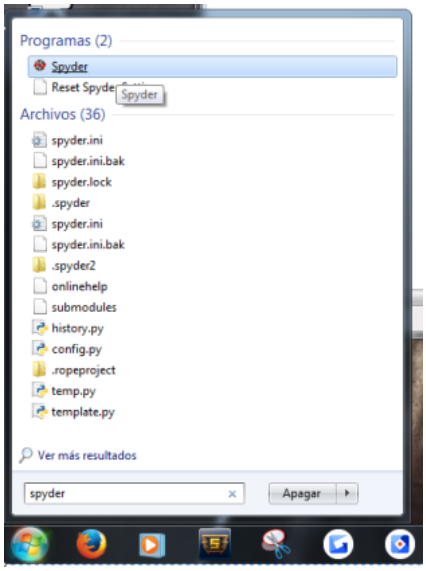
Recomendado por amistades y por mi propio jefe en la empresa, decidí utilizar Spyder como entorno para programar en Python.

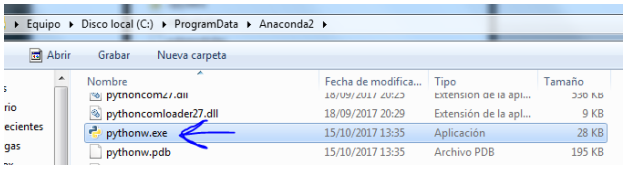
De la propia web de anaconda.org en la sección de downloads, me descargué la versión de Python 2.7 de 64 bits.





Con Anaconda instalado, ya podemos empezar a utilizar Spyder, que lo podremos encontrar en el menú inicio (buscando “Spyder”) o en la propia carpeta de instalación (con otro nombre):



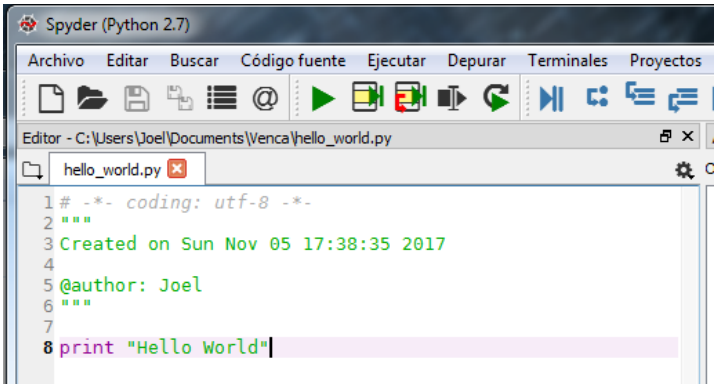


4.2.2 Python

Una de las tareas quizás más difíciles, era aprender un nuevo lenguaje de programación no dado en la carrera o visto muy por encima.

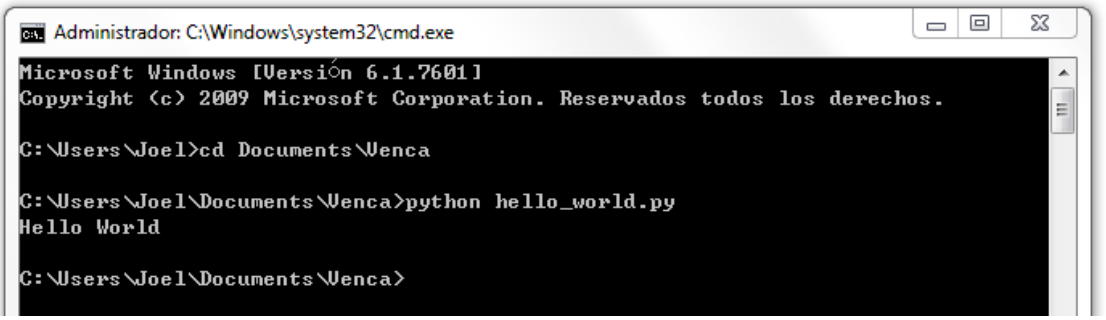
Lo primero y que no podía faltar fue escribir nuestro primer “Hello World!”. Tras buscar como se hacía, aquí ya nos dimos cuenta de que el lenguaje parecía ser bastante más fácil al resto dados en la carrera (C, C++, Java…).

Spyder tiene su propio terminal para las ejecuciones: es tan fácil como ejecutar con F5 o hacer click en el botón de “Play” (también tienes opción de ejecutar por líneas, por selección, etc.)




```
Python 2.7.14 |Anaconda, Inc.| (default, Oct 15 2017, 03:34:40) [MSC v.1500
64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.4.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

runfile('C:/Users/Joel/Documents/Venca/hello_world.py',
wdir='C:/Users/Joel/Documents/Venca')
Hello World
```

También podemos ejecutar los scripts de manera manual con el cmd de Windows:



Escribiendo > python [*nombre de script*], encontrándonos en la misma carpeta.

A partir de aquí todo fue buscar y buscar información sobre como crawlear webs, ver ejemplos, y ponerse a probar y probar código, hasta dar finalmente con las herramientas y programación correcta.

Herramientas utilizadas:

- Instalación vía cmd:
    - "*pip install [selenium/bs4/urllib/pyodbc/time/datetime]*":
- Descripción:
    - **Selenium webdriver**: Gracias a Selenium podemos manejar de forma nativa y desde el propio script un navegador, como puede ser Firefox o Chrome. Esto nos permitirá poder navegar por las páginas que requieran de un scroll manual para ver todos sus artículos o que tengan el código "oculto" en el navegador, entre otros casos.
    - **Bs4 – BeautifulSoup**: BeautifulSoup nos permitirá leer las páginas web, nos extraerá su código HTML y nos permitirá trabajar con él, consultando sus etiquetas y haciendo que podamos guardar en nuestras variables los datos que necesitemos.
    - **Urllib**: Nos permitirá leer urls.
    - **Pyodbc**: Herramienta que nos permitirá de una manera rápida y sencilla conectarnos con nuestras bases ODBC/SQL de datos, tanto para leer como para escribir datos.
    - **Time/datetime**: Nos permitirán poner fecha de ejecución a nuestros scripts y exactamente a cada url visitada.
    - **Ordered_set**: Clase escrita aparte para poder definir el orden en el cual aparecen unos productos en un escaparate predeterminado, y así poder saber los top X posicionados.

Script ordered_set:

```
# -*- coding: utf-8 -*-
"""
@author: joelsanchez
"""
import collections

SLICE_ALL = slice(None)
__version__ = '1.5.0'


def is_iterable(obj):

    return hasattr(obj, '__iter__') and not isinstance(obj, str)


class OrderedSet(collections.MutableSet):

    def __init__(self, iterable=None):
        self.items = []
        self.map = {}
        if iterable is not None:
            self |= iterable

    def __len__(self):
        return len(self.items)

      def __getitem__(self, index):
            if index == SLICE_ALL:
                  return self
            elif hasattr(index, '__index__') or isinstance(index, slice):
                  result = self.items[index]
                  if isinstance(result, list):
                        return OrderedSet(result)
                  else:
                        return result
            elif is_iterable(index):
                  return OrderedSet([self.items[i] for i in index])
            else:
                  raise TypeError("No sé cómo indexar un OrderedSet
por %r" %
                                  index)

      def copy(self):
            return OrderedSet(self)

      def __getstate__(self):
            if len(self) == 0:
                  return (None,)
            else:
                  return list(self)

      def __setstate__(self, state):
            if state == (None,):
                  self.__init__([])
            else:
                  self.__init__(state)

      def __contains__(self, key):
            return key in self.map
```

```
      def add(self, key):
            if key not in self.map:
                  self.map[key] = len(self.items)
                  self.items.append(key)
            return self.map[key]
      append = add



      def update(self, sequence):
            item_index = None
            try:
                  for item in sequence:
                        item_index = self.add(item)
            except TypeError:
                  raise ValueError('El argumento debe ser un iterable,
obtenido %s' % type(sequence))
            return item_index

      def index(self, key):
            if is_iterable(key):
                  return [self.index(subkey) for subkey in key]
            return self.map[key]

      def pop(self):
            if not self.items:
                  raise KeyError('El set está vacío')

            elem = self.items[-1]
            del self.items[-1]
            del self.map[elem]
            return elem


      def discard(self, key):
            if key in self:
                  i = self.items.index(key)
                  del self.items[i]
                  del self.map[key]
                  for k, v in self.map.items():
                        if v >= i:
                              self.map[k] = v - 1

      def clear(self):
            del self.items[:]
            self.map.clear()

      def __iter__(self):
            return iter(self.items)

      def __reversed__(self):
            return reversed(self.items)


      def __repr__(self):
            if not self:
                  return '%s()' % (self.__class__.__name__,)
            return '%s(%r)' % (self.__class__.__name__, list(self))
```

```
      def __eq__(self, other):
            if isinstance(other, OrderedSet):
                  return len(self) == len(other) and self.items ==
other.items
            try:
                  other_as_set = set(other)
            except TypeError:
                  return False
            else:
                  return set(self) == other_as_set
```

4.2.3 Scripts

Todos los scripts constan del mismo formato:

- Importamos librerías.
- Conectamos a la base de datos.
- Definimos variables y extraemos los datos a la base de datos.
- Cerramos conexiones.

En la **importación de librerías** utilizaremos las previamente mencionadas en el apartado 4.2.2.

Conexión a la base de datos:

```
cnxn = pyodbc.connect('DRIVER={SQL Server Native Client
10.0};SERVER=KOSMOS;DATABASE=[nombre_base_de_datos];UID=[usuar
io];PWD=[contraseña]')

cursor = cnxn.cursor()
```

Variables importantes*:

- Client: Siempre Venca.
- Brand: Marca que estamos rastreando.
- urls: Será el conjunto de escaparates que rastrearemos.
- main_url: Guardaremos el escaparate que estamos rastreando actualmente, para poder saber en qué escaparate estaba cada artículo y cuales hemos rastreado finalmente.
- linkP: Link del producto.
- timeP/timeIni/time_error: Variables de tiempo para saber cuándo hemos ejecutado el script y de que fechas son los datos, así como también de posibles errores.
- categoría: Categoría general a la que según la web está relacionado el articulo (p.e. Vestidos).
- subcategoria: Subcategoría más precisa del articulo (p.e. Vestidos de fiesta).
- nombreP: Nombre del producto.
- precioP: Precio actual y de venta del producto.

- precioRealP: Precio real del producto (solo será diferente en caso de descuento, siendo mayor).
- codigoP: Código del producto, que lo hace único en la web.
- ncolores: Número de colores que tiene el producto.
- tallaMin: Talla mínima del artículo.
- tallaMax: Talla máxima del artículo.

**los nombres exactos pueden variar en cada script, pero la información extraída es siempre la misma.*

Inserción datos extraídos en BBDD de SQL:

```
var_sql = """insert into crawlerVenca
(Client,Category,Brand,SubBrand,Description,PriceBeforeDiscount,ActualPr
ice,URL,Link,#Colors,MinSize,MaxSize,CodeProd,ID_Presentation,Execution_
Time,Top20,Crawl_Day, Crawl_Date) values
(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"""
cursor.execute(var_sql,
(client,categoria,brand,subcategoria,nombreP,precioRealP,precioP,main_ur
l,linkP,ncolores,tallaMin,tallaMax,codigoP,current_cont,fecha_ejecucion,
top,timeIni,timeP))
cursor.commit()
```

La gestión de errores en los scripts es casi imposible, ya que las webs están en constante cambio y ellas mismas tienen links caídos que se pueden rastrear y que dan un error al entrar a ellos. Aun así, se ha hecho una pequeña gestión de ellos, en la cual cada script por sí solo genera un informe .txt y cada vez que hay un error escribe el link que ha fallado y el porqué. En caso de detectar un error grave, por ejemplo, en número de artículos rastreados en un día, este archivo nos ayudará a saber que ha pasado.

El proyecto inicial constaba de un total de 7 tiendas (+Venca) a analizar: C&A, H&M, Kiabi, Mango, Shana, Stradivarius y Zara. Al obtener los primeros esbozos y resultados, no se tardó en pedir más tiendas hasta hacer un total de 11; Bershka, Blanco y Lefties.

Todos los scripts se adjuntan en el apartado 9. Anexos.

A continuación, se muestra como ejemplo el de Venca.

4.2.3.1 Venca

```
# -*- coding: utf-8 -*-
"""

@author: joelsanchez
"""

import sys
import time
import urllib
import pyodbc
import requests
from ordered_set import *
from selenium import webdriver
from bs4 import BeautifulSoup
from datetime import datetime
pmName = 'ordered_set'
pm = __import__(pmName)


headers = {'User-agent': 'Mozilla/5.0'}

cnxn = pyodbc.connect('DRIVER={SQL Server Native Client
10.0};SERVER=KOSMOS;DATABASE=MKT_CRAWLER;UID=[usuario];PWD=[contraseña]')

cursor = cnxn.cursor()
fecha_error = time.strftime("%Y%m%d")
urls={'http://www.venca.es/e/34/look-casual',
'http://www.venca.es/e/98/rebajas-vestidos-y-monos',
'http://www.venca.es/e/97/rebajas-jerseis',
'http://www.venca.es/e/21/vestidos',
'http://www.venca.es/e/51/superprecios',
'http://www.venca.es/e/11/tendencia-verano-1',
'http://www.venca.es/e/73/jerseis-y-chaquetas-tallas-grandes',
'http://www.venca.es/e/99/rebajas-faldas-y-shorts',
'http://www.venca.es/e/78/moda-tallas-grandes'}

url = "http://www.venca.es"
fecha_ejecucion = time.strftime("%Y%m%d %H:%M:%S")
visited = set()
fallos = set()
#sinDuplicados = set()
cont = 0
topMax = 20
client = 'Venca'
brand = 'Venca'
filename= "S:/Consultas_python/CrawlerVenca/Errores/" + fecha_error + "_"
+ brand + "_Errors.txt"


timeIni = datetime.now()
final = 0
```

```
driver = webdriver.Chrome()#'C:/chromedriver.exe'

escaparates = len(urls)
contE = 0
#mientras queden escaparates, seguimos en el bucle
while urls:

    #sacamos el primer escaparate
    current_url = urls.pop()
    #nos guardamos el link del escaparate para añadirlo a SQL al final
    main_url = current_url
    print "Crawling current_url : %s"%(current_url)
    #abrimos Firefox, vamos a la url del escaparate y hacemos 40 scrolls
hacia abajo, con esto llegaremos al final de todos los escaparates y
cogeremos el codigo HTML final con todos los productos
    try:
        driver.get(current_url);

        lastHeight = driver.execute_script("return
document.body.scrollHeight")
        while True:
            driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")
            time.sleep(1)
            newHeight = driver.execute_script("return
document.body.scrollHeight")
            if newHeight == lastHeight:
                break
            lastHeight = newHeight
        html2 = driver.execute_script("return
document.documentElement.innerHTML;")
    except Exception:
        driver.quit()
        sys.exit()


    soup= BeautifulSoup(html2)
    word = '/p/'
    sinDuplicados = OrderedSet()
    sinDuplicados.clear()
    contador = OrderedSet()
    contador.clear()
    #sacamos todos los links de productos del escaparate y los añadimos a un
set
    for tag in soup.findAll('a', href=True):
        if word in tag['href'] and 'http' not in tag['href']:
            new_url = url + tag['href']
            sinDuplicados.add(new_url)
            cont +=1
            contador.add(cont)
    cont = 0
    current_cont = len(sinDuplicados)
    productos = len(sinDuplicados)
    contP = 0
```

```
#bucle para recorrer todos los productos que hemos sacado del escaparate
anterior
    while sinDuplicados:

        try:
            #sacamos el primer producto
            current_url = sinDuplicados.pop()
            print "Crawling", main_url
            #comprobamos que efectivamente la URL sea de produto

            timeP = datetime.now()
            #current_cont = contador.pop()
            if current_cont <= topMax:
                top = 'TRUE'
            else:
                top = 'FALSE'

            print "Parsing", current_url
            #cogemos el codigo HTML de la pagina de producto
            webpage = requests.get(current_url, headers=headers)
            htmltext = webpage.content
            soup= BeautifulSoup(htmltext)
            #Sacamos categoría, subcategoria y nombre
            categorias = soup.find('div', { 'class' : "container
antiVoffset3" })
            categorias2 = categorias.findNext('a', href="/")
            categoria = categorias.findNext('a', href="/").getText()
            categoria = categoria.replace("\r","")
            categoria = categoria.replace("\n","")
            subcategoria2 = categorias2.findNext('a', href=True)
            subcategoria = categorias2.findNext('a', href=True).getText()
            subcategoria = subcategoria.replace("\r","")
            subcategoria = subcategoria.replace("\n","")
            nombreP = soup.find_all('meta', { 'property' :
"og:title"})[0]['content']

            #sacamos precio (precioP) y precio sin descontar (precioRealP)
            precioPentero = soup.find('span',
id="integerPricePart").getText()
            precioPdecimal = soup.find('sup',
id="decimalPricePart").getText()
            precioP = precioPentero + "." + precioPdecimal
            precioP = precioP.encode('utf-8')
            precioP = precioP[:precioP.index('€')]
            precioRealP = soup.find('li',{ 'class' : "col-xs-6 col-sm-6 col-
md-6 col-lg-6 text-right smallPrice"})
            if precioRealP == None:
                precioRealP = precioP
            else:
                precioRealP = soup.find('li',{ 'class' : "col-xs-6 col-sm-6
col-md-6 col-lg-6 text-right smallPrice"}).getText()
                precioRealP = precioRealP.replace("\r","")
                precioRealP = precioRealP.replace("\n","")
                precioRealP = precioRealP.replace(",",".")
                precioRealP = precioRealP.encode('utf-8')
                precioRealP = precioRealP[:precioRealP.index('€')]
```

```
#link directo al producto
            linkP = current_url

            try:
                codigoP = soup.find_all('input', { 'id' :
"modelId"})[0]['value']
            except IndexError:
                list_html = htmltext.split(':')
                codigoP = list_html[list_html.index('"Product","productID"')
+ 1]
                codigoP = codigoP.split(',')
                codigoP = codigoP[0]
                codigoP = codigoP.replace('"',"")


            #novedades = seccion propia

            word = "btn btn-default colorPicker"
            ncolores = htmltext.count(word)

            try:
                tallaMin = soup.find_all('a',tabindex="1")[0]['data-id-for-
select']


                tallaMax = soup.find_all('a', tabindex="1")
                last_div = None
                for last_div in tallaMax:pass #bucle para recorrer todas las
tallas y coger la ultima
                if last_div:
                    tallaMax = last_div.getText()
                    tallaMax = tallaMax[:tallaMax.index(" ")]
            except Exception:

                tallaMin = soup.find('p', {'class' :
"paddingSoldOut"}).getText()
                tallaMin = tallaMin[:tallaMin.index(" ")]


                tallaMax = soup.find_all('p',  {'class' : "paddingSoldOut"})
                last_div = None
                for last_div in tallaMax:pass #bucle para recorrer todas las
tallas y coger la ultima
                if last_div:
                    tallaMax = last_div.getText()
                    tallaMax = tallaMax[:tallaMax.index(" ")]
```

```
var_sql = """insert into crawlerVenca
(Client,Category,Brand,SubBrand,Description,PriceBeforeDiscount,ActualPrice,
URL,Link,#Colors,MinSize,MaxSize,CodeProd,ID_Presentation,Execution_Time,Top
20,Crawl_Day, Crawl_Date) values (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"""
            cursor.execute(var_sql,
(client,categoria,brand,subcategoria,nombreP,precioRealP,precioP,main_url,li
nkP,ncolores,tallaMin,tallaMax,codigoP,current_cont,fecha_ejecucion,top,time
Ini,timeP))
            cursor.commit()
            try:
                print "Categoria: %s - Subcategoria: %s - Nombre: %s -
Precio: %s - Precio sin descontar: %s - Link: %s - Codigo: %s - Color: %s -
Talla minima: %s - Talla
maxima: %s"%(categoria,subcategoria,nombreP,precioP,precioRealP,linkP,codigo
P,ncolores, tallaMin, tallaMax)
            except Exception:
                pass
            print "current cont %s/%s"%(current_cont,productos)

        except Exception,e:
            f=open(filename,"a")
            #write error to file:
            f.write("Error occured:\n" + current_url + " <------> " + str(e)
+ "\n") # some message specific to this iteration (page) should be added
here...
            #close error log file:
            f.close()
            fallos.add(current_url)


        current_cont += -1
        contP += 1
        print "Productos %s/%s"%(contP,productos)
        if contP == productos:
            final += 1
            print "Escaparate crawleado con exito"



    contE += 1
    print "Crawleado escaparate %s/%s"%(contE,escaparates)
    print "Con exito: %s/%s"%(final,escaparates)

print "Crawler de %s ha finalizado correctamente"%(brand)
print "Lista de fallos: %s"%(fallos)
driver.quit()
cursor.close()
```

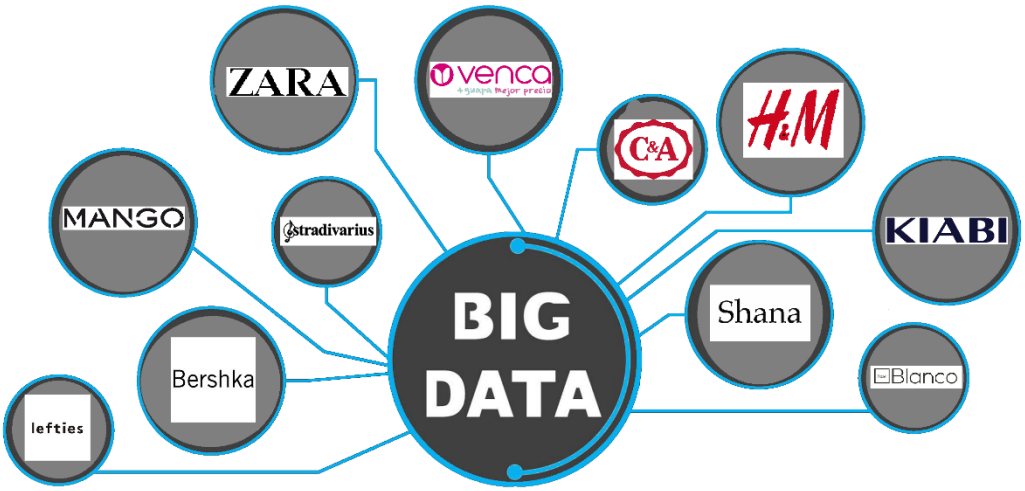
4.2.3.2 Y muchos más…

A medida que avanzaba el proyecto y empezaban a aparecer resultados, me fueron pidiendo más y más tipos de páginas para crawlear y las cuales han ido creando “clusters” y otros tipos de informes, aquí un listado de ellas:

- H&M Hombre
- Kiabi Hombre
- Venca Hombre
- H&M Niños (niñas, niños y bebés)
- Kiabi Niños
- Venca Niños
- Bonprix
- Detector de productos duplicados en Venca
- 3suisses Francia
- C&A Francia
- Camaieu Francia
- Etam Francia
- Gemo Francia
- H&M Francia
- Kiabi Francia
- LaHalle Francia
- Mango Francia
- Promod Francia
- Zara Francia
- Douglas Perfumería
- Druni Perfumería
- ElCorteIngles Perfumería
- Julia Perfumería
- PerfumesClub Perfumería
- PefumesIF Perfumería
- Asos TallasGrandes
- C&A TallasGrandes
- ElCorteInlges TallasGrandes
- H&M TallasGrandes
- Kiabi TallasGrandes
- LaRedoute TallasGrandes
- Mango TallasGrandes
- Msmode TallasGrandes
- Venca TallasGrandes
- Zalando TallasGrandes
- Aemet, para extraer tiempo que iba hacer

4.3 Incorporación de los datos

Los scripts van a generar una cantidad inmensa de datos cada vez que se ejecuten, para los cuales tenemos que crear una muy buena estructura de base de datos para poder analizarlos y consultarlos de una manera rápida e intuitiva.



Gracias a esto, podemos obtener datos de manera diaria/semanal/mensual, que ayudaran al departamento de compras y a la empresa a enriquecer sus bases de datos y a tomar decisiones basándose en la competencia, ayudándonos a definir mejor nuestras campañas, nuestros precios y a tener una visión general de Venca en comparación al resto.

Al fin y al cabo, Venca, al igual que cualquier otra empresa, intenta llegar a todos sus clientes por Internet intentando darles la mejor oferta posible y el mejor producto posible; un producto que se lleve en este momento y un precio competitivo en el mercado, y qué mejor manera de lograr esto que pudiendo analizar cómo lo hacen grandes y pequeñas empresas con las cuales competimos directamente.

Antes de ponernos con ello, tenemos que hacer un análisis y previsión de los datos que vamos a tener, volumen, información que nos van a aportar y de qué manera los queremos extraer para analizar finalmente, al igual que las veces que rellenaremos las tablas.

4.3.1 Análisis previo

El objetivo principal está claro: analizar 11 tiendas, sección de mujer. Pero no tenemos únicamente esto, sino también tiendas francesas (sección mujer) y varias tiendas que analizarán, aparte de la sección de mujer; también las de hombre, niño, niña y bebés.

Para enfrentarnos a este volumen de datos, al principio del proyecto opté por crear una tabla principal donde añadí el objetivo principal del proyecto, que era el análisis de las 11 tiendas pedidas, y puse también una variable país y añadí Francia. Para el resto creé una tabla independiente para cada uno, igualmente con la variable país por si se daba el caso que analizábamos estas ramas también fuera de España.

Pros:

- Tenía toda la información principal en una tabla, incluida Francia, lo que me permitía atacar solo una tabla (consultas, procesos, scripts…) y poder hacer análisis de precios entre países directamente desde la misma.
- Tener separado mujer, hombre, niño, niña y bebés y, por lo tanto, poder distribuir el espacio en varias tablas, lo que equivale a consultas más rápidas y precisas.
- Informes y scripts más directos, al atacar cada uno a la tabla que le toca.

Contras:

- En el caso de juntar España y Francia, gran volumen de datos, lo que ralentizaba las consultas.
- Información más dispersa, por lo que había que atacar más de una tabla en el caso de querer datos diferentes.
- Posibilidad de generar datos erróneos si, por ejemplo, no se filtra por país en la tabla que tiene datos de varios de éstos.

Al final, debido al problema del gran volumen de datos, creé una estructura con su tabla y sus procesos para Francia y lo separé de la tabla de España, ya que ralentizaban considerablemente las consultas.

En la decisión de la estructura final también influyó, por no decir que decidió todo, el número de ejecuciones de los scripts, optando finalmente y tras probar la ejecución diaria, por una ejecución semanal. La ejecución semanal nos daba un

volumen de datos más moderado y una velocidad más que aceptable para las consultas.

Análisis que detallo a continuación:

- **Diario**:
  - Pros:
    - Capacidad de detectar novedades, ofertas, descuentos y promociones diarias de cualquier marca.
    - Capacidad de reacción a las pocas horas en caso de gran cambio en la competencia.
    - Tener una vista general del mercado al día.
  - Contras:
    - Volumen incontrolable de datos.
    - Ralentización en todos los procesos de consulta debido al gran volumen de datos.
    - Hay que disponer de un tiempo diario para generar los reportes y comprobar que todo está correcto.

- **Semanal**:
  - Pros:
    - Generación de datos moderada y velocidad de las consultas más que aceptable.
    - Visión general del mercado semanal.
    - Capacidad de analizar novedades, ofertas, etc. Semanalmente.
    - Solo necesitas dedicar un día a la semana para la generación de los informes.
  - Contras:
    - Pierdes la opción del análisis y respuesta diaria.
- **Mensual**:
  - Pros:
    - Dispondría de las consultas más rápidas ya que el volumen de datos sería el menor de los 3.
    - En caso de que a la empresa así le interese, disponer de un análisis de mercado mensual.
    - Solo debes preocuparte una vez al mes de la generación de informes.

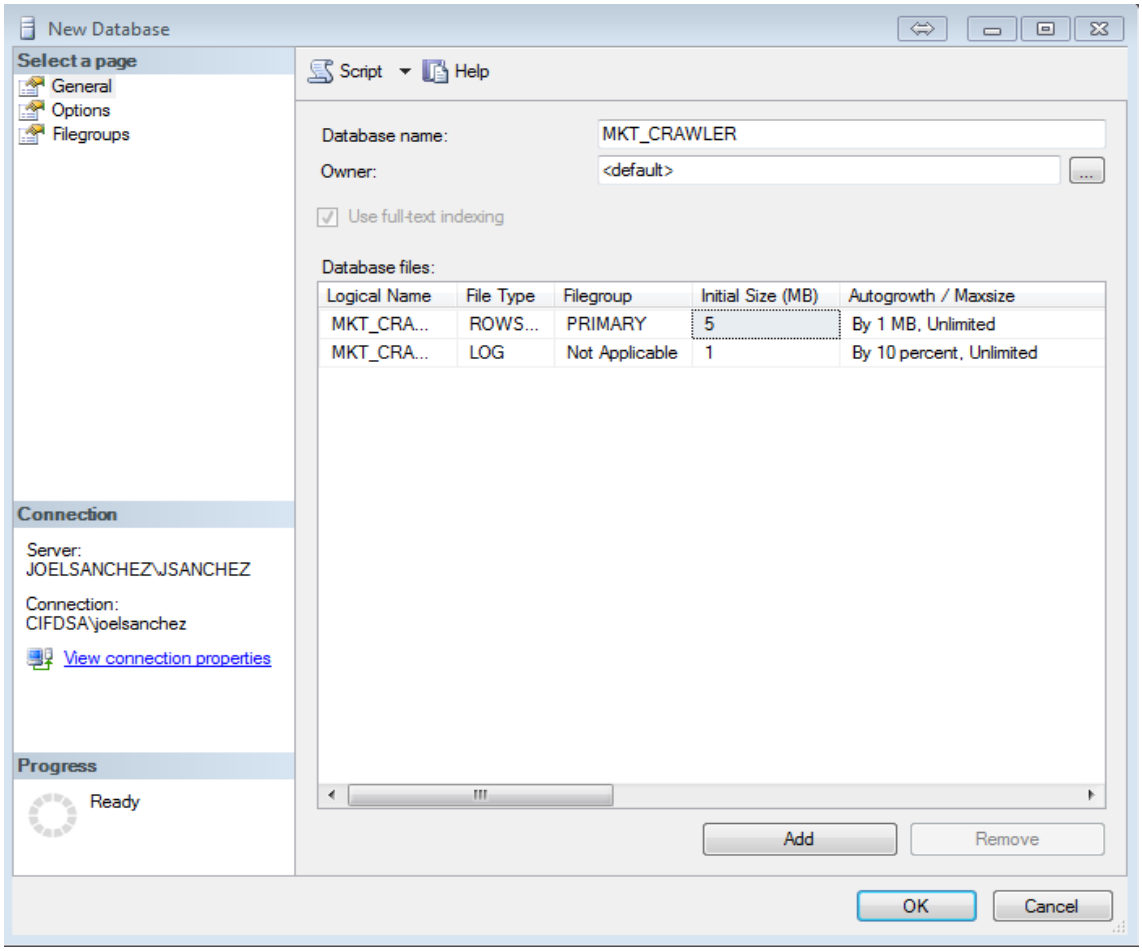
- Contras:
    - En el caso del análisis, es el “menos preciso” y el que peor opción a respuesta nos da.
    - Posible pérdida de mucha información durante un mes sobre la competencia.
    - Posible cambio en los códigos fuentes de las webs y, por lo tanto, fallo en los scripts asegurado. (Este estaría siempre, pero a diario lo detectas y lo corriges. Semanalmente también lo detectas y lo corriges de una semana para otra, y das menos margen en ambos casos a las páginas a cambiar su código).

4.3.2 Generación de tablas

4.3.2.1 Creación de la base de datos

En este caso, el servidor nos viene dado por la empresa, aun así, pudiendo ser un proyecto aplicado a cualquier empresa o particular, en el apartado 6.2 Coste económico adjunto también un estudio para la decisión a tomar por tal de elegir un tipo de servidor u otro y, para la creación del mismo seguiríamos cualquier tutorial que nos ofrecieran Google o el propio servidor en caso de ser en Cloud.

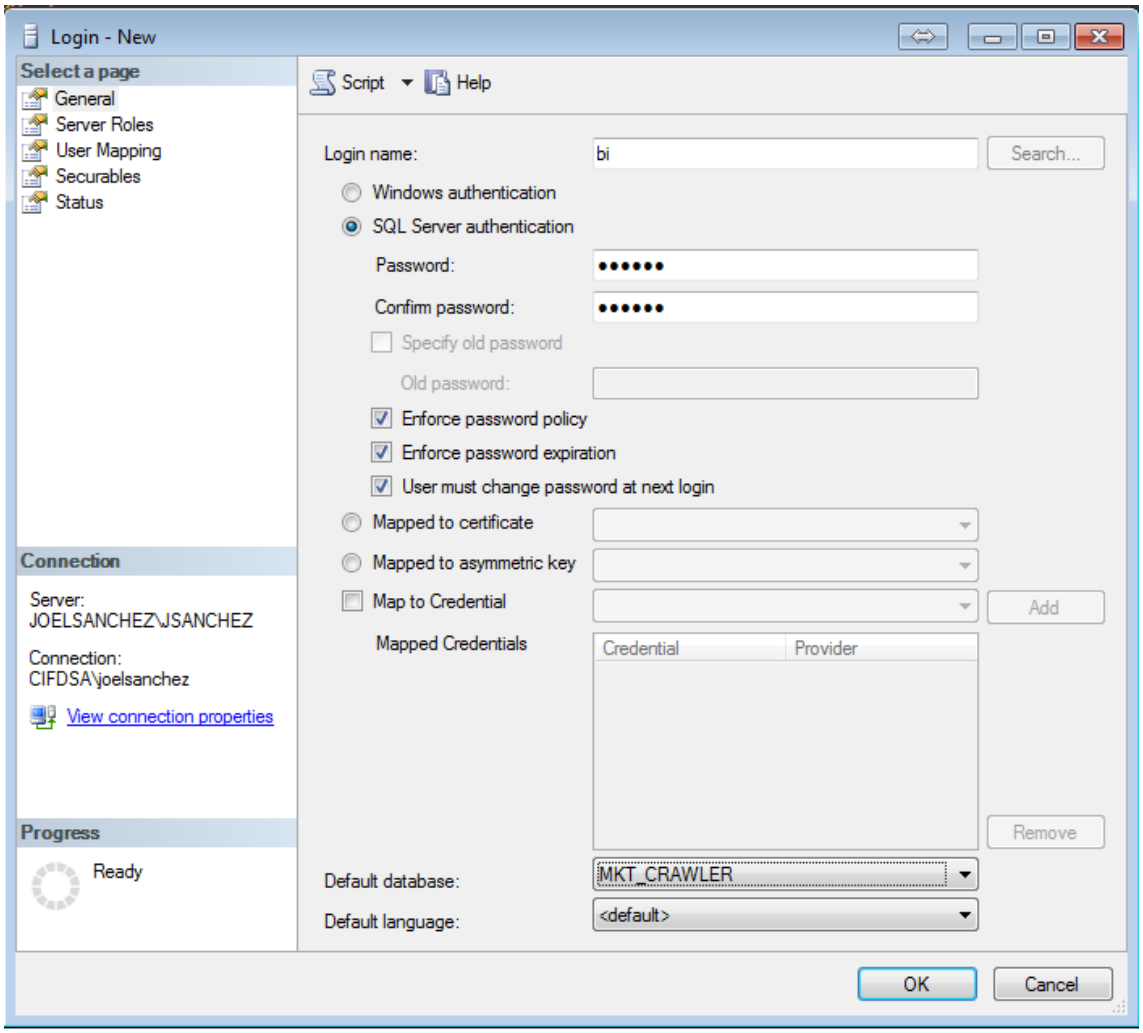
Para crear la base de datos, una vez abierta la interfaz de usuario de SQL y logueados en nuestro servidor, vamos a *Databases* > *New Database*…



Lo único que haremos es ponerle un nombre y darle a *"OK"*. La base de datos irá escalando automáticamente conforme entren los datos, para esto obviamente tenemos que tener conciencia de los datos que vamos a escalar y, asegurar que vamos a tener tanto espacio como poder de procesamiento para ellos. En este caso, lo asegura Venca conjuntamente conmigo, tras ver lo que iban ocupando las ejecuciones.

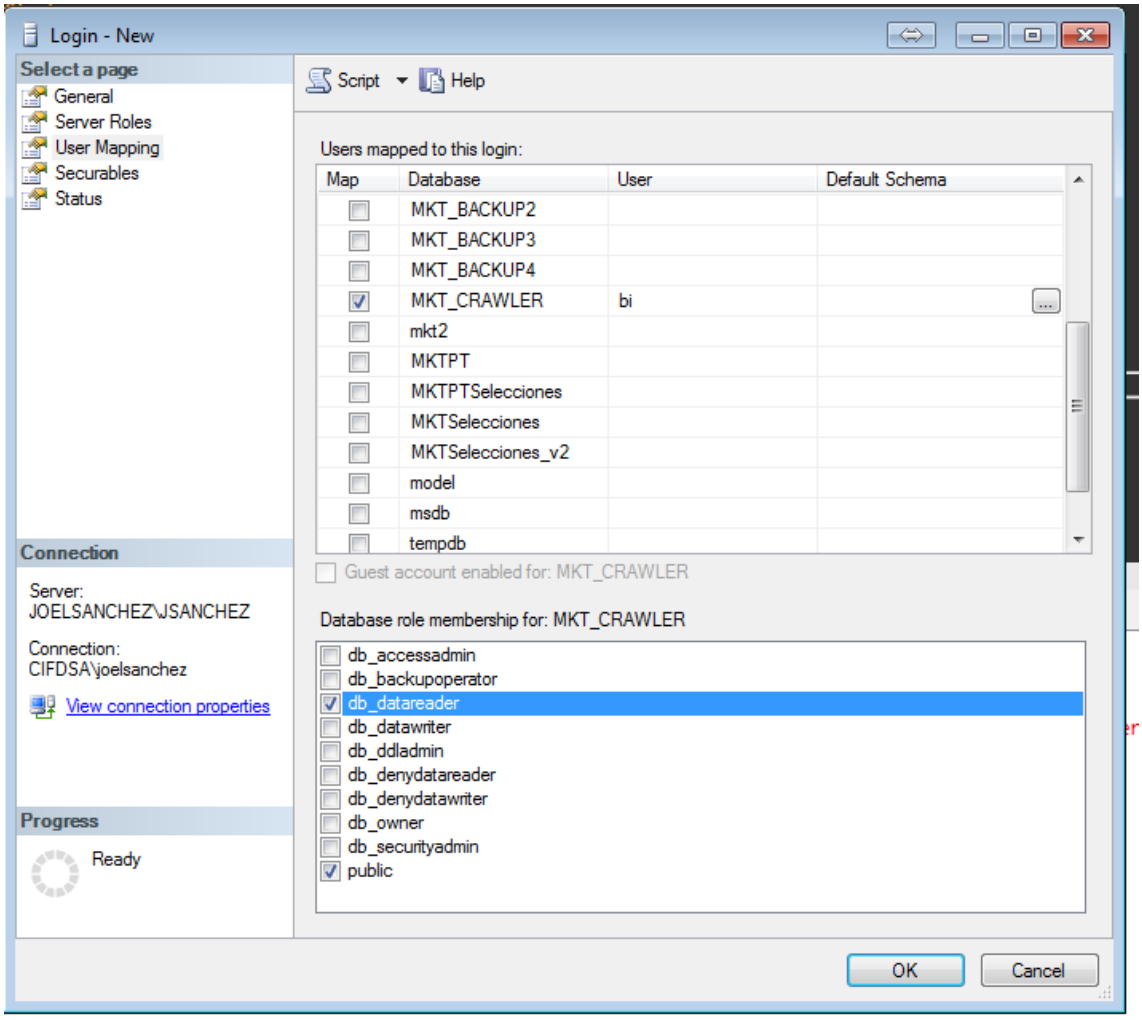
Tras haber generado la base de datos, nos quedaría generar los usuarios. Podríamos pensar en principio dos: el owner y el de usuario. Uno con acceso a todo y con todo tipo de permisos para poder escribir desde los scripts, leer, crear, borrar, etc. Y otro para el uso simplemente de lectura, ya sea tanto en desde Excel como desde el propio SQL.

Para crear un usuario nos vamos a *Security* > *Logins* > *New Login*…



Definimos un nombre, una contraseña y una base de datos por defecto en caso de que solo queramos que tenga acceso a una.

Y, por último, en *"User mapping",* definimos el rol que le queremos dar y a que bases de datos se lo queremos dar*:*



En este caso, de lectura y solo a MKT_CRAWLER.

También tenemos otros roles de solo escritura, owner, admin, back up, etc.

Una vez tenemos generada nuestra base de datos y nuestros usuarios, es hora de crear las tablas.

4.3.2.2 Generando las tablas

1. Formato tabla principal:

Nombre	Tipo	Longitud	Descripción
Client	Varchar	50	Cliente por el cual estamos ejecutando los scripts: Venca
Category	Varchar	500	Principal categoría del producto
Brand	Varchar	500	Marca que estamos rastreando
SubBrand	Varchar	500	Subcategoría a la que pertenece el producto
URL	Varchar	1000	URL del escaparate
Link	Varchar	1000	Link del producto
NEW	Varchar	1000	Marca para saber si es novedad
Description	Varchar	1000	Nombre del producto
Comment	Varchar	1000	Posible comentario que añadir sobre un producto
PriceBeforeDiscount	Float		Precio del producto antes de un posible descuento
ActualPrice	Decimal	16,2	Precio actual del producto
Top20	Varchar	500	Marca para saber si pertenece al top 20 de productos dentro de su escaparate
Crawl_Day	Datetime		Dia de ejecución
Crawl_Date	Datetime		Fecha completa de ejecución del script
Id_Presentation	Bigint		Número de posición dentro del escaparate del producto
#Colors	Int	15	Número de colores del producto
CodeProd	Bigint		Código del producto
MinSize	Nvarchar	1000	Talla mínima
MaxSize	Nvarchar	1000	Talla máxima
Execution_Time	Nvarchar	1000	Fecha de ejecución del producto en concreto
Linea_2	Varchar	50	Categoría “Venca” del producto
Cuartil	Bigint		Cuartil del producto, según su precio
Fecha	Smalldatetime		Fecha en formato “aaaammdd” de la ejecución
Dia	Int		Dia de la ejecución
Mes	Int		Mes de la ejecución
Año	Int		Año de la ejecución
Sem	Int		Semana de la ejecución
Dia_sem	Int		Dia de la semana de la ejecución
Id_num	Int		ID único e incremental de los datos de la tabla

2. Creación tabla principal:

```
USE [MKT_CRAWLER]
GO

/****** Object:  Table [dbo].[crawlervenca]  ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[crawlervenca](
	[Client] [varchar](50) NULL,
	[Category] [varchar](500) NULL,
	[Brand] [varchar](500) NULL,
	[SubBrand] [varchar](500) NULL,
	[URL] [varchar](1000) NULL,
	[Link] [varchar](1000) NULL,
	[NEW] [varchar](1000) NULL,
	[Description] [varchar](1000) NULL,
	[Comment] [varchar](1000) NULL,
	[PriceBeforeDiscount] [float] NULL,
	[ActualPrice] [decimal](16, 2) NULL,
	[Top20] [varchar](500) NULL,
	[Crawl_Day] [datetime] NULL,
	[Crawl_Date] [datetime] NULL,
	[ID_Presentation] [bigint] NULL,
	[#Colors] [int] NULL,
	[CodeProd] [bigint] NULL,
	[MinSize] [nvarchar](1000) NULL,
	[MaxSize] [nvarchar](1000) NULL,
	[Execution_Time] [nvarchar](1000) NULL,
	[linea_2] [varchar](50) NULL,
	[cuartil] [bigint] NULL,
	[fecha] [smalldatetime] NULL,
	[dia] [int] NULL,
	[mes] [int] NULL,
	[año] [int] NULL,
	[sem] [int] NULL,
	[dia_sem] [int] NULL,
	[id_num] [int] IDENTITY(1,1) NOT NULL
) ON [PRIMARY]

GO
```

3. Tabla novedades:

```
USE [MKT_CRAWLER]
GO

/****** Object:  Table [dbo].[crawlerNovedades]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[crawlerNovedades](
	[client] [varchar](50) NULL,
	[brand] [varchar](500) NULL,
	[category] [varchar](500) NULL,
	[url] [varchar](1000) NULL,
	[link] [varchar](1000) NULL,
	[description] [varchar](1000) NULL,
	[new] [varchar](1000) NULL,
	[PriceBeforeDiscount] [float] NULL,
	[ActualPrice] [decimal](16, 2) NULL,
	[ID_Presentation] [bigint] NULL,
	[Top20] [varchar](500) NULL,
	[Crawl_Day] [datetime] NULL,
	[Crawl_Date] [datetime] NULL,
	[#Colors] [int] NULL,
	[CodeProd] [bigint] NULL,
	[MinSize] [nvarchar](1000) NULL,
	[MaxSize] [nvarchar](1000) NULL,
	[Execution_Time] [nvarchar](1000) NULL,
	[linea_2] [varchar](50) NULL,
	[cuartil] [bigint] NULL,
	[fecha] [smalldatetime] NULL,
	[brcodeprod] [nvarchar](516) NULL,
	[dia] [int] NULL,
	[mes] [int] NULL,
	[año] [int] NULL,
	[sem] [int] NULL,
	[dia_sem] [int] NULL,
	[longDescription] [nvarchar](1000) NULL
) ON [PRIMARY]

GO
```

***** Añadimos brcodeprod, que no es más que los campos Brand y CodeProd unidos por tal del correcto funcionamiento del Excel de novedades (p.e. “Venca_154588”), y longDescription, que equivale al campo “comment” de la tabla principal.

4. Tabla SuperOfertas:

```
USE [MKT_CRAWLER]
GO

/****** Object:  Table [dbo].[crawlerSuperOfertas]    Script Date:
13/01/2018 17:41:22 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[crawlerSuperOfertas](
	[client] [varchar](50) NULL,
	[brand] [varchar](500) NULL,
	[category] [varchar](500) NULL,
	[url] [varchar](1000) NULL,
	[link] [varchar](1000) NULL,
	[description] [varchar](1000) NULL,
	[new] [varchar](1000) NULL,
	[PriceBeforeDiscount] [float] NULL,
	[ActualPrice] [decimal](16, 2) NULL,
	[ID_Presentation] [bigint] NULL,
	[Top20] [varchar](500) NULL,
	[Crawl_Day] [datetime] NULL,
	[Crawl_Date] [datetime] NULL,
	[#Colors] [int] NULL,
	[CodeProd] [bigint] NULL,
	[MinSize] [nvarchar](1000) NULL,
	[MaxSize] [nvarchar](1000) NULL,
	[Execution_Time] [nvarchar](1000) NULL,
	[linea_2] [varchar](50) NULL,
	[cuartil] [bigint] NULL,
	[fecha] [smalldatetime] NULL,
	[descuento] [float] NULL,
	[brcodeprod] [nvarchar](516) NULL,
	[dia] [int] NULL,
	[mes] [int] NULL,
	[año] [int] NULL,
	[sem] [int] NULL,
	[dia_sem] [int] NULL,
	[tipo_descuento] [nvarchar](20) NULL,
	[novedad] [int] NULL
) ON [PRIMARY]

GO
```

* Añadimos los campos descuento, donde incluiremos el descuento en forma de Número “0,55”, el campo tipo_descuento, que nos dirá en que franja de descuento se encuentra (>70%, 50-70, 0,1-10., etc.) y novedad, que nos dirá si además de tener descuento es un producto nuevo.

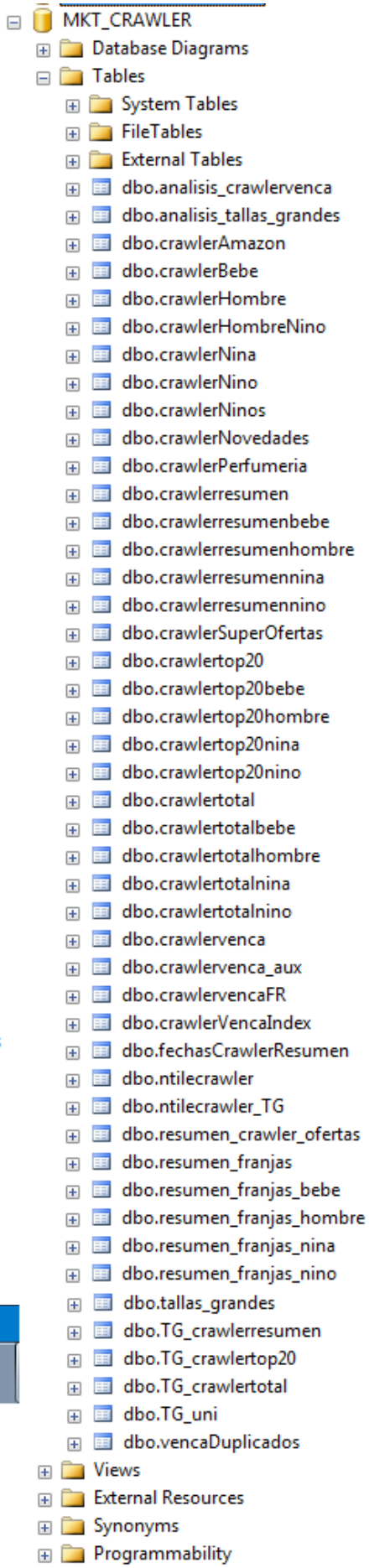
5. Tabla resumen final:

Nombre	Tipo	Longitud	Descripción
Brand	Varchar	500	Marca que hemos rastreado
Categoría	Varchar	50	Categoría “Venca” del producto
Fecha	Nvarchar	8	Fecha en formato “aaaammdd” de la ejecución
nModelos	Int		Número de modelos por categoría
PrecioMedio	Decimal	38,6	Precio medio de producto por categoría
PrecioMinimo	Decimal	16,2	Precio mínimo de producto por categoría
Q1	Decimal	38,6	Número de modelos en el primer cuartil
Q2	Decimal	38,6	Número de modelos en el segundo cuartil
Mediana	Decimal	21,6	Mediana en precios de producto por categoría
Q3	Decimal	38,6	Número de modelos en el tercer cuartil
Q4	Decimal	38,6	Número de modelos en el cuarto cuartil
PrecioMax	Decimal	16,2	Precio máximo de producto por categoría
Franja1	Int		Número de modelos en la franja 1 de precios
Franja2	Int		Número de modelos en la franja 2 de precios
Franja3	Int		Número de modelos en la franja 3 de precios
Franja4	Int		Número de modelos en la franja 4 de precios
Franja5	Int		Número de modelos en la franja 5 de precios
Franja6	Int		Número de modelos en la franja 6 de precios
Franja7	Int		Número de modelos en la franja 7 de precios
Tipo_top	Varchar	5	Producto general o top 20
Dia	Int		Dia de la ejecución
Mes	Int		Mes de la ejecución
Año	Int		Año de la ejecución
Sem	Int		Semana de la ejecución
Dia_sem	Int		Dia de la semana de la ejecución

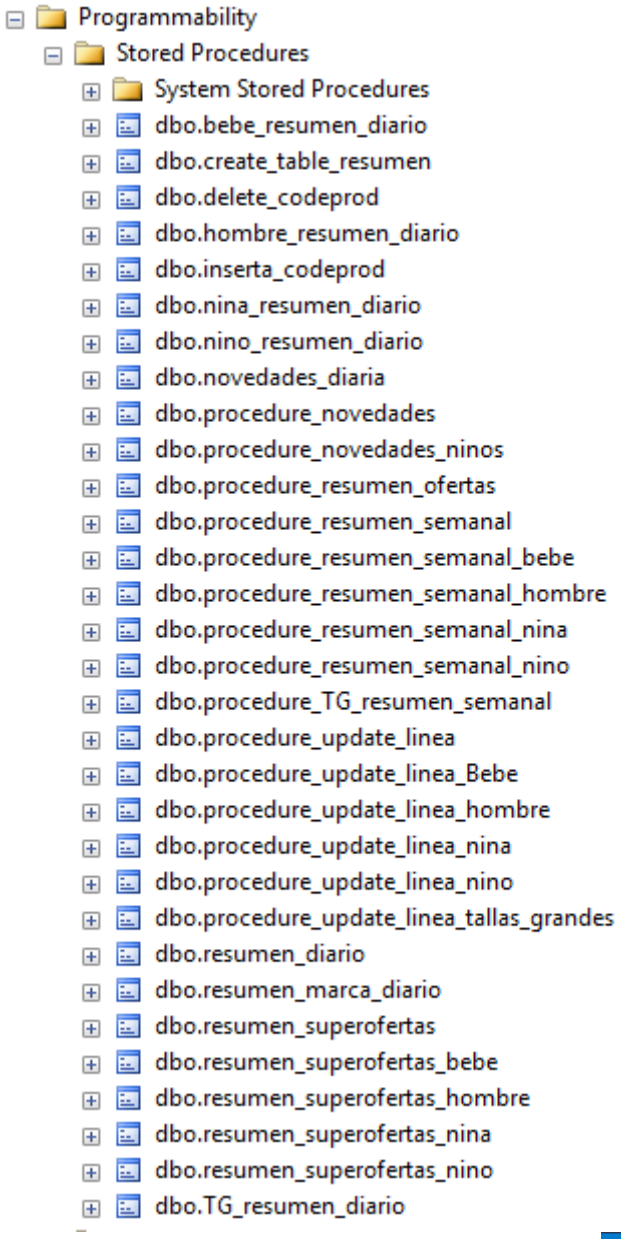
```
USE [MKT_CRAWLER]
GO

/****** Object:  Table [dbo].[crawlerresumen]    Script Date:
14/01/2018 18:05:10 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[crawlerresumen](
	[brand] [varchar](500) NULL,
	[Categoria] [varchar](50) NULL,
	[Fecha] [nvarchar](8) NULL,
	[nModelos] [int] NULL,
	[PrecioMedio] [decimal](38, 6) NULL,
	[PrecioMinimo] [decimal](16, 2) NULL,
	[Q1] [decimal](38, 6) NULL,
	[Q2] [decimal](38, 6) NULL,
	[Mediana] [decimal](21, 6) NULL,
	[Q3] [decimal](38, 6) NULL,
	[Q4] [decimal](38, 6) NULL,
	[PrecioMax] [decimal](16, 2) NULL,
	[Franja1] [int] NULL,
	[Franja2] [int] NULL,
	[Franja3] [int] NULL,
	[Franja4] [int] NULL,
	[Franja5] [int] NULL,
	[Franja6] [int] NULL,
	[Franja7] [int] NULL,
	[tipo_top] [varchar](5) NOT NULL,
	[dia] [int] NULL,
	[mes] [int] NULL,
	[año] [int] NULL,
	[sem] [int] NULL,
	[dia_sem] [int] NULL
) ON [PRIMARY]

GO
```

Estructura real de la base de datos MKT_CRAWLER:

Tablas → → → → → → →



Stored Procedures:



4.3.2 Procedures

4.3.2.1 Entendiendo el funcionamiento

Para entender correctamente el funcionamiento de los siguientes procedimientos, hay que entender la estructura en la que nos encontramos y lo que queremos conseguir finalmente. Nos centraremos en las tablas principales *crawlervenca*, *crawlerNovedades*, *crawlerSuperOfertas* y *crawlerresumen*, vistas previamente, pero no debemos olvidar que aparte de éstas existen tablas tanto para niño, niña, bebés, Francia, tallas grandes y perfumería, las cuales tienen la misma estructura que las generales y por lo tanto siguen los mismos procedimientos. Así que todo lo mostrado de ahora en adelante, hay que pensar que está hecho tantas veces como tablas diferentes tengamos.

La ejecución semanal para actualizar los datos sigue la siguiente estructura:

```
use mkt_crawler
/*--------------------------------------------------------------------*/
/*-------------- update_linea/categoría mujer  ----------------------*/
/*--------------------------------------------------------------------*/

declare @fecha nvarchar(8)
set @fecha = convert(nvarchar(8), getdate(),112)
exec [dbo].[procedure_update_linea] @fecha   --ok
go
/*--------------------------------------------------------------------*/
/*----------------------- novedades mujer ----------------------------*/
/*--------------------------------------------------------------------*/

declare @fecha nvarchar(8)
set @fecha = convert(nvarchar(8), getdate(),112)
exec [dbo].[procedure_novedades] @fecha
go
/*--------------------------------------------------------------------*/
/*------------------------ ofertas mujer -----------------------------*/
/*--------------------------------------------------------------------*/

declare @fecha nvarchar(8)
set @fecha = convert(nvarchar(8), getdate(),112)
exec [dbo].[resumen_superofertas] @fecha
exec [dbo].[procedure_resumen_ofertas] @fecha

go

/*--------------------------------------------------------------------*/
/*------------------- resumen semanal mujer ------------------------*/
/*--------------------------------------------------------------------*/

declare @fecha nvarchar(8)
set @fecha = convert(nvarchar(8), getdate(),112)
exec [dbo].[procedure_resumen_semanal] @fecha
```

Donde la variable **@fecha** se la pasamos a cada procedimiento, y es la fecha en la que lanzamos los procesos.

- **procedure_update_linea**: Todos los datos extraídos no nos vendrán con unas categorías de producto definidas o al menos, no definidas por nosotros. Para esto estará este procedure. Cogerá los datos de cada día/semana y analizando los nombres de los productos, sus links y las urls de sus escaparates les pondrá un nombre de categoría definido por Venca. Por ejemplo, “Camiseta de tirantes”, la podrá en la categoría de “Camisetas”, y así con todos los productos que vaya encontrando analizados en la última ejecución.
- **procedure_novedades**: Como su propio nombre indica, este será el procedure encargado de encontrar las novedades de la última ejecución de los scripts, comparando los productos con todo el historial que hay de ejecuciones y extrayendo solo los nuevos que no ha encontrado previamente.
- **resumen_superofertas**: Procedure encargado de detectar cualquier producto con descuento y dividirlo en diversas franjas puestas por Venca, siendo >70% la mayor y entre 0,1% y 10% la menor.
- **procedure_resumen_ofertas**: Con los datos extraídos previamente, hacemos un pequeño resumen para saber en cada categoría cuantos productos tenemos divididos por tipo descuento. Nos da una visión general de la oferta de la competencia, pudiendo llegar a saber el número de Camisetas en oferta y con qué tipo de descuento.
- **procedure_resumen_semanal**: Este procedure se encargará de hacer el resumen final con el cual elaboraremos los Excels. Se encarga de hacer un resumen para el top 20 de los productos (20 primeras posiciones de los escaparates) y otro para el total de productos. Primero, por categoría de producto, pone los precios a cada franja y finalmente con estas categorías y franjas llama a un subprocedure que se encargará de realizar el resumen final:
    - **resumen_diario**: Procedure que recibe como parámetros una fecha, una categoría y 7 franjas de precios; con ellas sacará por marca y categoría, el número de modelos por cada categoría, precio medio, precio mínimo, cuartiles, mediana, precio máximo y Número de artículos por franjas de precio.

4.3.2.2 Procedures

1. procedure_update_linea (acortado, original ocuparía más de 20 páginas):

```
--use MKT_CRAWLER
--drop procedure [dbo].[procedure_update_linea]
--create procedure [dbo].[procedure_update_linea] (@fecha nvarchar(8))
alter procedure [dbo].[procedure_update_linea] (@fecha nvarchar(8))
AS
BEGIN

      IF OBJECT_ID('mkt_crawler.dbo.analisis_crawlervenca', 'U') IS NOT NULL
        DROP TABLE mkt_crawler.dbo.analisis_crawlervenca
      select * into mkt_crawler.dbo.analisis_crawlervenca from crawlervenca
where convert(nvarchar(8),crawl_date,112) = @fecha

      delete from MKT_CRAWLER.dbo.crawlervenca where
convert(nvarchar(8),crawl_date,112) = @fecha

      delete from mkt_crawler.dbo.analisis_crawlervenca where actualprice = 0
or pricebeforediscount = 0

      update MKT_CRAWLER.dbo.analisis_crawlervenca
      set   linea_2 = case

                  […]

                  when description like '%c_miseta%' then 'Camisetas'
                  when description like '%c_misa%' then 'Camisas'
                  when description like '%americ%' then 'Americana'
                  when description like '%blazer%' then 'Americana'
                  when description like '%blasier%' then 'Americana'
                  when description like '%blzaer%' then 'Americana'
                  when description like '%parka%' then 'Parka'
                  when description like '%parca%' then 'Parka'
                  when description like '%plum_n%' then 'Parka'
                  when description like '%plum_fero%' then 'Parka'
                  when description like '%plum__fero%' then 'Parka'
                  when description like '%anora_%' then 'Parka'
                  when description like '%trenc%' then 'Parka'
                  when description like '%trenka%' then 'Parka'
                  when description like '%plumas%' then 'Parka'
                  when description like '%bomber%' then 'Bomber'
                  when description like '%j_rs_y%' then 'Jerseis'
                  when description like '%j_rs_i%' then 'Jerseis'
                  when description like '%c_rdig_n%' then 'Cardigan'
                  when description like '%antal_n%' then 'Pantalones'
                  when description like '%antal__n%' then 'Pantalones'
                  when description like '%antal___n%' then 'Pantalones'
                  when description like '%abrigo%' then 'Abrigo'
                  when description like '%chaquet_n%' then 'Abrigo'
                  when description like '%coat%' then 'Abrigo'
                  when description like '%jacket%' then 'Abrigo'
                  when description like '%ch_quet_%' then 'PA'
                  when description like '%mocas_n%' then 'Mocasin'
                  when description like '%MOCAS__N%' then 'Mocasin'
                  when description like '%zapatilla%' then 'Zapatillas'
                  when description like '%mono%' then 'Mono'
```

```
                when description like '%leggin%' then 'Leggings'
                when description like '%legin%' then 'Leggings'
                when description like '%jeggi%' then 'Jeggings'
                when description like '%jegin%' then 'Jeggings'
                when description like '%jogg%' then 'Joggings'
                when description like '%jogin%' then 'Joggings'
                when description like '%treggin%' then 'Treggings'
                when description like '%tregin%' then 'Treggings'
                when description like '%peto%' then 'Mono'
                when description like '%sudadera%' then 'Sudadera'
                when description  like '%fald%' then 'Falda'
                when description like '%gabardina%' then 'Abrigo'
                when description  like '%jeans%'  then 'Jeans'
                when description like '%bolero%' then 'Bolero'
                when description like '%braga%' then 'Bragas'
                when description like '%braguit%' then 'Bragas'
                when description like '%shorties%' then 'Bragas'
                when description like '%shorty%' then 'Bragas'
                when description like '%tanga%' then 'Bragas'
                when description like '%b__xer%' then 'Bragas'
                when description like '%b_xer%' then 'Bragas'
                when description like '%bagas%' then 'Bragas'
                when description like '%short%' then 'Bermuda_Short'
                when description like '%cmta%' then 'Camisetas'
                when description like '%gomas%' then 'Complementos'
                when description like '%anillo%' then 'Complementos'
                when description like '%bisuter%' then 'Complementos'
                when description like '%cuerpo%' then 'Body'
                when description like '%abarca%' then 'Alpargata'
                when description like '%pichi%' then 'Vestidos'
                when description like '%t_nica%' then 'Camisas'
                when description like '%bot_n%' then 'Botin'
                when description like '%bot__n%' then 'Botin'
                when description like '%bot___n%' then 'Botin'
                when description like '%boot%' then 'Botin'
                when description like '%bo_in%' then 'Botin'
                when description like '%bo_in%' then 'Botin'
                when description like '%alpargata%' then 'Alpargata'
                when description like '%bailarina%' then 'Manoletina'
                when description like '%bamba%' then 'Bamba'
                when description like '%blucher%' then 'Blucher'
                when description like '%gorra%' then 'Complementos'
                when description like '%cu_a%' then 'Sandalias'
                when description like '%sandal%' then 'Sandalias'
                when description like '%zuec%' then 'Zuecos'
                when description like '%bota%' then 'Botas'
                when description like '%chancl%' then 'Chanclas'
                when description like '%chinelas%' then 'Zapatillas'
                when description like '%sal_n%' then 'Salon'
                when description like '%SAL__N%' then 'Salon'
                when description like '%pala%' then 'Sandalias'
                when description like '%sneaker%' then 'Bamba'
                when description like '%trainer%' then 'Bamba'
                when description like '%brogue%' then 'Blucher'
                when description like '%loafer%' then 'Manoletina'
                when description like '%pumps%' then 'Manoletina'
                when description like '%moccas%' then 'Mocasin'
                when description like '%reebok%' then 'Bamba'
                when description like '%mules%' then 'Sandalias'
```

```
                when description like '%kitten%' then 'Salon'
                when description like '%cu%C3%B1a%' then 'Sandalias'
                when description like '%ballet%' then 'Manoletina'
                when description like '%bot%C3%Adn%' then 'Botin'
                when description like '%romanas%' then 'Sandalias'
                when description like '%playeras%' then 'Chanclas'
                when description like '%cangrejera%' then 'Sandalias'
                when description like '%cu__a%' then 'Sandalias'
                when description like '%kiowa%' then 'Botin'
                when description like '%mocas%C3%Adn%' then 'Mocasin'
                when description like '%espar_e_a%' then 'Esparteñas'
                when description like '%shoe%' then 'Zapatos'
                when description like '%zapa%' then 'Zapatos'
                when description like '%tac_n%' then 'Zapatos'
                when description like '%tac__n%' then 'Zapatos'
                when description like '%body%' then 'Body'
                when description like '%trousers%' then 'Pantalones'
                when description like '%pant_%' then 'Pantalones'
                when description like '%patal_n%' then 'Pantalones'
                when description like '%pntal_%' then 'Pantalones'
                when description like '%malla%' then 'Pantalones'
                when description like '%chino%' then 'Pantalones'
                when description like '%acampan%' then 'Pantalones'
                when description like '%ajustado%' then 'Pantalones'
                when description like '%bootcut%' then 'Pantalones'
                when description like '%boyfriend%' then 'Pantalones'
                when description like '%pirata%' then 'Pantalones'
                when description like '%slack%' then 'Pantalones'
                when description like '%pinocc%' then 'Pantalones'
                when description like '%ptl_%' then 'Pantalones'
                when description like '%tejano%' then 'Pantalones'
                when description like '%skirt%' then 'Falda'
                when description like '%bufanda%' then 'Complementos'
                when description like '%guante%' then 'Complementos'
                when description like '%top%' then 'Camisetas'
                when description like '%blus_n%' then 'Camisas'
                when description like '%blus__n%' then 'Camisas'
                when description like '%blouse%' then 'Camisas'
                when description like '%bot_n%' then 'Botin'
                when description like '%brag%' then 'Bragas'
                when description like '%braq%' then 'Bragas'
                when Description like '%alfombra%' then 'Hogar'
                when Description like '%sombras%' then 'Complementos'
                when description like '%bralet%' then 'Camisetas'
                when description like '%bustie%' then 'Corset'
                when description like '%cors_%' then 'Corset'
                when description like '%caft_n%' then 'Camisas'
                when description like '%kaft_n%' then 'Camisas'
                when description like '%t_nica%' then 'Camisas'
                when description like '%t__nica%' then 'Camisas'
                when description like '%camis_n%' then 'Camison'
                when description like '%camis__n%' then 'Camison'
                when description like '%camis_l%' then 'Camison'
                when description like '%camisesta%' then 'Camisetas'
                when description like '%camista%' then 'Camisetas'
                when description like '%capaz%' then 'Complementos'
                when description like '%capel%' then 'Complementos'
                when description like '%cmsa%' then 'Camisas'
                when description like '%albornoz%' then 'Baño'
                when description like '%culot%' then 'Bragas'
                when description like '%h_pster%' then 'Bragas'
```

```
                when description like '%jean%' then 'Jeans'
                when description like '%jerser%' then 'Jerseis'
                when description like '%slippers%' then 'Zapatillas'
                when description like '%runn%' then 'Bamba'
                when description like '%jumper%' then 'Jerseis'
                when description like '%polo%' then 'Camisas'
      […]


                else 'sin_def'
      end

      --where execution_time like '%20161017%'
      --where linea_2 = 'sin_def'
      where linea_2 is NULL


      update MKT_CRAWLER.dbo.analisis_crawlervenca
      set fecha = convert(nvarchar(8),crawl_date,112) where fecha is NULL



      IF OBJECT_ID('mkt_crawler.dbo.ntilecrawler', 'U') IS NOT NULL
        DROP TABLE mkt_crawler.dbo.ntilecrawler


      SELECT
            *,
            ntile(4) over (
                                    partition by
      convert(nvarchar(8),crawl_date,112),
                                                            Client,
                                                            brand,
                                                            linea_2
                                    order by actualprice
                                    ) q
      into mkt_crawler.dbo.ntilecrawler
      FROM MKT_CRAWLER.dbo.analisis_crawlervenca
      where cuartil is NULL
      --(63281 filas afectadas)
      /*
      delete from MKT_CRAWLER.dbo.analisis_crawlervenca where cuartil is NULL
      --(63281 filas afectadas)
      */
      SET IDENTITY_INSERT crawlervenca ON

      INSERT INTO crawlervenca (Client, Category, Brand, SubBrand, URL, Link,
NEW, Description, Comment, PriceBeforeDiscount, ActualPrice, ID_Presentation,
Top20, Crawl_Day, Crawl_Date, Id, Palabra1,
                                                IDCatSem1, Palabra2, IDCatSem2,
#Colors, CodeProd, MinSize, MaxSize, Execution_Time, id_num, linea, linea_2,
cuartil, fecha)
      SELECT Client, Category, Brand, SubBrand, URL, Link, NEW, Description,
Comment, PriceBeforeDiscount, ActualPrice, ID_Presentation, Top20, Crawl_Day,
                  Crawl_Date, Id, Palabra1, IDCatSem1, Palabra2, IDCatSem2,
#Colors, CodeProd, MinSize, MaxSize, Execution_Time, id_num, linea, linea_2,
q, fecha
      FROM MKT_CRAWLER.dbo.ntilecrawler
      --(63281 filas afectadas)

END
```

2. procedure_novedades:

```
/*novedades*/
--drop procedure [dbo].[procedure_novedades]
--create procedure [dbo].[procedure_novedades] (@fecha nvarchar(8))
alter procedure [dbo].[procedure_novedades] (@fecha nvarchar(8))
AS
BEGIN

	insert into mkt_crawler.dbo.crawlerNovedades
(client,brand,category,url,link,description,new,PriceBeforeDiscount,ActualPr
ice,ID_Presentation,Top20,Crawl_Day,Crawl_Date,#Colors,CodeProd,MinSize,MaxS
ize,Execution_Time,linea_2,cuartil,fecha,brcodeprod,longDescription)
	select

	client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha,
brand+'_'+convert(nvarchar(15),codeprod) brcodeprod, 'NULL' longDescription
	from (select

	client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha, --
brand,link,codeprod,execution_time,crawl_date,
		row_number() over( partition by brand,CodeProd order by
execution_time) novedad from mkt_crawler.dbo.crawlervenca
		where client = 'venca'
		) a
	where novedad = 1  and convert(nvarchar(8),crawl_date,112) = @fecha
	order by	brand,category,Description

	update mkt_crawler.dbo.crawlerNovedades
	set fecha = convert(nvarchar(8),crawl_date,112) where fecha is NULL

END
```

3. resumen_superofertas:

```
--drop procedure [dbo].[resumen_superofertas]
--create procedure [dbo].[resumen_superofertas] (@fecha nvarchar(8))
alter procedure [dbo].[resumen_superofertas] (@fecha nvarchar(8))
AS
BEGIN
--mkt_crawler.dbo.analisis_mkt_crawler.dbo.analisis_crawlervenca
	insert into crawlerSuperOfertas
(client,brand,category,url,link,description,new,PriceBeforeDiscount,ActualPr
ice,ID_Presentation,Top20,Crawl_Day,Crawl_Date,#Colors,CodeProd,MinSize,MaxS
ize,Execution_Time,linea_2,cuartil,fecha,descuento,brcodeprod,tipo_descuento
,novedad)
	select

	client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha,
		(1-
convert(float,actualprice)/convert(float,pricebeforediscount)) descuento,
		brand+'_'+convert(nvarchar(15),codeprod) brcodeprod,
		'> 70%' tipo_descuento, novedad
	from (select
client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha,
		row_number() over( partition by brand,CodeProd order by
execution_time) novedad from mkt_crawler.dbo.analisis_crawlervenca where
client = 'venca' and convert(nvarchar(8),crawl_date,112) = @fecha) a
	where client = 'venca' and convert(nvarchar(8),crawl_date,112) =
@fecha and (1-
convert(float,actualprice)/convert(float,pricebeforediscount)) >= '0.7' and
novedad = 1

	insert into crawlerSuperOfertas
(client,brand,category,url,link,description,new,PriceBeforeDiscount,ActualPr
ice,ID_Presentation,Top20,Crawl_Day,Crawl_Date,#Colors,CodeProd,MinSize,MaxS
ize,Execution_Time,linea_2,cuartil,fecha,descuento,brcodeprod,tipo_descuento
,novedad)
	select

	client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha,
		(1-
convert(float,actualprice)/convert(float,pricebeforediscount)) descuento,
		brand+'_'+convert(nvarchar(15),codeprod) brcodeprod,
	--into mkt_crawler.dbo.crawlerSuperOfertas
		'Entre 50% y 70%' tipo_descuento, novedad
	from (select
client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha,
		row_number() over( partition by brand,CodeProd order by
execution_time) novedad from mkt_crawler.dbo.analisis_crawlervenca where
client = 'venca' and convert(nvarchar(8),crawl_date,112) = @fecha) a
where client = 'venca' and convert(nvarchar(8),crawl_date,112) = @fecha and
(1-convert(float,actualprice)/convert(float,pricebeforediscount)) >= '0.5'
and (1-convert(float,actualprice)/convert(float,pricebeforediscount))< '0.7'
and novedad = 1
```

```
insert into crawlerSuperOfertas
(client,brand,category,url,link,description,new,PriceBeforeDiscount,ActualPr
ice,ID_Presentation,Top20,Crawl_Day,Crawl_Date,#Colors,CodeProd,MinSize,MaxS
ize,Execution_Time,linea_2,cuartil,fecha,descuento,brcodeprod,tipo_descuento
,novedad)
      select

      client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha,
            (1-
convert(float,actualprice)/convert(float,pricebeforediscount)) descuento,
            brand+'_'+convert(nvarchar(15),codeprod) brcodeprod,
      --into mkt_crawler.dbo.crawlerSuperOfertas
            'Entre 30% y 50%' tipo_descuento, novedad
      from (select
client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha,
            row_number() over( partition by brand,CodeProd order by
execution_time) novedad from mkt_crawler.dbo.analisis_crawlervenca where
client = 'venca' and convert(nvarchar(8),crawl_date,112) = @fecha) a
      where client = 'venca' and convert(nvarchar(8),crawl_date,112) =
@fecha and (1-
convert(float,actualprice)/convert(float,pricebeforediscount)) >= '0.3' and
(1-convert(float,actualprice)/convert(float,pricebeforediscount))< '0.5' and
novedad = 1

      insert into crawlerSuperOfertas
(client,brand,category,url,link,description,new,PriceBeforeDiscount,ActualPr
ice,ID_Presentation,Top20,Crawl_Day,Crawl_Date,#Colors,CodeProd,MinSize,MaxS
ize,Execution_Time,linea_2,cuartil,fecha,descuento,brcodeprod,tipo_descuento
,novedad)
      select

      client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha,
            (1-
convert(float,actualprice)/convert(float,pricebeforediscount)) descuento,
            brand+'_'+convert(nvarchar(15),codeprod) brcodeprod,
      --into mkt_crawler.dbo.crawlerSuperOfertas
            'Entre 20% y 30%' tipo_descuento, novedad
      from (select
client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha,
            row_number() over( partition by brand,CodeProd order by
execution_time) novedad from mkt_crawler.dbo.analisis_crawlervenca where
client = 'venca' and convert(nvarchar(8),crawl_date,112) = @fecha) a
      where client = 'venca' and convert(nvarchar(8),crawl_date,112) =
@fecha and (1-
convert(float,actualprice)/convert(float,pricebeforediscount)) >= '0.2' and
(1-convert(float,actualprice)/convert(float,pricebeforediscount))< '0.3' and
novedad = 1
insert                        into                        crawlerSuperOfertas
(client,brand,category,url,link,description,new,PriceBeforeDiscount,ActualPr
ice,ID_Presentation,Top20,Crawl_Day,Crawl_Date,#Colors,CodeProd,MinSize,MaxS
ize,Execution_Time,linea_2,cuartil,fecha,descuento,brcodeprod,tipo_descuento
,novedad)
```

```
select
      client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha,
            (1-
convert(float,actualprice)/convert(float,pricebeforediscount)) descuento,
            brand+'_'+convert(nvarchar(15),codeprod) brcodeprod,
      --into mkt_crawler.dbo.crawlerSuperOfertas
            'Entre 10% y 20%' tipo_descuento, novedad
      from (select
client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha,
            row_number() over( partition by brand,CodeProd order by
execution_time) novedad from mkt_crawler.dbo.analisis_crawlervenca where
client = 'venca' and convert(nvarchar(8),crawl_date,112) = @fecha) a
      where client = 'venca' and convert(nvarchar(8),crawl_date,112) =
@fecha and (1-
convert(float,actualprice)/convert(float,pricebeforediscount)) >= '0.1' and
(1-convert(float,actualprice)/convert(float,pricebeforediscount))< '0.2' and
novedad = 1

      insert into crawlerSuperOfertas
(client,brand,category,url,link,description,new,PriceBeforeDiscount,ActualPr
ice,ID_Presentation,Top20,Crawl_Day,Crawl_Date,#Colors,CodeProd,MinSize,MaxS
ize,Execution_Time,linea_2,cuartil,fecha,descuento,brcodeprod,tipo_descuento
,novedad)
      select

      client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha,
            (1-
convert(float,actualprice)/convert(float,pricebeforediscount)) descuento,
            brand+'_'+convert(nvarchar(15),codeprod) brcodeprod,
      --into mkt_crawler.dbo.crawlerSuperOfertas
            'Entre 0.1% y 10%' tipo_descuento, novedad
      from (select
client,brand,category,url,link,description,new,PriceBeforeDiscount,
ActualPrice, ID_Presentation, Top20,Crawl_Day, Crawl_Date,#Colors, CodeProd,
MinSize, MaxSize, Execution_Time,linea_2, cuartil,fecha,
            row_number() over( partition by brand,CodeProd order by
execution_time) novedad from mkt_crawler.dbo.analisis_crawlervenca where
client = 'venca' and convert(nvarchar(8),crawl_date,112) = @fecha) a
      where client = 'venca' and convert(nvarchar(8),crawl_date,112) =
@fecha and (1-
convert(float,actualprice)/convert(float,pricebeforediscount)) > '0' and (1-
convert(float,actualprice)/convert(float,pricebeforediscount))< '0.1' and
novedad = 1

END
```

4. procedure_resumen_ofertas:

```
/*resumen_ofertas*/

--use MKT_CRAWLER
--drop procedure [dbo].[procedure_resumen_ofertas]
--create procedure [dbo].[procedure_resumen_ofertas] (@fecha nvarchar(8))
alter procedure [dbo].[procedure_resumen_ofertas] (@fecha nvarchar(8))
AS
BEGIN
--select distinct fecha from MKT_CRAWLER.dbo.crawlerSuperOfertas

	insert into MKT_CRAWLER.dbo.resumen_crawler_ofertas
	select
		brand Marca,
		linea_2 Categoria,
		Tipo_descuento,
		convert(nvarchar(8),crawl_date,112) Fecha,
		count(distinct codeprod) nModelos,
		Round(avg(actualprice),2,0) PrecioMedio,
		Round(min(actualprice),2,0) PrecioMinimo,
		Round(max(actualprice),2,0) PrecioMax,
		new Tallas_Grandes
	from MKT_CRAWLER.dbo.crawlerSuperOfertas q
	where convert(nvarchar(8),crawl_date,112) = @fecha
	group by
		brand,
		linea_2,
		tipo_descuento,
		new,
		convert(nvarchar(8),crawl_date,112)
	order by brand asc,Categoria asc,tipo_descuento asc

END
```

5. procedure_resumen_semanal:

```
--drop procedure [dbo].[procedure_resumen_semanal]
--create procedure [dbo].[procedure_resumen_semanal] (@fecha nvarchar(8))
alter procedure [dbo].[procedure_resumen_semanal] (@fecha nvarchar(8))
AS
BEGIN

	IF OBJECT_ID('mkt_crawler.dbo.crawlertotal', 'U') IS NOT NULL
	  DROP TABLE mkt_crawler.dbo.crawlertotal

	select
		*
	into mkt_crawler.dbo.crawlertotal
	from mkt_crawler.dbo.crawlervenca
	where convert(nvarchar(8),crawl_date,112) = @fecha

	update mkt_crawler.dbo.crawlertotal
	set fecha = convert(nvarchar(8),crawl_date,112) where fecha is NULL


	IF OBJECT_ID('mkt_crawler.dbo.crawlertop20', 'U') IS NOT NULL
	  DROP TABLE mkt_crawler.dbo.crawlertop20

	select
		*
	into mkt_crawler.dbo.crawlertop20
	from crawlervenca
	where top20 = 'TRUE' and convert(nvarchar(8),crawl_date,112) = @fecha

	update mkt_crawler.dbo.crawlertop20
	set fecha = convert(nvarchar(8),crawl_date,112) where fecha is NULL

	declare @linea_2 nvarchar(50),@f1 int, @f2 int, @f3 int, @f4 int, @f5
int, @f6 int, @f7 int
	DECLARE campos_cursor CURSOR
	   FOR	select * from mkt_crawler.dbo.resumen_franjas
	OPEN campos_cursor
	FETCH NEXT FROM campos_cursor
	into  @linea_2,@f1,@f2,@f3,@f4,@f5,@f6,@f7

	WHILE @@FETCH_STATUS = 0

	BEGIN

	  exec [dbo].[resumen_diario]
@fecha,@linea_2,@f1,@f2,@f3,@f4,@f5,@f6,@f7

	  FETCH NEXT FROM campos_cursor
	  into @linea_2,@f1,@f2,@f3,@f4,@f5,@f6,@f7
	END
	CLOSE campos_cursor
	DEALLOCATE campos_cursor

END
```

Analizando a la competencia

a. **resumen_diario:**

```
--drop procedure dbo.resumen_diario
create procedure [dbo].[resumen_diario] (@fecha nvarchar(8),@linea_2
nvarchar(50),@f1 int, @f2 int, @f3 int, @f4 int, @f5 int, @f6 int, @f7 int)
AS
BEGIN

insert into MKT_CRAWLER.dbo.crawlerresumen
(brand,Categoria,Fecha,nModelos,PrecioMedio,PrecioMinimo,Q1,Q2,Mediana,Q3,Q4
,PrecioMax,Franja1,Franja2,Franja3,Franja4,Franja5,Franja6,Franja7,tipo_top)
select
      brand,
      linea_2 Categoria,
      convert(nvarchar(8),crawl_date,112) Fecha,
      count(distinct codeprod) nModelos,
      Round(avg(actualprice),2,0) PrecioMedio,
      Round(min(actualprice),2,0) PrecioMinimo,
      Round(( SELECT
            avg(actualprice)
            FROM crawlertotal c
            WHERE cuartil = 1 and c.brand = q.brand and c.linea_2 =
q.linea_2 and convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client !=
'Francia'
      ),2,0) Q1,
      Round(( SELECT
            avg(actualprice)
            FROM crawlertotal c
            WHERE cuartil = 2 and c.brand = q.brand and c.linea_2 =
q.linea_2 and convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client !=
'Francia'
      ),2,0) Q2,
      Round(( SELECT
      (
       (SELECT MAX(actualprice) FROM
         (SELECT TOP 50 PERCENT c.actualprice FROM crawlertotal c WHERE
c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia'
ORDER BY c.actualprice) AS BottomHalf)
        +
       (SELECT MIN(actualprice) FROM
         (SELECT TOP 50 PERCENT c.actualprice FROM crawlertotal c WHERE
c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia'
ORDER BY c.actualprice DESC) AS TopHalf)
      ) / 2
      ),2,0) Mediana,
      Round(( SELECT
            avg(actualprice)
            FROM crawlertotal c
            WHERE cuartil = 3 and c.brand = q.brand and c.linea_2 =
q.linea_2 and convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client !=
'Francia'
      ),2,0) Q3,
      Round(( SELECT
            avg(actualprice)
            FROM crawlertotal c
            WHERE cuartil = 4 and c.brand = q.brand and c.linea_2 =
q.linea_2 and convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client !=
'Francia'),2,0) Q4,
```

```
Round(max(actualprice),2,0) PrecioMax,
	( SELECT
		count(distinct c.codeprod)
		FROM crawlertotal c
		WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice <@f1) Franja1,
	( SELECT
		count(distinct c.codeprod)
		FROM crawlertotal c
		WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice >= @f1 AND c.actualprice <@f2) Franja2,
	( SELECT
		count(distinct c.codeprod)
		FROM crawlertotal c
		WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice >= @f2 AND c.actualprice <@f3) Franja3,
	( SELECT
		count(distinct c.codeprod)
		FROM crawlertotal c
		WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice >= @f3 AND c.actualprice <@f4) Franja4,
	( SELECT
		count(distinct c.codeprod)
		FROM crawlertotal c
		WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice >= @f4 AND c.actualprice <@f5) Franja5,
	( SELECT
		count(distinct c.codeprod)
		FROM crawlertotal c
		WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice >= @f5 AND c.actualprice <@f6) Franja6,
	( SELECT
		count(distinct c.codeprod)
		FROM crawlertotal c
		WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice >= @f6 AND c.actualprice <@f7) Franja7,

	'total' tipo_top
	--into mkt.dbo.crawlerresumen
	from crawlertotal q
	where convert(nvarchar(8),crawl_date,112) = @fecha and Client !=
'Francia' and linea_2 = @linea_2
	group by
		brand,
		linea_2,
		convert(nvarchar(8),crawl_date,112)

insert into MKT_CRAWLER.dbo.crawlerresumen
(brand,Categoria,Fecha,nModelos,PrecioMedio,PrecioMinimo,Q1,Q2,Mediana,Q3,Q4
,PrecioMax,Franja1,Franja2,Franja3,Franja4,Franja5,Franja6,Franja7,tipo_top)
```

```
select
            brand,
            linea_2 Categoria,
            convert(nvarchar(8),crawl_date,112) Fecha,
            count(distinct codeprod) nModelos,
            Round(avg(actualprice),2,0) PrecioMedio,
            Round(min(actualprice),2,0) PrecioMinimo,
            Round(( SELECT
                  avg(actualprice)
                  FROM crawlertop20 c
                  WHERE cuartil = 1 and c.brand = q.brand and c.linea_2 =
q.linea_2 and convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client !=
'Francia'
            ),2,0) Q1,
            Round(( SELECT
                  avg(actualprice)
                  FROM crawlertop20 c
                  WHERE cuartil = 2 and c.brand = q.brand and c.linea_2 =
q.linea_2 and convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client !=
'Francia'
            ),2,0) Q2,
            Round(( SELECT
            (
             (SELECT MAX(actualprice) FROM
               (SELECT TOP 50 PERCENT c.actualprice FROM crawlertop20 c
WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia'
ORDER BY c.actualprice) AS BottomHalf)
              +
             (SELECT MIN(actualprice) FROM
               (SELECT TOP 50 PERCENT c.actualprice FROM crawlertop20 c
WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia'
ORDER BY c.actualprice DESC) AS TopHalf)
            ) / 2
            ),2,0) Mediana,
            Round(( SELECT
                  avg(actualprice)
                  FROM crawlertop20 c
                  WHERE cuartil = 3 and c.brand = q.brand and c.linea_2 =
q.linea_2 and convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client !=
'Francia'
            ),2,0) Q3,
            Round(( SELECT
                  avg(actualprice)
                  FROM crawlertop20 c
                  WHERE cuartil = 4 and c.brand = q.brand and c.linea_2 =
q.linea_2 and convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client !=
'Francia'
            ),2,0) Q4,
            Round(max(actualprice),2,0) PrecioMax,
            ( SELECT
                  count(distinct c.codeprod)
                  FROM crawlertop20 c
                  WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice <@f1) Franja1,
            ( SELECT
                  count(distinct c.codeprod)
                  FROM crawlertop20 c
```

```
WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice >= @f1 AND c.actualprice <@f2) Franja2,
            ( SELECT
                  count(distinct c.codeprod)
                  FROM crawlertop20 c
                  WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice >= @f2 AND c.actualprice <@f3) Franja3,
            ( SELECT
                  count(distinct c.codeprod)
                  FROM crawlertop20 c
                  WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice >= @f3 AND c.actualprice <@f4) Franja4,
            ( SELECT
                  count(distinct c.codeprod)
                  FROM crawlertop20 c
                  WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice >= @f4 AND c.actualprice <@) Franja5,
            ( SELECT
                  count(distinct c.codeprod)
                  FROM crawlertop20 c
                  WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice >= @f5 AND c.actualprice <@f6) Franja6,
            ( SELECT
                  count(distinct c.codeprod)
                  FROM crawlertop20 c
                  WHERE c.brand = q.brand and c.linea_2 = q.linea_2 and
convert(nvarchar(8),c.crawl_date,112) = @fecha and c.Client != 'Francia' and
c.actualprice >= @f6 AND c.actualprice <@f7) Franja7,
      'top20' tipo_top
      --into mkt.dbo.crawlerresumen
      from crawlertop20 q
      where convert(nvarchar(8),crawl_date,112) = @fecha and Client !=
'Francia' and linea_2 = @linea_2
      group by
            brand,
            linea_2,
            convert(nvarchar(8),crawl_date,112)
END
```

4.4 Informes

4.4.1 Conocimiento de los datos

Una vez tenemos nuestra base de datos consolidada y recogiendo datos a diario o cada cierto periodo de tiempo, toca generar unos informes fáciles de entender y de leer para todo el mundo.

Gracias a estos informes podremos saber cómo evoluciona nuestro plan de campaña online frente al de otras marcas y seguidamente tomar decisiones de como continuar con él.

A continuación, se muestra una parte de los informes en la que podemos comparar la categoría “Vestidos” de Venca respecto a otras marcas como pueden ser Kiabi, C&A, H&M…

En el siguiente cuadro podemos observar el número de modelos que dispone cada marca para esta categoría, cuál es su precio medio, precio mínimo y máximo, entre otros.

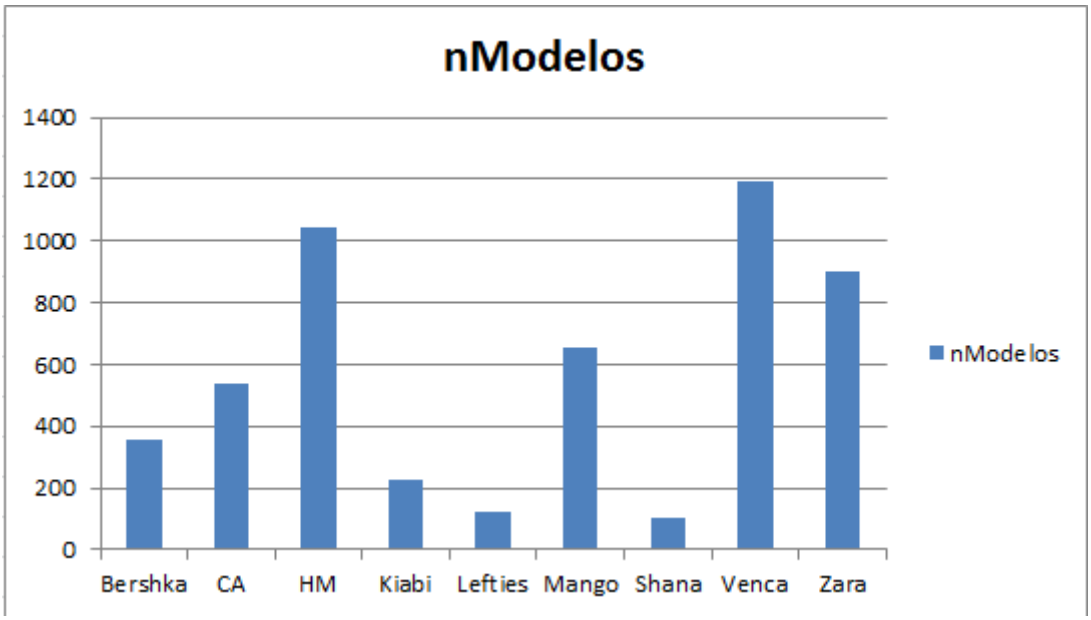
brand	Categoria	Fecha	nModelos	PrecioMedio	PrecioMinimo	Q1	Q2	Mediana	Q3	Q4	PrecioMax
Bershka	Vestidos	20170515	92	21,21	9,99	13,89	17,96	19,99	21,85	31,32	39,99
CA	Vestidos	20170515	287	35,05	6,99	12,93	23,64	29,9	36,98	66,96	129,9
HM	Vestidos	20170515	695	25,26	4,99	10,4	16,11	19,99	23,95	50,71	199
Kiabi	Vestidos	20170515	129	19,65	7	11,34	18,36	20	20,9	28,07	80
Lefties	Vestidos	20170515	58	13,86	7	9,96	12,57	15	15,61	17,3	21
Mango	Vestidos	20170515	581	40,8	9,99	20,53	29,28	29,99	39,78	73,68	149,99
Shana	Vestidos	20170515	18	14,1	12,99	12,99	13,13	13,99	14,99	15,45	15,99
Stradivarius	Vestidos	20170515	2	24,95	19,95	19,95	29,95	24,95	en blanco	en blanco	29,95
Venca	Vestidos	20170515	1005	23,88	5,39	12,12	18,77	19,99	24,25	40,4	119
Zara	Vestidos	20170515	578	28,31	12,95	16,84	25,35	25,95	29,28	41,82	89,95

Y así con cualquier categoría de ropa que hayamos fijado:

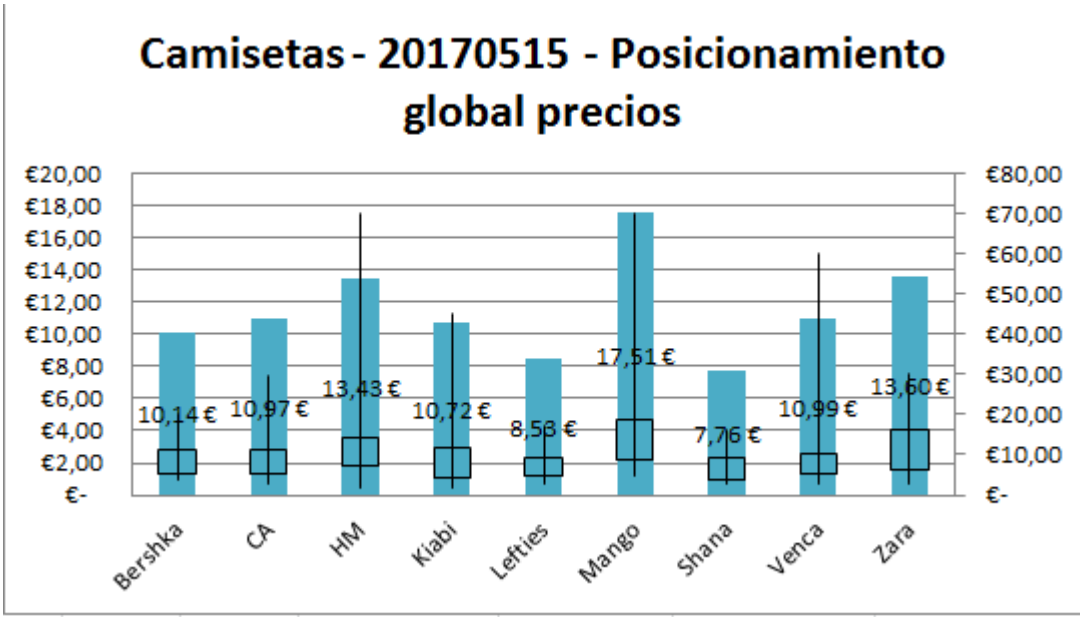
brand	Categoria	Fecha	nModelos	PrecioMedio	PrecioMinimo	Q1	Q2	Mediana	Q3	Q4	PrecioMax
Bershka	Camisetas	20170515	357	10,14	3,99	5,44	8,64	9,99	11,19	15,31	19,99
CA	Camisetas	20170515	538	10,97	3	5,17	7,96	9,9	11,29	19,5	29,9
HM	Camisetas	20170515	1045	13,43	1,99	7,04	10,36	12,99	14,36	21,99	69,99
Kiabi	Camisetas	20170515	223	10,72	2	4,43	8,59	10	11,68	18,2	45
Lefties	Camisetas	20170515	123	8,53	3	5,01	7,36	8,25	9,48	12,26	17
Mango	Camisetas	20170515	653	17,51	4,99	8,84	14,92	15,99	18,88	27,45	69,99
Shana	Camisetas	20170515	106	7,76	2,99	3,62	6,24	7,99	9,33	11,85	15,99
Venca	Camisetas	20170515	1194	10,99	2,99	5,21	7,77	8,99	10,42	20,56	59,99
Zara	Camisetas	20170515	900	13,6	2,95	6,47	11,91	12,95	16,08	19,96	29,95

Otra manera para mostrar los resultados es mediante gráficas.

A continuación, se muestra por ejemplo la distribución de modelos de camisetas en este caso, de cada marca, en la cual podemos ver como se sitúa Venca.

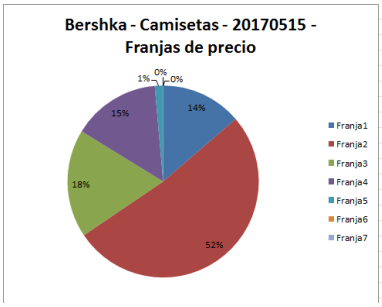


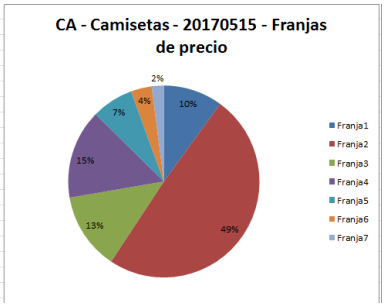
En el siguiente gráfico vemos el posicionamiento global de precios en cada marca y en cada categoría.

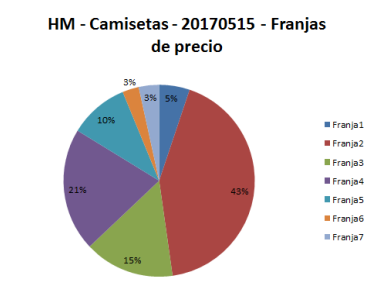


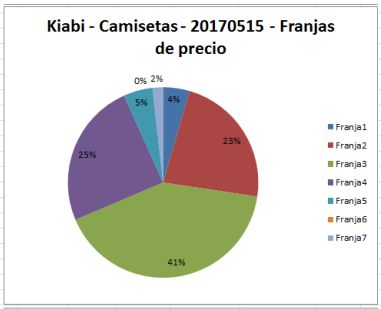
Siendo la columna izquierda el precio medio de los productos y la derecha los cuartiles de precios.

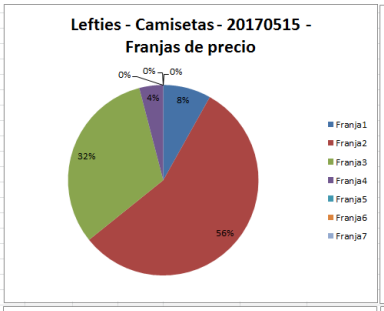
Y, por último, la distribución de modelos por franjas de precio de cada marca.

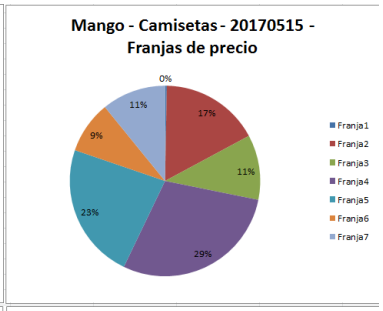


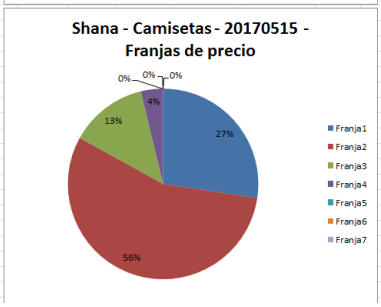


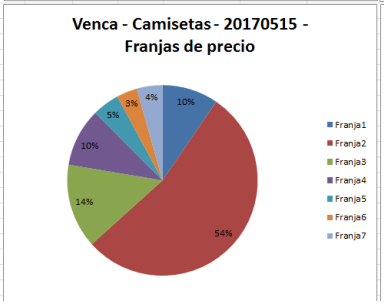


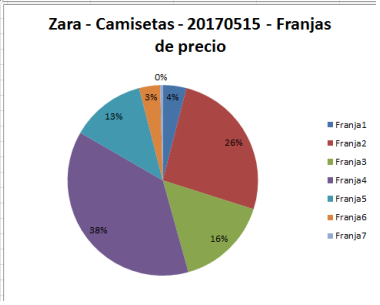












Habiendo definido previamente con el equipo al cargo, los precios para situar en cada franja.

Que en este caso serían (franjas en €):

Categoria	f1	f2	f3	f4	f5	f6	f7
Abrigo	40	50	60	80	100	130	999
Alpargata	0	0	0	0	0	0	999
Americana	20	30	36	40	50	70	999
Bamba	0	0	0	0	0	0	999
Bañador	0	0	0	0	0	0	999
Baño	0	0	0	0	0	0	999
Bata	16	20	26	30	40	50	999
Bermuda_Short	10	16	20	28	34	40	999
Bikini	0	0	0	0	0	0	999
Blucher	0	0	0	0	0	0	999
Body	13	16	20	26	30	38	999
Bolero	0	0	0	0	0	0	999
Bomber	20	30	40	70	90	110	999
Botas	0	0	0	0	0	0	999
Botin	0	0	0	0	0	0	999
Bragas	3	6	8	10	15	26	999
Bragas _Bikini	0	0	0	0	0	0	999
Camisas	10	16	22	28	34	40	999
Camisetas	5	10	14	19	24	29	999
Camison	10	16	20	26	30	40	999
Capa	0	0	0	0	0	0	999
Cardigan	13	18	23	27	33	38	999
Cazadora	20	35	50	65	80	95	999
Chaleco	16	20	24	28	32	36	999
Chanclas	0	0	0	0	0	0	999
Chandal	0	0	0	0	0	0	999
Complementos	0	0	0	0	0	0	999
Conjunto_Lenceria	10	20	26	30	40	50	999
Corset	0	0	0	0	0	0	999
Escarpines	0	0	0	0	0	0	999
Esparteñas	0	0	0	0	0	0	999
Fajas	10	16	20	26	30	40	999
Falda	10	15	20	25	30	35	999
Hogar	0	0	0	0	0	0	999
Jeans	16	20	27	34	41	48	999
Jeggings	16	20	27	34	41	48	999
Jerseis	13	18	23	28	33	38	999
Joggings	13	20	26	30	40	50	999
Leggings	8	11	14	17	20	23	999
Manoletina	0	0	0	0	0	0	999
Medias_Calcetines	0	0	0	0	0	0	999
Mocasin	0	0	0	0	0	0	999

Mono	16	23	30	37	44	51	999
PA	40	50	60	80	100	130	999
Pantalones	13	20	27	34	41	48	999
Parka	30	40	50	60	70	80	999
Pijama	10	16	20	30	40	50	999
Rincon_Sexy	0	0	0	0	0	0	999
Salon	0	0	0	0	0	0	999
Sandalias	0	0	0	0	0	0	999
Separables_Moda	20	30	40	70	90	110	999
sin_def	0	0	0	0	0	0	999
Sudadera	10	14	18	22	26	30	999
Sujetador	6	8	10	16	20	40	999
Sujetador_Bikini	0	0	0	0	0	0	999
Traje	36	50	60	70	90	110	999
Treggings	13	20	27	34	41	48	999
Trikini	0	0	0	0	0	0	999
Vestidos	13	20	27	34	41	48	999
Zapatillas	0	0	0	0	0	0	999
Zapatos	0	0	0	0	0	0	999
Zuecos	0	0	0	0	0	0	999

4.3.2 Automatizando el proceso

La generación de los informes podría llegar a ser diaria y requiere de un coste sobre todo temporal muy importante si no se hace de la manera más automatizada posible.

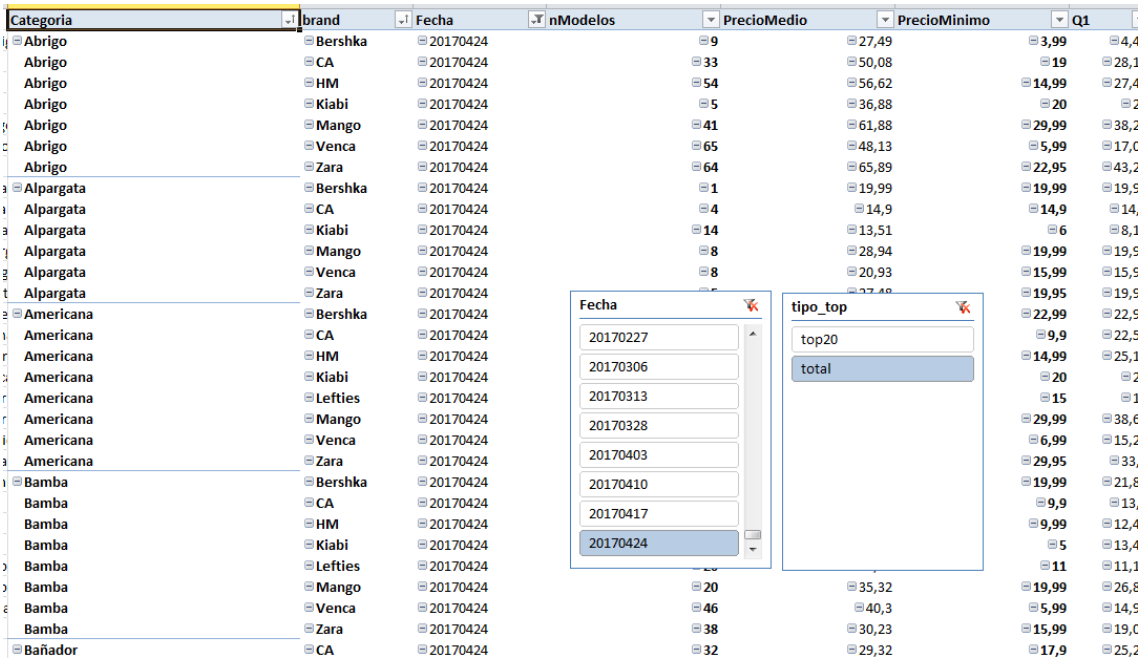
Tenemos que tener en cuenta que dentro del informe tenemos que tener tantas pestañas/hojas de Excel como categorías de producto hayamos generado (62), y en cada pestaña generar X gráficos y X tablas para leer esos datos de estas categorías.

Gracias a VBA y las macros, podemos conseguir que estas pestañas y gráficos se generen de manera automática con tan solo pulsar una combinación de teclas.

Lo primero que nos hace falta, es conectar nuestra base de datos a Excel.

Esto lo conseguimos:

1. Insertando una nueva tabla dinámica.
2. Utilizando una fuente de conexión externa.
3. Examinando en busca de más y eligiendo un nuevo origen.
4. DSN (nombre de origen de datos) de ODBC.
5. Seleccionando tu origen de datos y su respectiva tabla.



Una vez tenemos los datos, nos hace falta un patrón y una plantilla para que la macro sepa qué tiene que duplicar y en base a qué datos.

Para esto nos basta con tener siempre los datos ordenados de la misma forma y hacer una hoja de ejemplo de lo que queremos.

Por ejemplo, los datos de las tablas mostradas previamente, los cogeremos de la hoja en la que tenemos los datos del SQL con un CONSULTAV:

=CONSULTAV($A3;vccategoria!$A$1:$T$502;B$1;0)

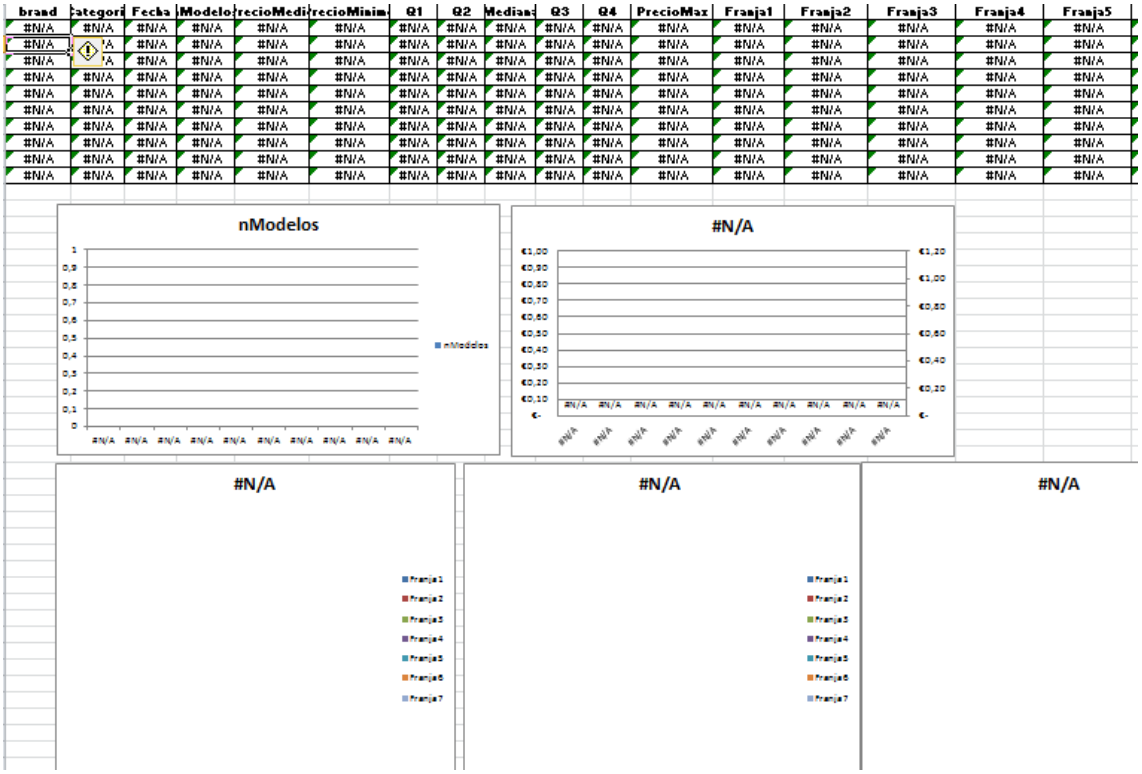
En A3 hemos preparado el valor que queremos buscar, y lo buscaremos en la tabla de datos que la tenemos en categoría y si lo encuentra cogeremos el valor de la B1 que concuerde con el nuestro buscado. De esta manera rellenamos este tipo de tablas:

brand	Categoria	Fecha	nModelos	PrecioMedio	PrecioMinimo	Q1	Q2	Mediana	Q3	Q4	PrecioMax
Bershka	Camisetas	20170515	357	10,14	3,99	5,44	8,64	9,99	11,19	15,31	19,99
CA	Camisetas	20170515	538	10,97	3	5,17	7,96	9,9	11,29	19,5	29,9
HM	Camisetas	20170515	1045	13,43	1,99	7,04	10,36	12,99	14,36	21,99	69,99
Kiabi	Camisetas	20170515	223	10,72	2	4,43	8,59	10	11,68	18,2	45
Lefties	Camisetas	20170515	123	8,53	3	5,01	7,36	8,25	9,48	12,26	17
Mango	Camisetas	20170515	653	17,51	4,99	8,84	14,92	15,99	18,88	27,45	69,99
Shana	Camisetas	20170515	106	7,76	2,99	3,62	6,24	7,99	9,33	11,85	15,99
Venca	Camisetas	20170515	1194	10,99	2,99	5,21	7,77	8,99	10,42	20,56	59,99
Zara	Camisetas	20170515	900	13,6	2,95	6,47	11,91	12,95	16,08	19,96	29,95

Con la ayuda de un patrón que nos dirá en todo momento en qué filas exactamente encontramos cada categoría para poder realizar correctamente cada hoja de Excel:

Etiquetas de fila	Cuenta de Categoria					patron
Abrigo	7	2	8	A2:A8	B8	Abrigo
Alpargata	6	9	14	A9:A14	B14	Alpargata
Americana	8	15	22	A15:A22	B22	Americana
Bamba	8	23	30	A23:A30	B30	Bamba
Bañador	7	31	37	A31:A37	B37	Bañador
Baño	7	38	44	A38:A44	B44	Baño
Bata	7	45	51	A45:A51	B34	Bata
Bermuda_Short	10	52	61	A52:A61	B61	Bermuda_Short
Bikini	5	62	66	A62:A66	B66	Bikini

Realizamos los gráficos necesarios, a partir de los datos obtenidos, y dejamos todo en blanco, como hoja a seguir para la macro.



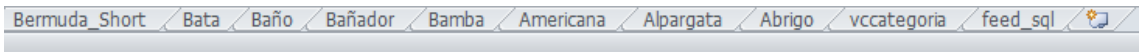
Esta es la hoja que nuestra macro simplemente duplicará y le pondrá a cada una de las hojas el nombre de la categoría definida en parámetros (patrón) y rellenará la primera columna de cada hoja, con las marcas que encuentra en cada categoría.

El CONSULTAV automáticamente se rellenará conforme se cree la nueva hoja, con los datos que encuentre de cada marca.

Ya solo queda escribir la macro:

```
Sub duplicador()
'
' duplicador Macro
'
' Acceso directo: CTRL+d
'


    Sheets("Parametros").Select
    nombre_hoja = Range("G1").Value

    For I = 2 To 62
        Sheets("Parametros").Select
        hoja_destino = Cells(I, 7).Value
        Cells(I, 8).Value = hoja_destino
        rangoCopia = Cells(I, 5).Value
        Sheets(nombre_hoja).Select
        Sheets(nombre_hoja).Copy Before:=Sheets(1)
        Sheets(nombre_hoja + " (2)").Select
        Sheets(nombre_hoja + " (2)").Name = hoja_destino
        Sheets("vccategoria").Select
        Range(rangoCopia).Select
        Application.CutCopyMode = False
        Selection.Copy
        Sheets(hoja_destino).Select
        Range("A3").Select
        Selection.PasteSpecial Paste:=xlPasteValues,
Operation:=xlNone, SkipBlanks _
        :=False, Transpose:=False

        Next I


End Sub
```

Tras esto solo nos quedará, asignarle una combinación de teclas, y ejecutar.



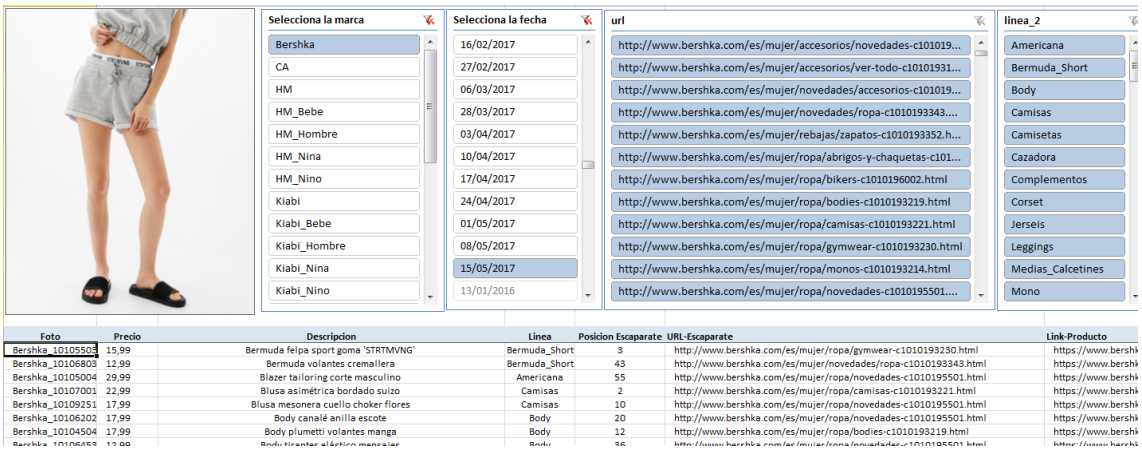
4.3.3 Generando informes adicionales

Una vez tenemos el informe preparado y automatizado, con los datos que tenemos se nos pueden ocurrir muchos más informes distintos y atractivos para la empresa.

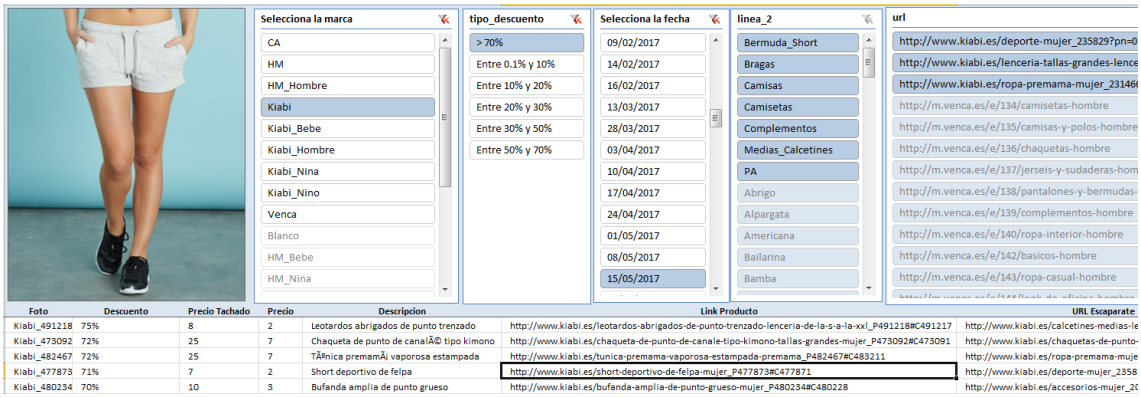
Por ejemplo, un informe sobre las novedades diarias de cada marca, o un informe sobre las ofertas diarias sobre las mismas.

A estos informes incluso al ser más reducidos les podemos añadir las imágenes de cada producto que ha salido nuevo o que esta de oferta, para que ya no tengan ni que entrar en la web para verlo.

Ejemplo de salida de novedades:



Ejemplo de salida de ofertas:



En el informe de novedades, podremos ver cada marca que artículos ha puesto nuevos en su web, por fecha, categoría, escaparate…

En el de ofertas, podremos ver cada marca qué productos tiene en un rango de descuentos (p.e. >70%, entre 50 y 70…), por escaparate, por categoría, su precio actual y antes de descontar…

Para estos ambos excels nos basta con una pestaña, en la cual tendremos los datos extraídos de sus respectivas tablas de SQL apartados donde no se vean (u ocultos) y la tabla que los leerá a partir de fórmulas dirigidas a las celdas en la que se encuentran los datos reales.

Las imágenes las insertaremos con una macro VBA desde una carpeta, en la cual las descargaremos mediante scripts en Python.

La macro quedaría tal que así:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)

On Error Resume Next
If Not Intersect(Target, Range("A2:A600")) Is Nothing Then
    Image1.Picture = _
        LoadPicture(ActiveWorkbook.Path & "\imagenes\" & Target &
".jpg")
    Image1.AutoSize = False
    Image1.PictureSizeMode = fmPictureSizeModeStretch

End If

End Sub
```

La cual nos permite, con tan solo movernos por la columna de “Foto”, hacer match con el nombre de la imagen en la carpeta, e insertarla automáticamente arriba a la izquierda a tiempo real mientras nos desplazamos por cada una.

Y quedaría un script para descargarnos las imágenes, por ejemplo, el de novedades, que se muestra a continuación:

```
# -*- coding: utf-8 -*-
"""

@author: joelsanchez
"""

import urllib
import re
import pyodbc
import requests
from bs4 import BeautifulSoup
from datetime import datetime
pmName = 'ordered_set'
pm = __import__(pmName)

timeIni = datetime.now()

headers = {'User-agent': 'Mozilla/5.0'}

#conectamos a SQL
cnxn = pyodbc.connect('DRIVER={SQL Server Native Client
10.0};SERVER=KOSMOS;DATABASE=MKT_CRAWLER;UID=[usuario];PWD=[contraseña]')

cursor = cnxn.cursor()


sql = "SELECT distinct Link from mkt_crawler.dbo.crawlerNovedades where
brand not in ('HM_Hombre', 'HM_Nino', 'HM_Nina', 'HM_Bebe', 'Kiabi_Hombre',
'Kiabi_Nino', 'Kiabi_Nina', 'Kiabi_Bebe', 'Venca_Hombre', 'Venca_Nino',
'Venca_Nina', 'Venca_Bebe') and execution_time like '%20170515%'"
urls = set()
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Fetch all the rows in a list of lists.
   results = cursor.fetchall()
   #para cada fila de SQL cogemos la informacion (links) y creamos un set de
links
   for row in results:
      new_urls = {row[0]}
      urls.update(new_urls)
      # Now print fetched result
      print "URL to crawl=%s" %(urls)

except:
   print "Error: unable to fecth data"

cursor.close()

fallos = set()

contMax = len(urls)
cont = 0
```

Analizando a la competencia

```
while urls:
    #sacamos el primer escaparate
    current_url = urls.pop()
    cont += 1
    #nos guardamos el link del escaparate para añadirlo a SQL al final
    print "Crawling current_url : %s"%(current_url)
    print "Crawling %s/%s"%(cont,contMax)
    try:
        webpage = requests.get(current_url, headers=headers)
        htmltext = webpage.content
        soup= BeautifulSoup(htmltext)

        if 'bershka' in current_url:
            list_of_wrd = current_url.split('-')
            codigoP = list_of_wrd[len(list_of_wrd)-1]
            codigoP = codigoP[codigoP.index('p')+1:codigoP.index('.')]
            coodigoP = int(codigoP)
            codigoP = str(codigoP)
            codigoP = codigoP.lstrip('0')
            saveIn =
"M:/INFORMES_BI/novedades_competencia/imagenes/Bershka_" + codigoP + ".jpg"
            linkImg = soup.findAll('meta', { "property" :
"og:image"})[0]['content']
            try:
                response = requests.get(linkImg)
                if response.status_code == 200:
                    f = open(saveIn, 'wb')
                    f.write(response.content)
                    f.close()
            except Exception:
                urllib.urlretrieve(linkImg, saveIn)

        elif 'c-and-a' in current_url:
            codigoP = soup.find('div', { "class" : "detailsTabContent
detailsTabContentDetail1 active"})['data-productid']
            coodigoP = int(codigoP)
            codigoP = str(codigoP)
            codigoP = codigoP.lstrip('0')
            saveIn = "M:/INFORMES_BI/novedades_competencia/imagenes/CA_" +
codigoP + ".jpg"
            linkImg = soup.find('div', { "class" : "imageItem"})
            linkImg = linkImg.find('a')['href']
            try:
                response = requests.get(linkImg)
                if response.status_code == 200:
                    f = open(saveIn, 'wb')
                    f.write(response.content)
                    f.close()
            except Exception:
                urllib.urlretrieve(linkImg, saveIn)
        elif 'blanco.com' in current_url:
            list_url = current_url.split('/')
            codigoP = list_url[len(list_url)-1].split('-')
            codigoP = codigoP[len(codigoP)-1]
            codigoP = codigoP[:codigoP.index('.')]
            codigoP = codigoP.lstrip('0')
            saveIn = "M:/INFORMES_BI/novedades_competencia/imagenes/Blanco_"
+ codigoP + ".jpg"
            linkImg = soup.findAll('meta', { "property" :
"og:image"})[0]['content']
```

Analizando a la competencia

```
try:
                response = requests.get(linkImg)
                if response.status_code == 200:
                    f = open(saveIn, 'wb')
                    f.write(response.content)
                    f.close()
            except Exception:
                urllib.urlretrieve(linkImg, saveIn)

        elif 'hm.com' in current_url:
            list_of_words = htmltext.split()
            codigoP = list_of_words[list_of_words.index("'baseProductCode'")
- 1]
            codigoP = codigoP[14:]
            codigoP = codigoP.replace("'","")
            codigoP = codigoP.replace(",","")
            coodigoP = int(codigoP)
            codigoP = str(codigoP)
            codigoP = codigoP.lstrip('0')
            saveIn = "M:/INFORMES_BI/novedades_competencia/imagenes/HM_" +
codigoP + ".jpg"
            linkImg= soup.find('img', { "class" : "product-detail-thumbnail-
image"})['src']
            linkImg= 'http:'+linkImg
            try:
                response = requests.get(linkImg)
                if response.status_code == 200:
                    f = open(saveIn, 'wb')
                    f.write(response.content)
                    f.close()
            except Exception:
                urllib.urlretrieve(linkImg, saveIn)
        elif 'kiabi' in current_url:
            codigoP = soup.find('meta', { 'property' : "og:url"})['content']
#codigo producto
            codigoP = codigoP[-6:]
            codigoP = int(re.search(r'\d+', codigoP).group())
            codigoP = str(codigoP)
            codigoP = codigoP.lstrip('0')
            saveIn = "M:/INFORMES_BI/novedades_competencia/imagenes/Kiabi_"
+ codigoP + ".jpg"
            linkImg = soup.find('li', { "class" : "exists"})
            linkImg = linkImg.find('a')['href']
            try:
                response = requests.get(linkImg)
                if response.status_code == 200:
                    f = open(saveIn, 'wb')
                    f.write(response.content)
                    f.close()
            except Exception:
                urllib.urlretrieve(linkImg, saveIn)
        elif 'mango' in current_url:
            codigoP = soup.find_all('input',
id="id_producto_hidden")[0]['value']
            coodigoP = int(codigoP)
            codigoP = str(codigoP)
            codigoP = codigoP.lstrip('0')
            saveIn = "M:/INFORMES_BI/novedades_competencia/imagenes/Mango_"
+ codigoP + ".jpg"
```

```
linkImg = soup.find_all('meta', property="og:image")[0]['content']
            if 'http:' not in linkImg:
                linkImg = soup.find_all('input',
id="zoomFicha_hid")[0]['value']
            try:
                response = requests.get(linkImg)
                if response.status_code == 200:
                    f = open(saveIn, 'wb')
                    f.write(response.content)
                    f.close()
            except Exception:
                urllib.urlretrieve(linkImg, saveIn)
        elif 'shana' in current_url:
            codigoP = soup.find('input', { 'name' : "id_product"})['value']
            coodigoP = int(codigoP)
            codigoP = str(codigoP)
            codigoP = codigoP.lstrip('0')
            saveIn = "M:/INFORMES_BI/novedades_competencia/imagenes/Shana_"
+ codigoP + ".jpg"
            try:
                linkImg = soup.find_all('img', xonload="onLoad()")[0]['src']
            except Exception:
                linkImg = soup.find_all('img', {'class' :
'current'})[0]['src']
            try:
                response = requests.get(linkImg)
                if response.status_code == 200:
                    f = open(saveIn, 'wb')
                    f.write(response.content)
                    f.close()
            except Exception:
                urllib.urlretrieve(linkImg, saveIn)
        elif 'venca' in current_url:
            try:
                codigoP = soup.find_all('input', { 'id' :
"modelId"})[0]['value']
                coodigoP = int(codigoP)
                codigoP = str(codigoP)
                codigoP = codigoP.lstrip('0')
            except IndexError:
                list_html = htmltext.split(':')
                codigoP = list_html[list_html.index('"Product","productID"')
+ 1]
                codigoP = codigoP.split(',')
                codigoP = codigoP[0]
                codigoP = codigoP.replace('"',"")
                coodigoP = int(codigoP)
                codigoP = str(codigoP)
                codigoP = codigoP.lstrip('0')
            saveIn = "M:/INFORMES_BI/novedades_competencia/imagenes/Venca_"
+ codigoP + ".jpg"
            linkImg                    =                    soup.find_all('meta',
property="og:image")[0]['content']
```

```
try:
                response = requests.get(linkImg)
                if response.status_code == 200:
                    f = open(saveIn, 'wb')
                    f.write(response.content)
                    f.close()
            except Exception:
                urllib.urlretrieve(linkImg, saveIn)
        elif 'zara' in current_url:
            try:
                #codigo producto
                codigoP = soup.find_all('input', { 'name' :
"parentId"})[0]['value']
            except Exception:
                list_url = current_url.split('-')
                codigoP = list_url[len(list_url)-1]
                codigoP = codigoP[codigoP.index('p')+1:codigoP.index('.')]
            coodigoP = int(codigoP)
            codigoP = str(codigoP)
            codigoP = codigoP.lstrip('0')
            saveIn = "M:/INFORMES_BI/novedades_competencia/imagenes/Zara_" +
codigoP + ".jpg"
            linkImg = soup.find('div', { "id" : "main-images"})
            linkImg = linkImg.find('a')['href']
            linkImg = 'http:' + linkImg
            #linkImg = linkImg[:linkImg.index("?")]
            response = requests.get(linkImg)
            if response.status_code == 200:
                f = open(saveIn, 'wb')
                f.write(response.content)
                f.close()
        elif 'lefties' in current_url:
            list_htmls = htmltext.split('"')
            try:
                #codigo producto
                codigoP = list_htmls[list_htmls.index('productId')+2]
            except Exception:
                list_url = current_url.split('-')
                codigoP = list_url[len(list_url)-1]
                codigoP = codigoP[codigoP.index('p')+1:codigoP.index('.')]
            coodigoP = int(codigoP)
            codigoP = str(codigoP)
            codigoP = codigoP.lstrip('0')
            saveIn =
"M:/INFORMES_BI/novedades_competencia/imagenes/Lefties_" + codigoP + ".jpg"
            linkImg = soup.find_all('meta',
property="og:image")[0]['content']

            try:
                response = requests.get(linkImg)
                if response.status_code == 200:
                    f = open(saveIn, 'wb')
                    f.write(response.content)
                    f.close()
            except Exception:
                urllib.urlretrieve(linkImg, saveIn)
```

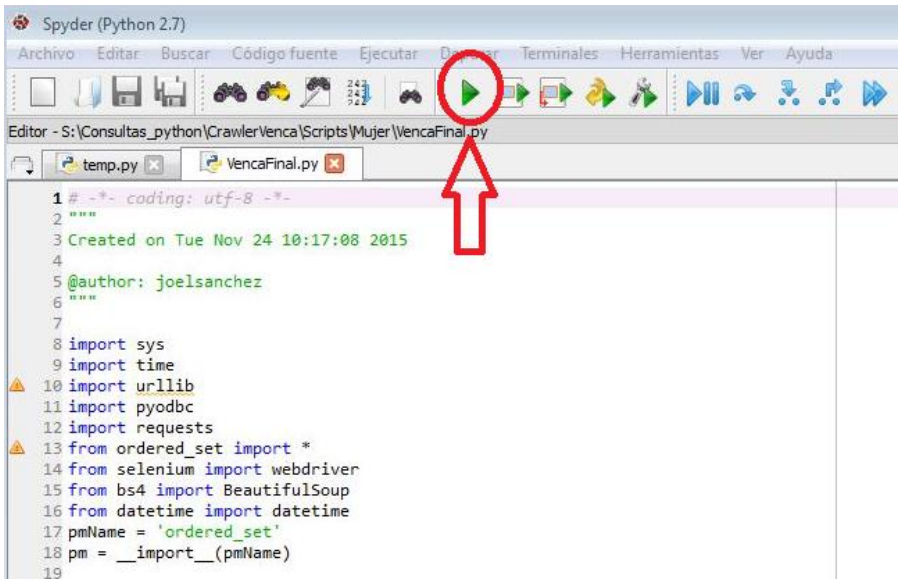
```
elif 'stradivarius' in current_url:
            list_c = htmltext.split(' ' and ';')
            codigoP = list_c[list_c.index('\r\n\tinditex.iCountryCode =
"ES"') + 5]
            codigoP = codigoP.replace("\r","")
            codigoP = codigoP.replace("\n","")
            codigoP = codigoP[codigoP.index('=') + 1:]
            codigoP = codigoP.replace(" ","")
            coodigoP = int(codigoP)
            codigoP = str(codigoP)
            codigoP = codigoP.lstrip('0')
            saveIn =
"M:/INFORMES_BI/novedades_competencia/imagenes/Stradivarius_" + codigoP +
".jpg"
            linkImg = soup.find_all('meta',
property="og:image")[0]['content']
            try:
                linkImg = linkImg.replace('https','http')
            except Exception:
                pass
            response = requests.get(linkImg)
            if response.status_code == 200:
                f = open(saveIn, 'wb')
                f.write(response.content)
                f.close()


    except Exception:
        fallos.add(current_url)
        pass

print "Lista de fallos: %s"%(fallos)
```

5. EJECUCIÓN Y DISTRIBUCIÓN

5.1 Ejecución

Tenemos dos tipos de ejecución posibles:

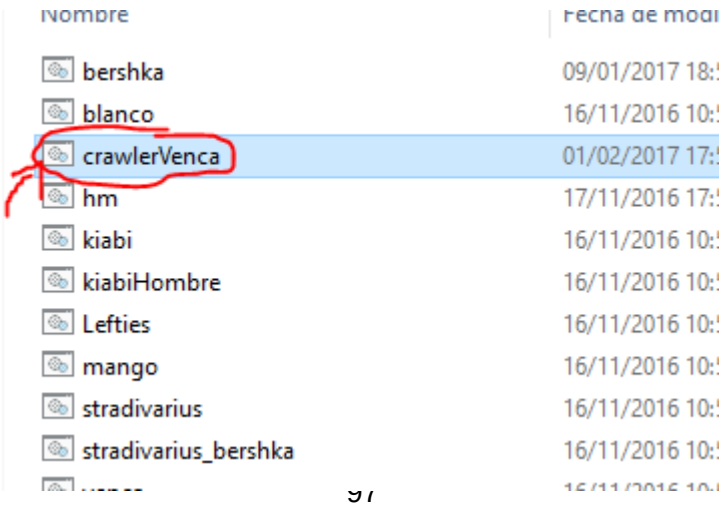
- **Manual**: Siempre tenemos la posibilidad de abrir Spyder y ejecutar desde el propio programa cualquier script.



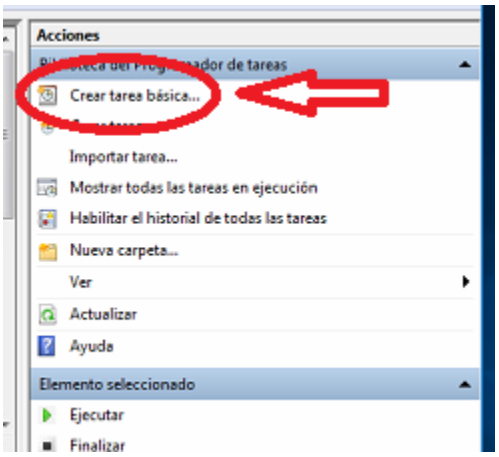
La ventaja de esto es que podemos ejecutarlo en cualquier momento que nos lo pidan, e incluso modificar el script para analizar solo un escaparate en concreto o cualquier información que quieran obtener al momento.

También podemos ejecutar de manera manual, sin necesidad de abrir Spyder, un archivo .bat con el script:

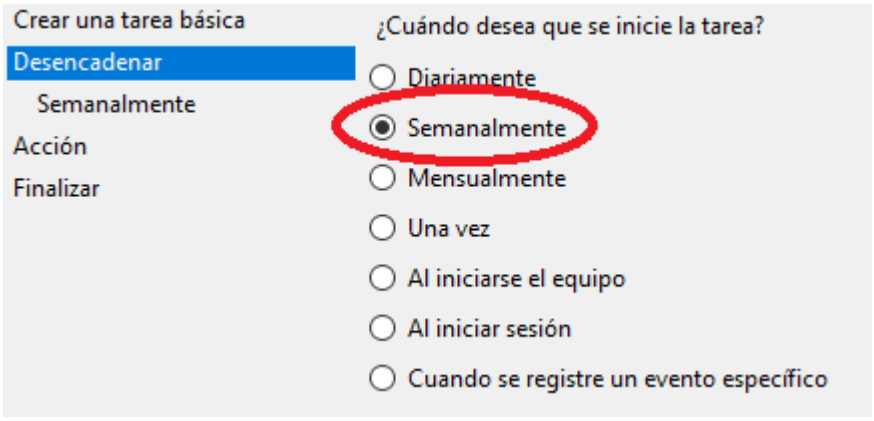
```
start cmd.exe /K python S:\Consultas_python\CrawlerVenca\Scripts\Mujer\VencaFinal.py
```

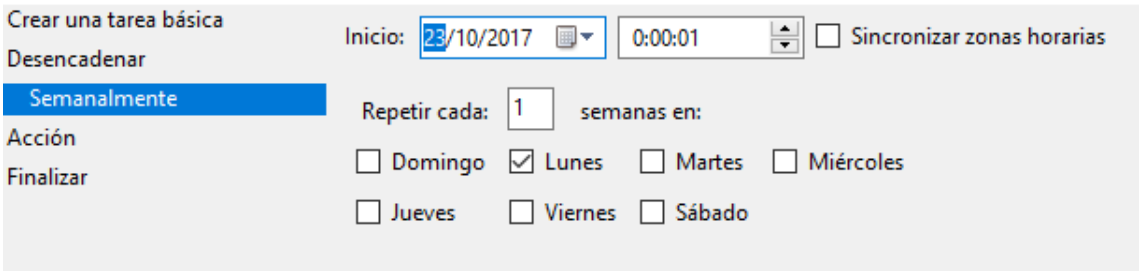


- **Automática**: Mediante el programador de tareas de Windows. Para ello abrimos “Programador de tareas”, y creamos una tarea básica:

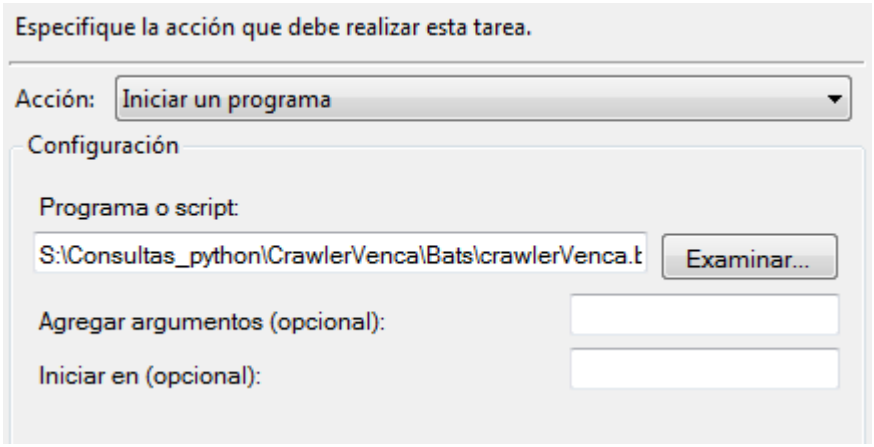


Tras esto, debemos elegir un nombre para la tarea y si queremos una descripción. A continuación, nos pedirá cuándo se va a ejecutar esta tarea, si diariamente, semanalmente, mensualmente, una sola vez, etc.





Por último, en acción seleccionamos “Iniciar un programa”, y aquí buscamos el archivo .bat de los scripts, podemos tener todos los programas en un .bat o generar tantos .bat y tareas como scripts queramos ejecutar y añadirlos de uno en uno.



Y el archivo .bat quedaría tal que así:

```
start cmd.exe /K python
S:\Consultas_python\CrawlerVenca\Scripts\Mujer\KiabiFinal.py
timeout 2
start cmd.exe /K python
S:\Consultas_python\CrawlerVenca\Scripts\Mujer\VencaFinal.py
timeout 2
start cmd.exe /K python
S:\Consultas_python\CrawlerVenca\Scripts\Hombre\Bershka.py
timeout 3
start cmd.exe /K python
"S:\Consultas_python\CrawlerVenca\Scripts\Menores\HM.py"
timeout 3
start cmd.exe /K python
"S:\Consultas_python\CrawlerVenca\Scripts\Menores\Stradivarius.py"
timeout 3
start cmd.exe /K python
"S:\Consultas_python\CrawlerVenca\Scripts\Menores\Lefties.py"
timeout 3
start cmd.exe /K python
S:\Consultas_python\CrawlerVenca\Scripts\Mujer\ZaraFinal.py
timeout 2
start cmd.exe /K python
S:\Consultas_python\CrawlerVenca\Scripts\Mujer\CAfinal.py
timeout 2
start cmd.exe /K python
S:\Consultas_python\CrawlerVenca\Scripts\Mujer\ShanaFinal.py
```

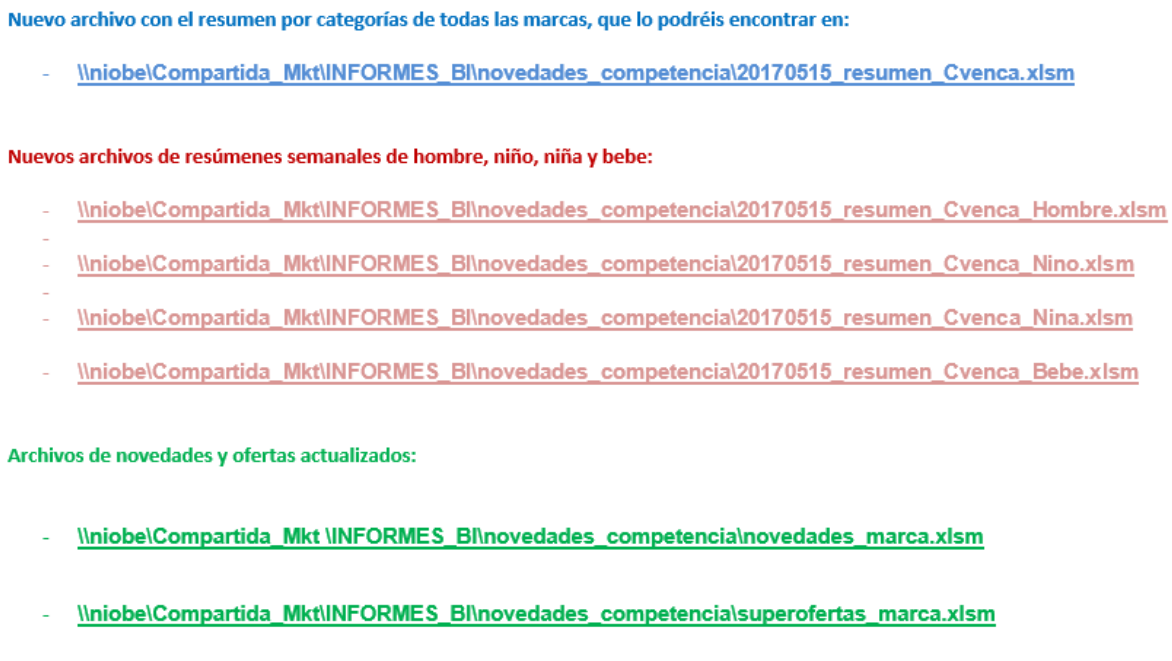
5.2 Distribución

Y, por último, una vez hemos visto como se generan los informes y se ejecutan los scripts, solo queda la distribución de estos informes a toda la empresa.

Hay infinidad de formas posibles: carpetas compartidas, en la “nube”, por e-mail, por FTP, etc.

Utilizaremos dos de ellas: una será carpetas compartidas con aviso por e-mail, y la otra será aprendiendo una plataforma completamente nueva para mí, pero muy importante para la empresa, que es Pentaho. Una plataforma capaz de automatizar probablemente cualquier proceso que se nos pueda ocurrir, y que hará que tan solo con un doble click nos mueva automáticamente los ficheros a un FTP, los renombre y los mueva de carpeta o cree una copia de ellos.

La primera de ellas no requiere demasiada explicación, una vez cada lunes tenemos los informes hechos, los copiamos a una carpeta compartida por los departamentos que los vayan a utilizar, y finalmente mandamos un e-mail con las direcciones a estas carpetas para que los puedan abrir con tan solo un click:



Nuevo archivo con el resumen por categorías de todas las marcas, que lo podréis encontrar en:

- \\niobe\Compartida_Mkt\INFORMES_BI\novedades_competencia\20170515_resumen_Cvenca.xlsm

Nuevos archivos de resúmenes semanales de hombre, niño, niña y bebe:

- \\niobe\Compartida_Mkt\INFORMES_BI\novedades_competencia\20170515_resumen_Cvenca_Hombre.xlsm
- \\niobe\Compartida_Mkt\INFORMES_BI\novedades_competencia\20170515_resumen_Cvenca_Nino.xlsm
- \\niobe\Compartida_Mkt\INFORMES_BI\novedades_competencia\20170515_resumen_Cvenca_Nina.xlsm
- \\niobe\Compartida_Mkt\INFORMES_BI\novedades_competencia\20170515_resumen_Cvenca_Bebe.xlsm

Archivos de novedades y ofertas actualizados:

- \\niobe\Compartida_Mkt \INFORMES_BI\novedades_competencia\novedades_marca.xlsm
- \\niobe\Compartida_Mkt\INFORMES_BI\novedades_competencia\superofertas_marca.xlsm

5.2.1 Distribución con Pentaho

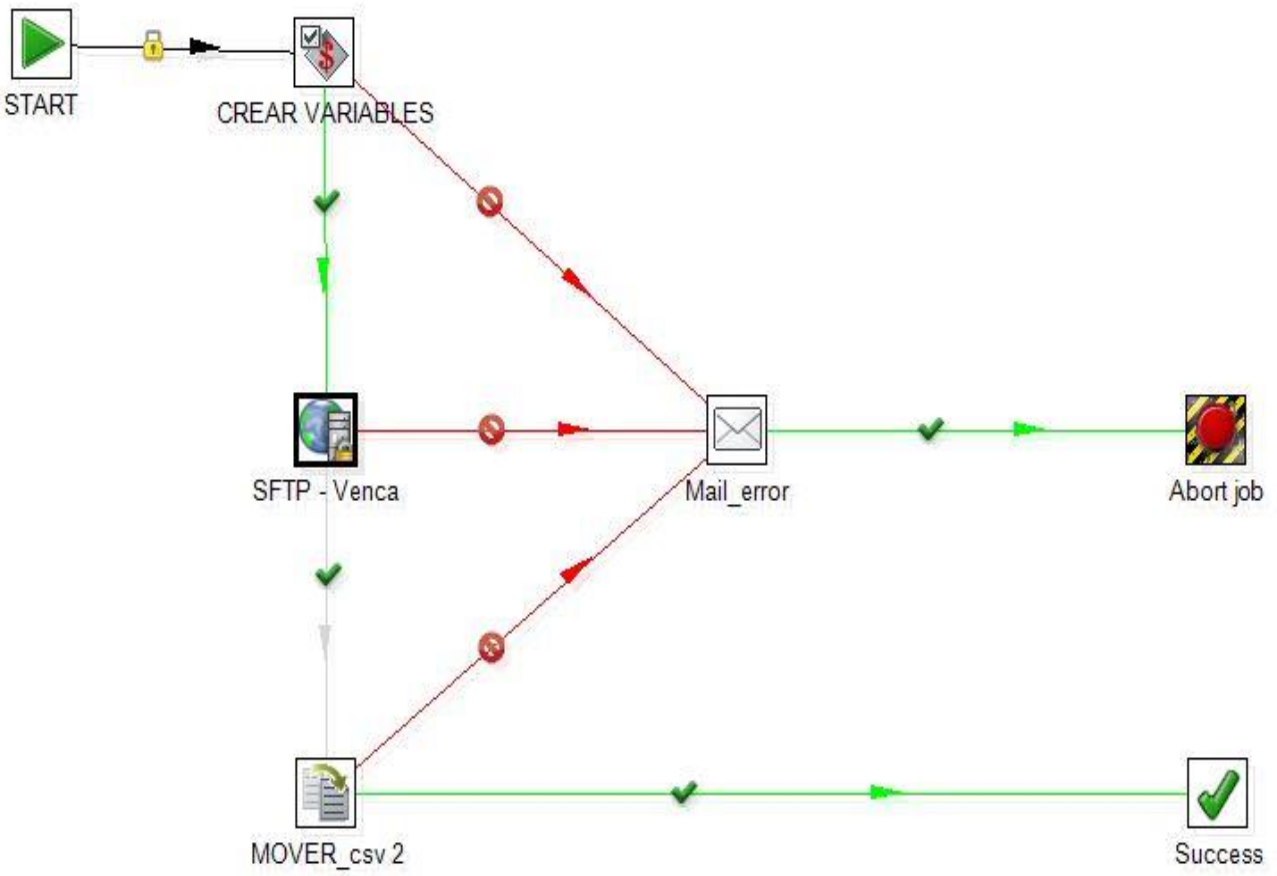
1. Crear proceso ETL

A través de este proceso ETL realizaré diferentes pasos para finalmente incorporar los datos a nuestro SFTP:

1. Crearé las variables con las rutas de salida y destino para los archivos.
2. Conexión al SFTP para dejar los archivos.
3. Mover los ficheros a la carpeta “Tratado”
4. En caso de cualquier fallo en los pasos anteriores, se mandará un email.

Todos estos pasos se realizan a través de lo que en la jerga llaman JOB (trabajo).

El Job es el siguiente:



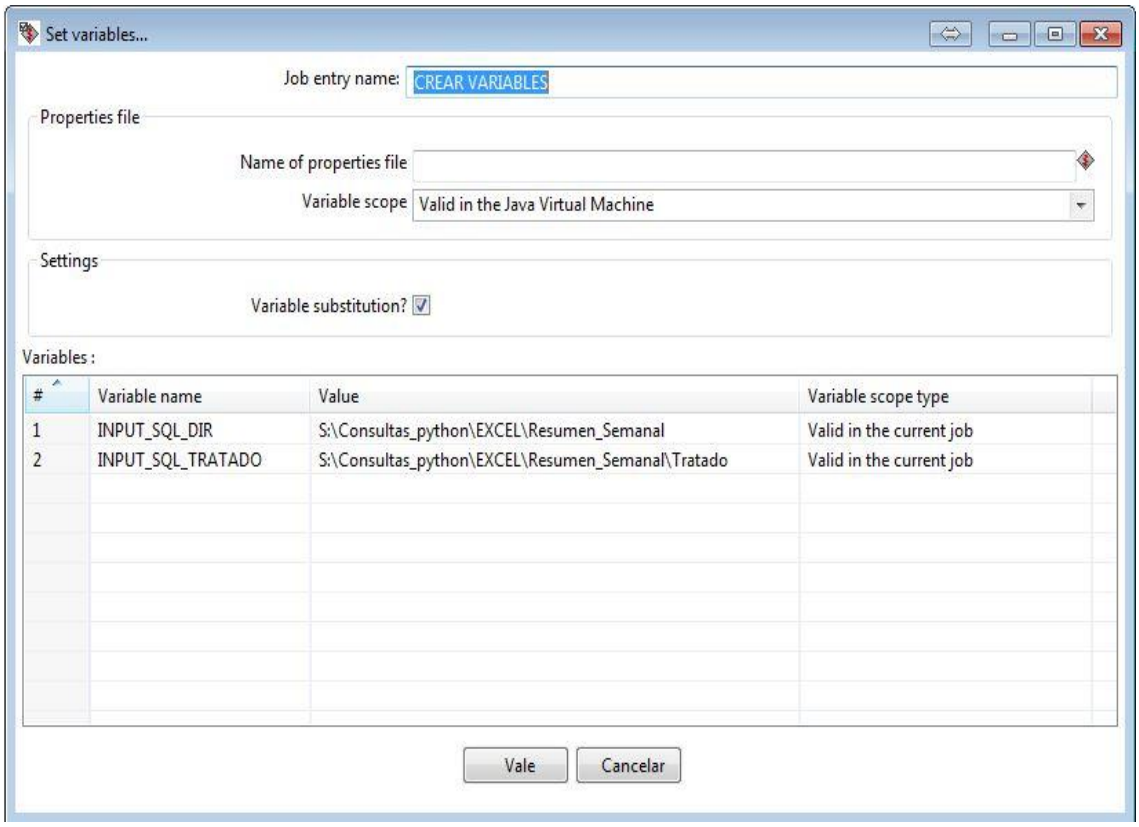
A continuación, se muestra cuáles son los pasos que seguir:

1. **START:**

   Cada Job debe empezar con un Start

2. **Crear Variables:**

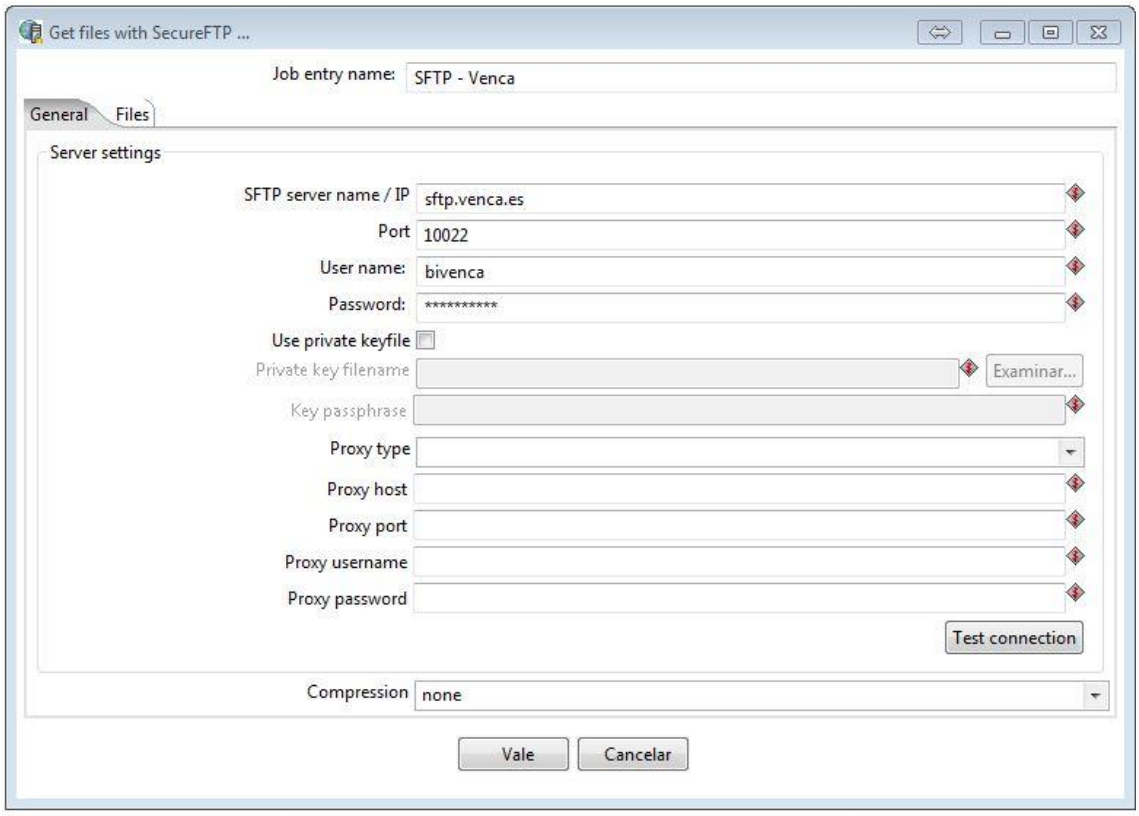
   El primer paso, para seguir una cierta coherencia entre los varios procesos ETL que se deberán de crear para cada plataforma, es la creación de variables. Estas variables nos indicarán donde se ubicarán nuestros ficheros a incorporar antes y después de ser tratados:



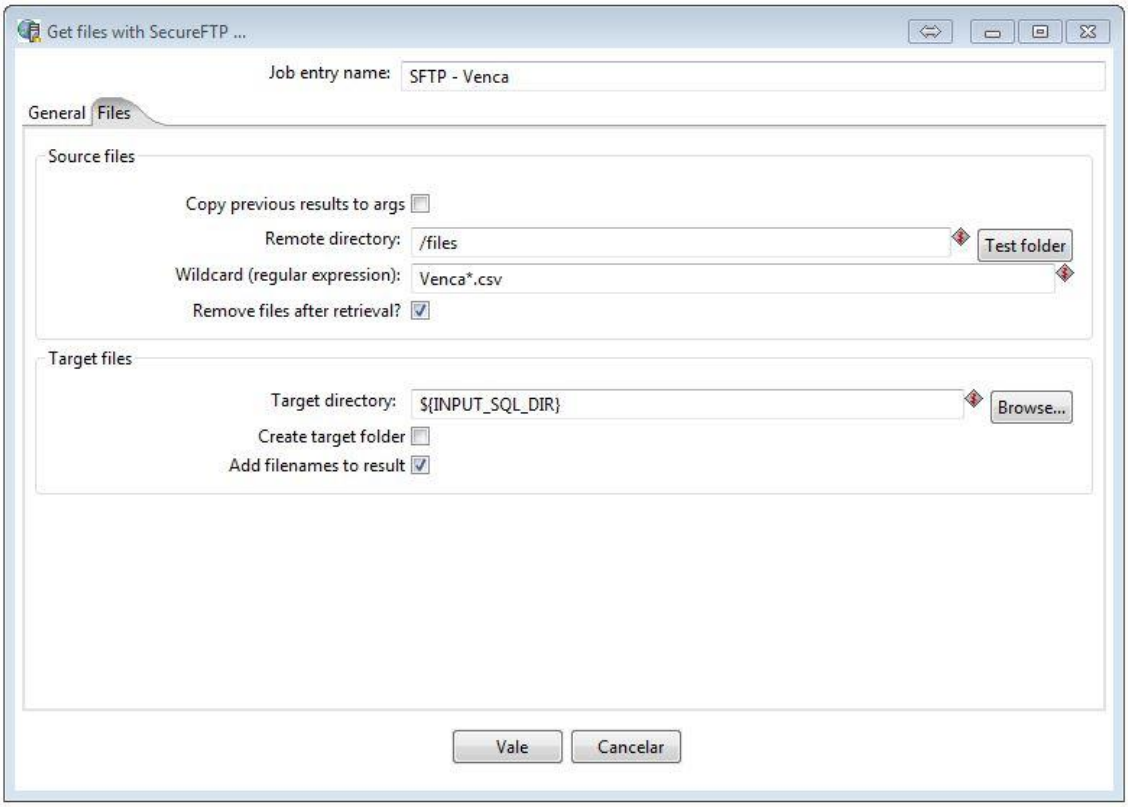
3. **Dejar los ficheros en el SFTP:**

   El siguiente paso es el de la conexión al SFTP para poder depositar los ficheros que contienen el resumen semanal, las novedades y las ofertas.

Analizando a la competencia

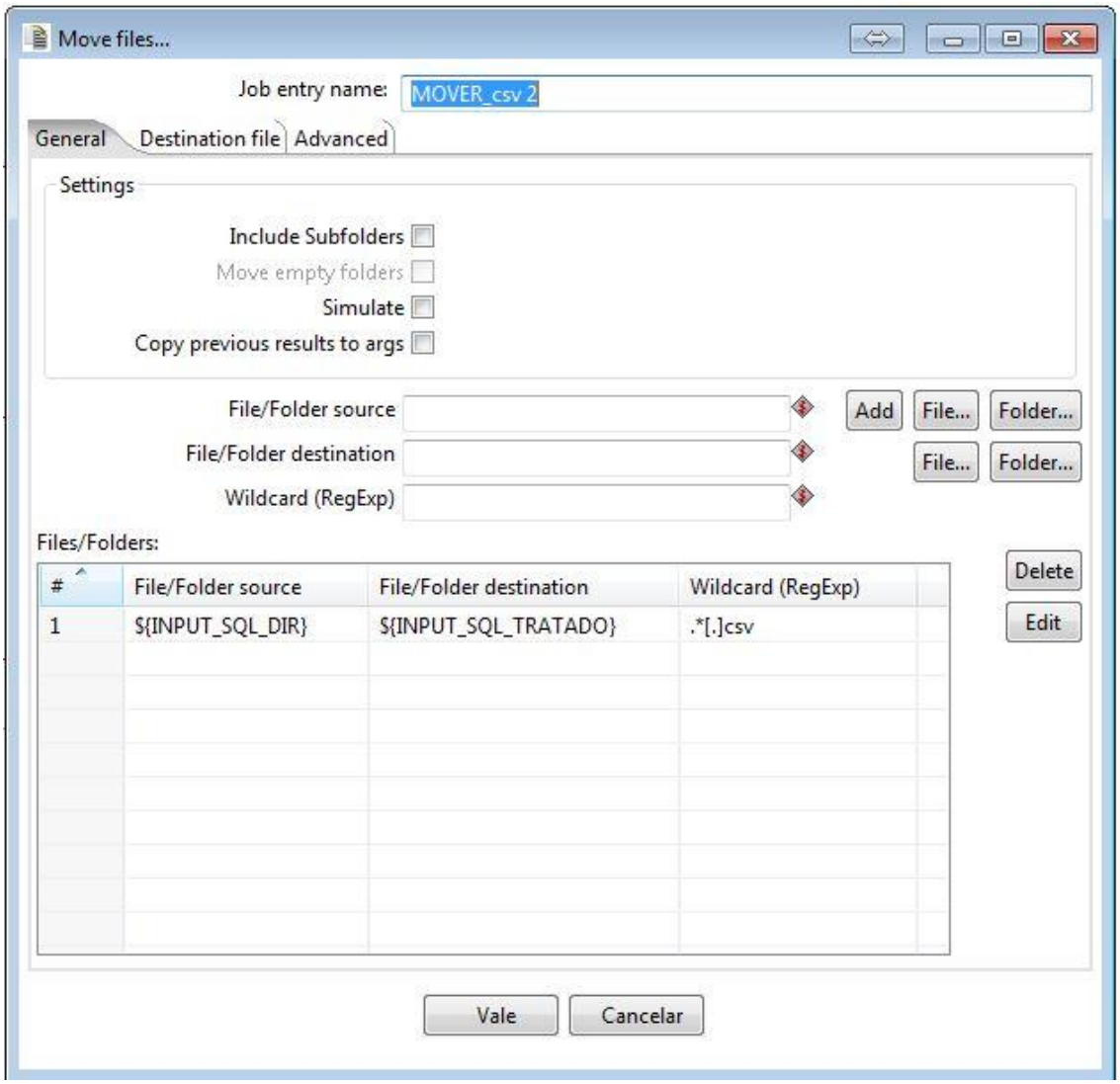


En este paso se especifica los datos de conexión al servidor SFTP, en que carpeta están los ficheros a mover y a través de la variable creada en el paso anterior se detalla el directorio en el cual debe cogerlos.



4. Mover ficheros:

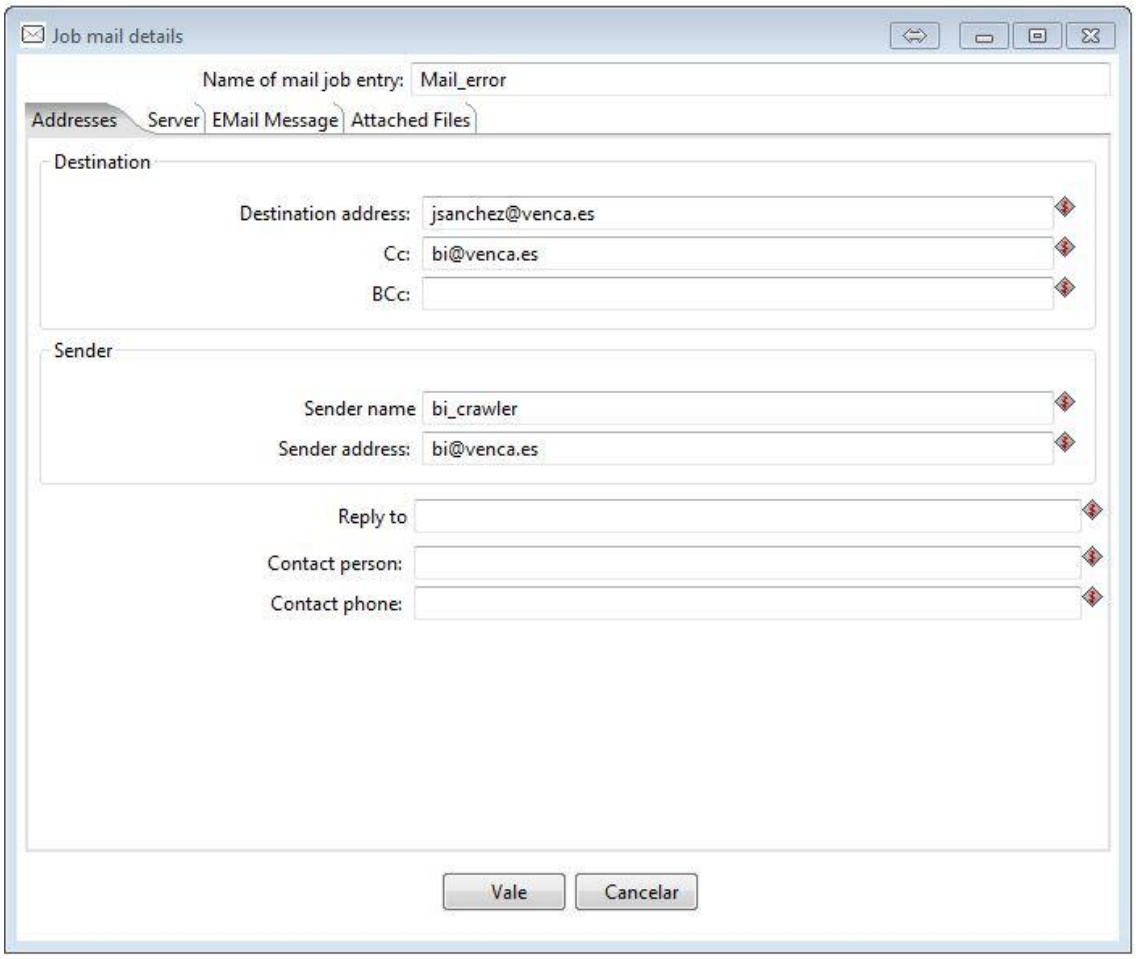
Este es el penúltimo paso del Job. En este caso, lo que hará es mover los ficheros que encuentre en la carpeta a “Tratado”. De esta forma, si vuelve a ejecutar el Job no repetirá ficheros en el momento de incorporarlos al SFTP.

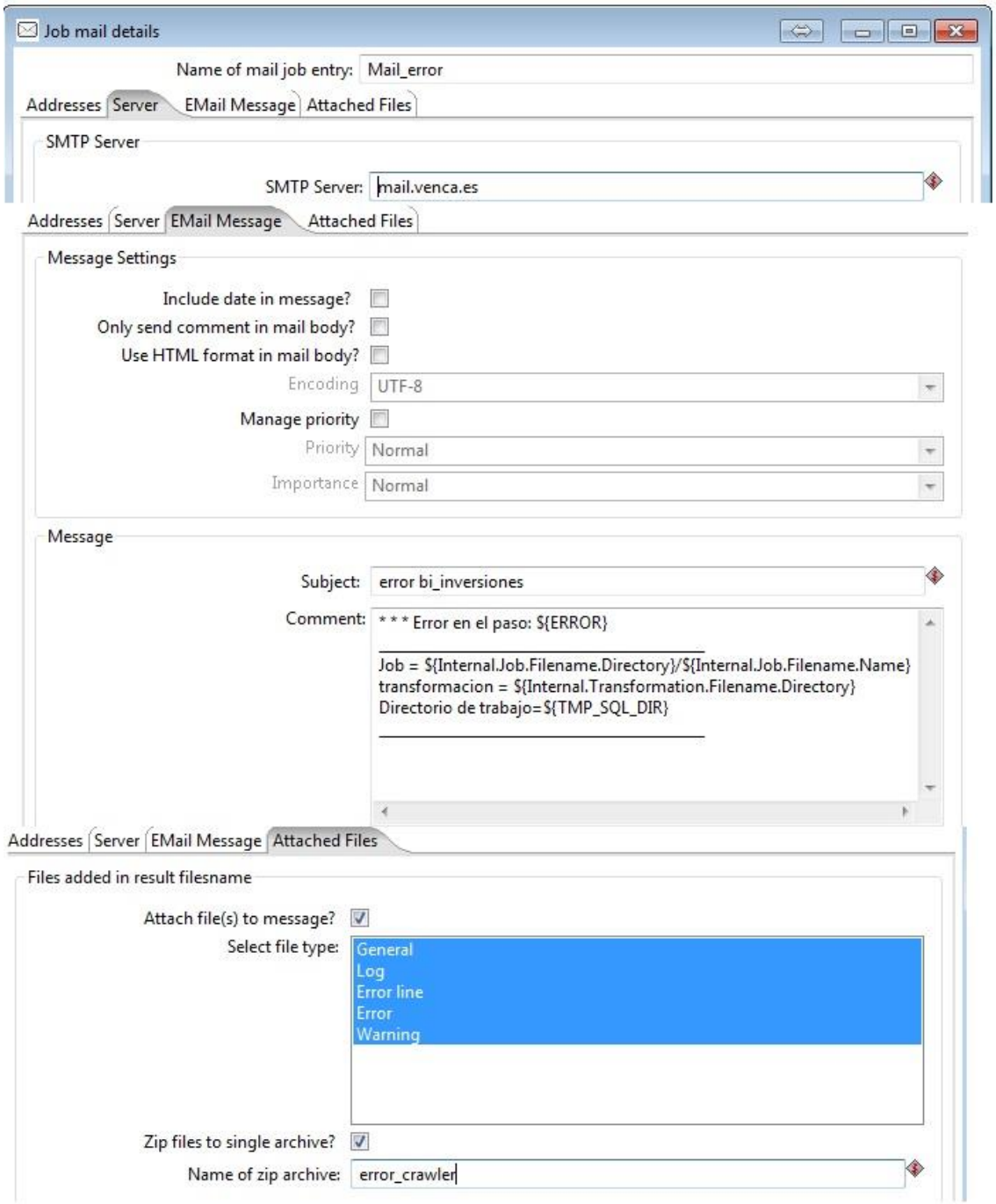


5. MAIL:

En el caso que la conexión al SFTP o la descarga del fichero no se haya completado satisfactoriamente se genera un email de error, y una vez éste sea enviado aborte el trabajo.

En los detalles del email se aconseja poner como sujeto el nombre del trabajo que ha dado el error, y como archivos adjuntos los logs y errores de éste.




Job mail details
Name of mail job entry: Mail_error
Addresses Server EMail Message Attached Files
SMTP Server
SMTP Server: mail.venca.es
Addresses Server EMail Message Attached Files
Message Settings
Include date in message?
Only send comment in mail body?
Use HTML format in mail body?
Encoding UTF-8
Manage priority
Priority Normal
Importance Normal
Message
Subject: error bi_inversiones
Comment: * * * Error en el paso: ${ERROR}
Job = ${Internal.Job.Filename.Directory}/${Internal.Job.Filename.Name}
transformacion = ${Internal.Transformation.Filename.Directory}
Directorio de trabajo=${TMP_SQL_DIR}
Addresses Server EMail Message Attached Files
Files added in result filesname
Attach file(s) to message?
Select file type:
General
Log
Error line
Error
Warning
Zip files to single archive?
Name of zip archive: error_crawler

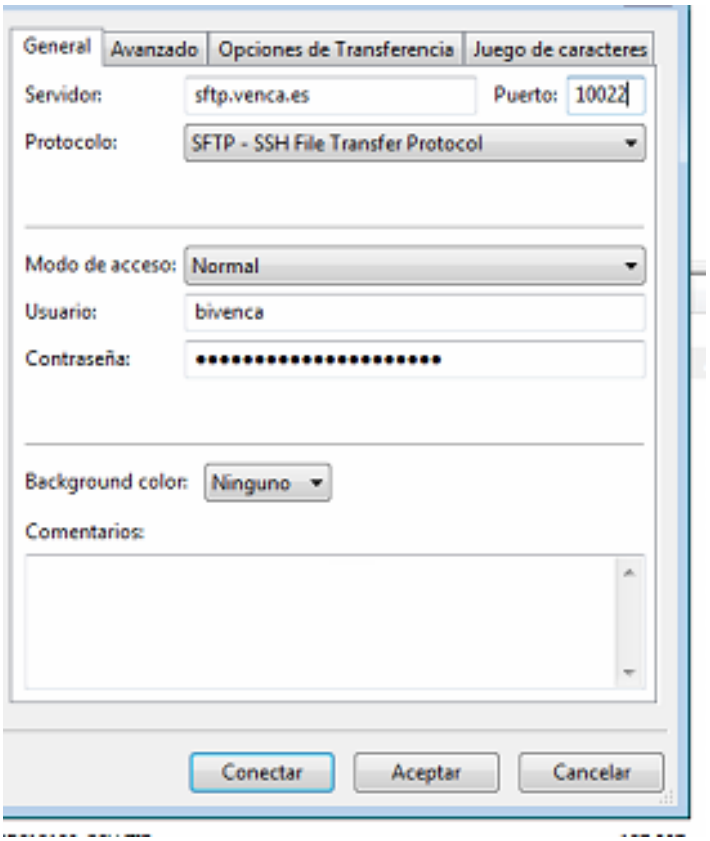
2. Ejecución:

Para la ejecución, al igual que para los scripts, generaremos un archivo .bat que contendrá lo siguiente:

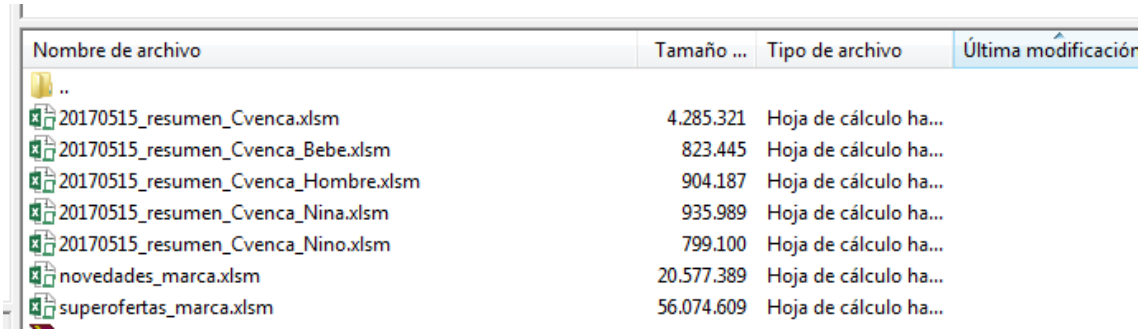
```
Echo ================================================================
rem DIRECTORIO=H:\BI\run\PDF

rem ----------------------
set tDate=%DATE%
set tTime=%TIME%
rem
ECHO INICI %tDate%  %tTime%
ECHO INICI %tDate%  %tTime%  >
S:\Consultas_python\EXCEL\Resumen_Semanal\log_bat\informes_tras.txt

rem cmd /C KOFAX_TO_SP.bat %*
echo
====\KOFAX_TO_SP=====================================================
==================
C:\data-integration\kitchen /level=Minimal/param=nom_fitxer=""
/file=S:\Consultas_python\EXCEL\Resumen_Semanal\InformesToSFTP.kjb %
*>
S:\Consultas_python\EXCEL\Resumen_Semanal\log_bat\informes_tras.log
2>
S:\Consultas_python\EXCEL\Resumen_Semanal\log_bat\informes_tras_err.
txt



set tDate=%DATE%
set tTime=%TIME%
ECHO FINAL %tDate%  %tTime%  >>
S:\Consultas_python\EXCEL\Resumen_Semanal\informes_tras.txt
ECHO FINAL %tDate%  %tTime%
rem SLEEP 5
```

3. Comprobación SFTP con FileZilla:

Conexión:



Ficheros:

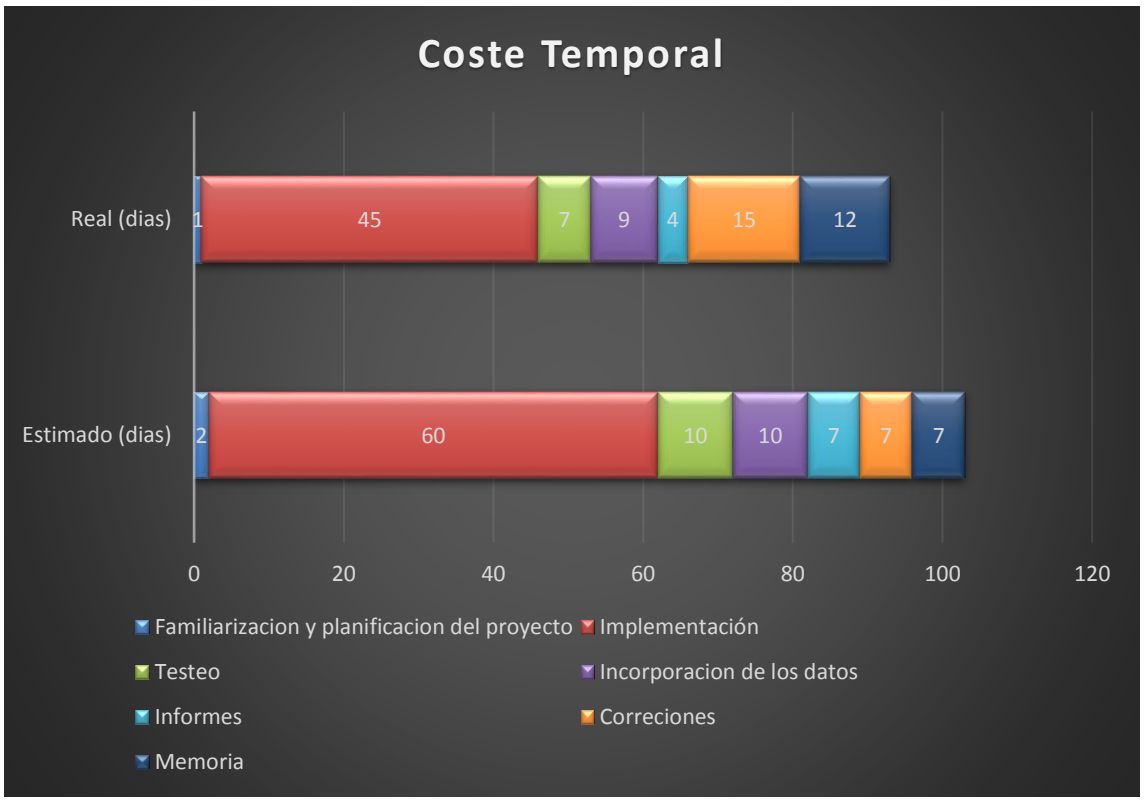


6. PLANIFICACIÓN FINAL Y ANÁLISIS ECONÓMICO

6.1 Coste temporal

La siguiente tabla muestra los distintos costes temporales divididos en las diferentes tareas a ejercer en el proyecto y por tiempo estimado tanto por la empresa acordado conmigo, tanto por el real que finalmente fue.

Tarea	Estimado (días)	Real (días)
Familiarización y planificación del proyecto	2	1
Implementación	60	45
Testeo	10	7
Incorporación de los datos	10	9
Informes	7	4
Correcciones	7	15
Memoria	7	12
TOTAL	103	93
Meses	3,4	3,1
Horas	824	744



Como podemos ver en la tabla, el proyecto se finalizó antes de lo estimado. Esto es debido mayormente a la dificultad que se le dio a la implementación del proyecto, ya que era programar unos scripts que nadie antes en la empresa se había puesto a programar, y en mi caso, era aprender un lenguaje nuevo de programación para poder realizarlos. Esta programación se acabó en 1 mes y medio, y se pensaron 2 meses.

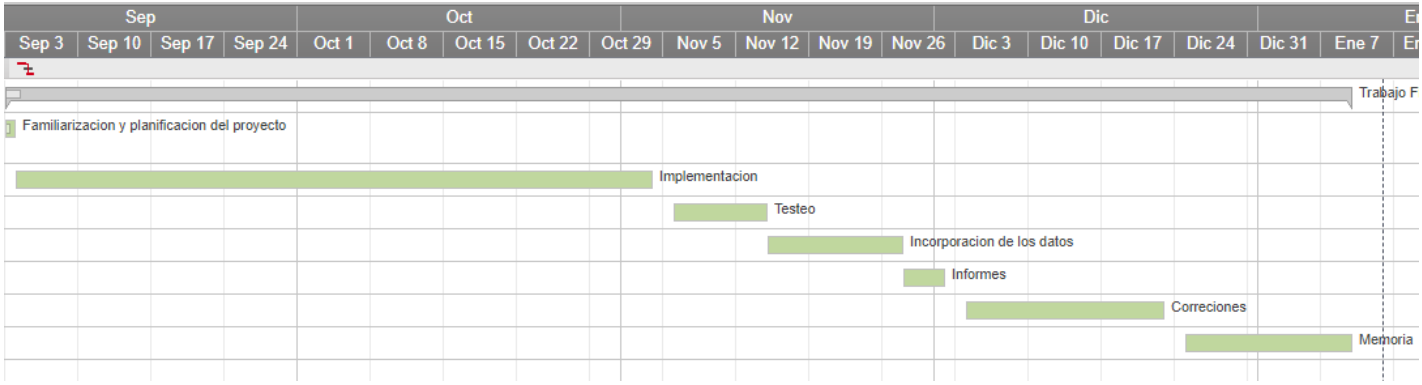
Por otro lado, la otra gran diferencia de tiempo viene en las correcciones, ya que se pensó que una vez realizado el proyecto al completo, scripts, test, la incorporación en bases de datos, gestión de las bases con SQL y la generación de informes, corregir posibles fallos iba a ser trivial y no iba a requerir más de una semana. La variación de 8 días viene dada a que conforme pasaban los días y se iban ejecutando los scripts, encontramos que las webs podían cambiar su código fuente, tenía que llenar los scripts de “try catch” para poder dar distintas posibilidades para extraer un dato y prever el cambio de la web, también vimos de la manera que crecían las bases de datos y como ralentizaban las consultas, tuve que dividir en tablas más pequeñas los datos. Y todo esto fue lo que hizo estar más tiempo del previsto corrigiendo errores.

Analizando a la competencia

En total el proyecto se realizó en 3,1 meses frente a los 3,4 previstos, que son 744 horas reales contra 824 previstas, un 10% más rápido de lo que se preveía. 10 días antes exactamente.

Diagrama temporal de Gantt:

Nombre de la tarea	Fecha de inicio	Fecha de finaliza...	Duración
Trabajo Final de Grado	**03/09/17**	**09/01/18**	**93d**
Familiarizacion y planificacion del proyecto	03/09/17	03/09/17	1d
Implementacion	04/09/17	03/11/17	45d
Testeo	06/11/17	14/11/17	7d
Incorporacion de los datos	15/11/17	27/11/17	9d
Informes	28/11/17	01/12/17	4d
Correciones	04/12/17	22/12/17	15d
Memoria	25/12/17	09/01/18	12d



6.2 Coste económico

Vamos a considerar por un lado el coste del personal, en esta casi 1 persona y con lo que sería sueldo de programador, que está actualmente entorno a unos 35 euros, y por otro lado de infraestructura, ya que necesitaremos por un lado un buen ordenador donde programar y gestionar las bases de datos y los informes, y, por otro lado, necesitaremos unos servidores donde almacenar todos los datos.

Personal	Estimado	Real
Horas	824	744
Coste total	28 840	26 040

Aquí podemos ver como finalmente he acabado saliendo más barato de lo estimado, ya que realice el proyecto en un total de 10 días menos del tiempo estimado por la empresa. Suponiendo este adelanto, un ahorro de 2 800 € y consiguiente ganancia de confianza y demostración a la empresa que ellos están por delante de cualquier bien económico personal.

A la hora del coste de infraestructura para un proyecto de este calibre, tenemos que analizar detenidamente los pros y contras de elegir un servidor u otro, ya que es una inversión muy importante. A continuación, analizamos las 3 opciones disponibles:

- **On-premises**: Sería la opción en físico, ya sea particular o en la empresa. En el caso de que la empresa ya disponga de un servidor con espacio y poder de procesamiento suficiente para correr el proyecto, no habría problema. En caso de ser particular o no disponer de ello, deberíamos de empezar a analizar que necesitamos y el coste que tendría.
    - Pros:
        - Precio: Pagas exactamente por lo que quieres y vas a utilizar.
        - Puedes aprovechar infraestructura y caudal.
        - Seguridad personalizada.

  - Contras:
    - Necesitas una dimensión mínima → Tienes que ir redimensionando en caso de que tus bases de datos vayan creciendo más de lo esperado o necesiten de más poder de procesamiento (+RAM, +CPU, +HDD).
    - Tienes que llevar tú el control de back ups y seguridad.
    - Mantenimiento.
  - Precio: ± 6 000 €.
    - Servidor + Licencias (Windows + SQL)
- **Cloud – IaaS**: Infraestructura como servicio, es decir, contratamos el servidor entero en Cloud.
  - Pros:
    - Conectividad.
    - Pagas por uso.
    - Redimensionamiento on demand.
    - Licencias incluidas.
  - Contras:
    - Precio elevado para un uso 24/7.
    - Back up y mantenimiento se pagan a parte.
  - Precio: ± 2 400 € anuales.
- **Cloud – SaaS**: Software como servicio, en esta opción contratamos el servicio de SQL por un lado, y por otro la máquina virtual encargada de apuntar a este servidor SQL. La mejor opción para SQL es Azure, ya que es la opción oficial de Microsoft, la cual nos garantiza back ups y tener el producto siempre actualizado con sus últimas versiones. Para la opción de máquina virtual, tras analizar el mercado tenemos a DigitalOcean con precios y variantes bastante más asequibles que cualquier otro.
  - Pros:
    - Ídem IaaS.
    - Mantenimiento, actualizaciones y back ups asegurados.
  - Contras:
    - Sigue teniendo un precio elevado para un uso 24/7.
  - Precio: ± 940 € anuales.

Infraestructura	**Coste**
Ordenador Personal ***(Intel i7-7700 / 16GB / 2TB HDD / 240 SSD M.2 / GTX 1060 6GB)*** ***+*** ***(teclado + ratón Logitech MK120)*** ***+*** ***(2x Benq GW2270HE 21.5" LED Full HD)***	1 628,58
Servidor	6 000
Coste Total	7 628,58

Lo que daría un coste total de proyecto de **33 668,58 €***.

**Calculado en base a escoger un servidor por ejemplo físico. El presupuesto final puede variar según el acuerdo y análisis que se haga con la empresa/cliente tanto para la infraestructura como para el salario.*

7. CONCLUSIONES

En primer lugar, me veo obligado a señalar como, a nivel individual, este proyecto me ha apasionado desde el primer momento y ha supuesto no sólo algo que he disfrutado hasta el final, sino también la base de nuevos aprendizajes, y es que durante la realización he aprendido un lenguaje de programación innovador. Durante la carrera opté por una especialización más centrada en bases de datos y SQL, y este proyecto me ha ayudado a profundizar todavía más dentro del mismo ámbito. Con todo, el nivel de SQL adquirido durante el proyecto ha sido realmente notable no solo a nivel de programación y escritura, sino también a nivel de administración de bases de datos gestionando la mía propia dentro de un servidor de Venca, generando y gestionando mis tablas, etc. Asimismo, he aprendido a manejar Excel de una manera que jamás habría imaginado posible; he descubierto trucos, fórmulas, su propio lenguaje de programación VBA y he hecho cosas que hasta el momento desconocía que se podían hacer con dicho programa. Por último, he aprendido una plataforma nueva que desconocía por completo, como es Pentaho, pero que para la empresa es importantísima ya que automatiza la mayoría de sus procesos; la cantidad de cosas que se pueden automatizar y hacer con dicho programa me dejaron totalmente anonadado.

En resumen, a nivel personal he adquirido muchísimo conocimiento de gran valor para mi desarrollo personal y que a su vez me ha servido para acabar de especializarme en aquello que, precisamente, ya empecé en la carrera: el SQL. He descubierto por completo mundos nuevos como son Python, Excel y Pentaho. Me ha hecho marcarme nuevas metas, como podría ser hacer un diseño web y app y, por qué no decirlo, me ha hecho ganarme un puesto en la empresa.

A nivel laboral, el proyecto ha servido para sustituir a un analizador que tenían contratado previamente a nivel externo, pagando X dinero cada mes. No sólo ha supuesto un ahorro a nivel económico, sino que, además, se ha podido disponer de los informes cuando y donde se han requerido, se ha ampliado el campo de análisis añadiendo más marcas de las que tenían contratadas, así como otros países, secciones, etc. Han pasado de analizar 4 marcas españolas en sección mujer, a 11 marcas en el mismo ámbito, marcas francesas, otras de hombre, niño, niña, bebés, marcas de perfumería, tallas grandes, etc.

Mirando al futuro, este proyecto me ha servido para ganarme la confianza de la empresa, entrando a formar parte de ella por tiempo indefinido. El proyecto seguirá

creciendo, ya que añadiremos marcas nuevas u otras secciones. Asimismo, buscaré formas de rastrear cada vez más eficientes y, quizás, elabore una pequeña web donde poder mostrar los resultados en vivo y de manera mucho más visual y en la que cualquier persona en la empresa pueda navegar libremente y analizar los datos que ellos quieran en el momento que quieran. Por último, a nivel personal y con tal de evolucionar individualmente, me gustaría desarrollar una app en la cual se pudieran tener las mejores ofertas de las tiendas, llegasen notificaciones de un producto en concreto que se desee, se pudiese seguir su evolución de precios, etc., ya sea tanto a nivel de análisis de empresa como a nivel de uso personal.

A modo de cierre, no puedo acabar esta memoria sin agradecer a todo mi equipo, Marina Albaladejo (adjunta de Business Intelligence), Jordi Morató (Controller Marketing - BI) y especialmente a Luis Martínez (Responsable de BI), por haber confiado en mí y haber permitido que realice este proyecto y por haberme acogido en su equipo por lo que esperemos que sea mucho tiempo más. De verdad, muchísimas gracias.

8. BIBLIOGAFIA

Beaulieu, A. (2009). *Learning SQL*. 2nd ed. Beijing [etc.]: O'Reilly.

Forta, B. (1999). *Sams teach yourself SQL in 10 minutes*. 4th ed. Indiana: Sams.

Kouzis-Loukas, D. (2016). *Learning Scrapy*. 1st ed. Birmingham: Packt.

Lutz, M. and Ascher, D. (1999). *Learning Python*. 5th ed. USA: O'Reilly & Associates

Matthes, E. (2015). *Python crash course*: *A Hands-On, Project-Based Introduction to Programming.* 1st ed. San Francisco: No Starch Press.

Mitchell, R. (2016). *Web Scraping with Python: Collecting Data from the Modern Web*. 4th ed. Beijin [etc.]: Oreilly & Associates Inc.

Theregister.co.uk. (2018). *DigitalOcean cuts cloud server pricing to stop rivals eating its lunch*. [online] Available at: https://www.theregister.co.uk/2018/01/18/digitalocean_cuts_cloud_server_pricing_to_meet_and_beat_competitive_pressure/.

Walkenbach, J. (2004). *Excel VBA Programming For Dummies*. 4th ed. Indiana: Wiley Publishing.

9. ANEXOS

Bershka

```
# -*- coding: utf-8 -*-
"""

@author: joelsanchez
"""

import time
import re
import pyodbc
import requests
from ordered_set import *
from selenium import webdriver
from bs4 import BeautifulSoup
from datetime import datetime
pmName = 'ordered_set'
pm = __import__(pmName)
i = datetime.now()

cnxn = pyodbc.connect('DRIVER={SQL Server Native Client
10.0};SERVER=KOSMOS;DATABASE=MKT_CRAWLER;UID=[user];PWD=[password]')

cursor = cnxn.cursor()

#Funcion para saber si un string contiene numeros, en algunas webs nos
sirve para comprobar si un link es de producto o no
def hasNumbers(inputString):
    return bool(re.search(r'\d', inputString))


headers = {'User-agent': 'Mozilla/5.0'}
fecha_ejecucion = time.strftime("%Y%m%d %H:%M:%S")
timeIni = datetime.now()
fecha_error = time.strftime("%Y%m%d")
fallos = set()
cont = 0
topMax = 20
final = 0
client = 'Venca'
brand = 'Bershka'

filename= "S:/Consultas_python/CrawlerVenca/Errores/" + fecha_error + "_"
+ brand + "_Errors.txt"
```

```
"""
connected = False
while connected == False:

    try:
        driver = webdriver.Chrome()#'C:/chromedriver.exe'
        time.sleep(3)
        #urls = {'http://www.bershka.com/es/'}
        urls = {'http://www.bershka.com/es/mujer/ropa/abrigos-y-chaquetas-
c1010193212.html'}
        current_url = urls.pop()
        driver.get(current_url)
        time.sleep(10)
        driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")
        time.sleep(3)
        html2 = driver.execute_script("return
document.documentElement.innerHTML;")
        soup= BeautifulSoup(html2)
        connected = True

    except Exception:
        pass

word = '/es/mujer'

for tag in soup.findAll('a', href=True):
    print tag['href']
    product = hasNumbers(tag['href'])
    product2 = bool(re.search('p[0-9]{3}', tag['href']))
    if product is True and word in tag['href'] and '*' not in tag['href']:
        #urls.add(tag['href'])
        list_url = tag['href'].split('/')
        if len(list_url) <= 7 and 'tarjeta-regalo' not in tag['href'] and
'/novedades/mujer' not in tag['href'] and product2 is False:
            urls.add(tag['href'])
print urls

driver.quit()
"""

urls = {'http://www.bershka.com/es/mujer/ropa/camisas-c1010193221.html',
'http://www.bershka.com/es/mujer/inspiraci%C3%B3n/%23bershkastyle-
c1010193528.html', 'http://www.bershka.com/es/mujer/zapatos/ver-todo-
c1010193192.html', 'http://www.bershka.com/es/mujer/ropa/novedades-
c1010195501.html', 'http://www.bershka.com/es/mujer/rebajas/ropa-
c1010193350.html', 'http://www.bershka.com/es/mujer/accesorios/bolsos-
c1010193138.html', 'http://www.bershka.com/es/mujer/ropa/b%C3%A1sicos-
c1010193227.html',
'http://www.bershka.com/es/mujer/accesorios/bisuter%C3%ADa-
c1010193140.html' }
```

```
escaparates = len(urls)
contE = 0
#bucle de las urls previamente sacadas del SQL, hasta que no se recorran
todas, seguimos
while urls:

    #sacamos la primera URL a recorrer
    current_url = urls.pop()

    #nos quedamos con su nombre ya que sera informacion para la base de
datos
    main_url = current_url
    print "Crawling current_url : %s"%(current_url)
    #sacamos su codigo HTML, simulando que somos Mozilla
    webpage = requests.get(current_url, headers=headers)
    htmltext = webpage.content
    soup= BeautifulSoup(webpage.content)
    word = 'html'
    sinDuplicados = OrderedSet()
    sinDuplicados.clear()
    contador = OrderedSet()
    contador.clear()
    #recorremos la URL y extraemos todos los links, filtramos por los de
producto mirando si contiene HTML y numeros, y lo guardamos en un set, para
no tener repetidos y recorrer luego
    for tag in soup.findAll('a', href=True):
        product = hasNumbers(tag['href'])
        if product is True and word in tag['href'] and '*' not in
tag['href'] and '/-5/land' not in tag['href']:
            sinDuplicados.add(tag['href'])
            cont +=1
            contador.add(cont)
    cont = 0
    #bucle para recorrer los productos que hemos extraido previamente
    current_cont = len(sinDuplicados)
    productos = len(sinDuplicados)
    contP = 0


    while sinDuplicados:
        try:
            current_url = sinDuplicados.pop()
            print "Crawling", main_url
            timeP = datetime.now()
            webpage = requests.get(current_url, headers=headers)
            htmltext = webpage.content
            soup= BeautifulSoup(htmltext)
            print "Parsing", current_url

            #current_cont = contador.pop()
            if current_cont <= topMax:
                top = 'TRUE'
            else:
                top = 'FALSE'
```

```
            list_of_words = current_url.split('/')
            categoria = list_of_words[list_of_words.index('es') + 2]
            subcategoria = list_of_words[list_of_words.index('es') + 3]
            print "Categoria: %s -
Subcategoria: %s"%(categoria,subcategoria)

            nombreP = soup.find('title').getText()
            nombreP = nombreP.split('-')[0]
            print "Nombre: %s"%(nombreP)

            list_html2 = htmltext.split('"')
            precioP = list_html2[list_html2.index('priceCurrency') + 6]
            precioRealP = precioP
            print "Precio: %s"%(precioP)

           #link directo al producto
            linkP = current_url

            list_of_wrd = current_url.split('-')
            codigoP = list_of_wrd[len(list_of_wrd)-1]
            codigoP = codigoP[codigoP.index('p')+1:codigoP.index('.')]
            print "Codigo: %s"%(codigoP)


            #comprobamos si es novedad
            if 'novedades' in main_url:
                novedades = 'TRUE'
            else:
                novedades = 'FALSE'
            print "Novedades: %s"%(novedades)

            ncolores = 0
            print "Colores: %s"%(ncolores)
            tallaMin = 0
            tallaMax = 0
            print "tallaMin: %s"%(tallaMin)
            print "tallaMax: %s"%(tallaMax)

            var_sql = """insert into crawlerVenca
(Client,Category,Brand,SubBrand,Description,PriceBeforeDiscount,ActualPrice,
URL,Link,MinSize,MaxSize,CodeProd,ID_Presentation,Execution_Time,Top20,Crawl
_Day,Crawl_Date,NEW) values (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"""
            cursor.execute(var_sql,
(client,categoria,brand,subcategoria,nombreP,precioRealP,precioP,main_url,li
nkP,tallaMin,tallaMax,codigoP,current_cont,fecha_ejecucion,top,timeIni,timeP
,novedades))
            cursor.commit()
            print "current cont %s/%s"%(current_cont,productos)
        except Exception,e:
            f=open(filename,"a")
            #write error to file:
            f.write("Error occured:\n" + current_url + " <------> " + str(e)
+ "\n") # some message specific to this iteration (page) should be added
here...
            #close error log file:
            f.close()
```

```
        current_cont += -1
        contP += 1
        print "Productos %s/%s"%(contP,productos)
        if contP == productos:
            final += 1
            print "Escaparate crawleado con exito"

    contE += 1
    print "Crawleado escaparate %s/%s"%(contE,escaparates)
    print "Con exito: %s/%s"%(final,escaparates)

print "Crawler de %s ha finalizado correctamente"%(brand)
print "Lista de fallos %s"%(fallos)
cursor.close()
```

Blanco

```
# -*- coding: utf-8 -*-
"""

@author: joelsanchez
"""

import sys
import time
import urllib
import pyodbc
import requests
import ctypes
import re
from ordered_set import *
from selenium import webdriver
from bs4 import BeautifulSoup
from datetime import datetime
pmName = 'ordered_set'
pm = __import__(pmName)
#i = datetime.now()

cnxn = pyodbc.connect('DRIVER={SQL Server Native Client
10.0};SERVER=KOSMOS;DATABASE=MKT_CRAWLER;UID=[user];PWD=[password]')

cursor = cnxn.cursor()


headers = {'User-agent': 'Mozilla/5.0'}

urls = {"http://www.blanco.com/es/es_es/"}
fecha_error = time.strftime("%Y%m%d")
fecha_ejecucion = time.strftime("%Y%m%d %H:%M:%S")
timeIni = datetime.now()
cont = 0
final = 0
topMax = 20
errores = 0
fallos = set()
client = 'Venca'
brand = 'Blanco'
filename= "S:/Consultas_python/CrawlerVenca/Errores/" + fecha_error + "_"
+ brand + "_Errors.txt"

word = '/es/es_es/'
word2 = '.html'

current_url = urls.pop()
webpage = requests.get(current_url, headers=headers)
htmltext = webpage.content
soup= BeautifulSoup(webpage.content)
#recorremos la URL y extraemos todos los links, filtramos por los de
producto mirando si contiene HTML y numeros, y lo guardamos en un set,
para no tener repetidos y recorrer luego
for tag in soup.findAll('a', href=True):
    if word in tag['href'] and word2 in tag['href']:
        urls.add(tag['href'])
```

```
escaparates = len(urls)
contE = 0
#bucle de escaparates, mientras queden links sigue haciendo
while urls:

    #sacamos el primer escaparate
    current_url = urls.pop()
    #nos quedamos con su link
    main_url = current_url
    print "Crawling current_url : %s"%(current_url)
    #entramos desde Mozilla y leemos la web y cogemos su codigo html
    webpage = requests.get(current_url, headers=headers)
    htmltext = webpage.content
    soup= BeautifulSoup(webpage.content)
    #word = 'html'
    sinDuplicados = OrderedSet()
    sinDuplicados.clear()
    contador = OrderedSet()
    contador.clear()
    for tag in soup.findAll('a', href=True):
        product = bool(re.search('-[0-9]{11}', tag['href']))
        if product == True:
            sinDuplicados.add(tag['href'])
            cont +=1

            #contador.add(cont)
    cont = 0
    current_cont = len(sinDuplicados)
    productos = len(sinDuplicados)
    contP = 0

    while sinDuplicados:

        try:
            current_url = sinDuplicados.pop()
            print "Crawling", main_url
            print "Parsing", current_url
            timeP = datetime.now()
            #current_cont = contador.pop()
            if current_cont <= topMax:
                top = 'TRUE'
            else:
                top = 'FALSE'

            webpage = requests.get(current_url, headers=headers)
            htmltext = webpage.content
            soup= BeautifulSoup(htmltext)

            list_url = main_url.split('/')

            try:
                categoria = list_url[len(list_url)-2]
                subcategoria = list_url[len(list_url)-1]
                subcategoria = subcategoria[:subcategoria.index('.')]
                if categoría == 'es_es':
                    categoría = list_url[len(list_url)-1]
                    categoría = categoría[:categoria.index('.')]
                    subcategoria = categoría
```

```
            except Exception:
                categoria ='Mujer'
                list_cat = current_url.split('/')
                subcat = list_cat[len(list_cat)-1]
                subcats = subcat.split('-')
                subcategoria = subcats[0]



            print "Categoria: %s"%(categoria)
            print "Subcategoria: %s"%(subcategoria)

            list_of_words = current_url.split('/')
            list_nombre = list_of_words[len(list_of_words)-1].split('-')
            nombreP = ' '.join(list_nombre[:-1])
            print "Nombre: %s"%(nombreP)

            list_htmls = htmltext.split('"')
            precioRealP = list_htmls[list_htmls.index('productOldPrice') +
1]
            precioRealP = precioRealP.replace(",","")
            precioRealP = precioRealP.replace(":","")
            precioP = list_htmls[list_htmls.index('productPrice')+1]
            precioP = precioP.replace(",","")
            precioP = precioP.replace(":","")

            print "Precio: %s"%(precioP)
            print "Precio sin descontar: %s"%(precioRealP)

            linkP = current_url

            list_url = current_url.split('/')
            codigoP = list_url[len(list_url)-1].split('-')
            codigoP = codigoP[len(codigoP)-1]
            codigoP = codigoP[:codigoP.index('.')]
            print "Codigo: %s"%(codigoP)

            contColores = 'class="color"'
            ncolores = htmltext.count(contColores)
            print "Colores: %s"%(ncolores)



            tallaMin = soup.find('span' , {'class' : "size"}).getText()
            tallaMax = soup.findAll('span' , {'class' : "size"})
            last_div = None
            for last_div in tallaMax:pass #bucle para recorrer todas las
tallas y coger la ultima
            if last_div:
                tallaMax = last_div.getText()
            if tallaMin =='u':
                tallaMin = 'Unica'
                tallaMax = 'Unica'

            print "tallaMin: %s"%(tallaMin)
            print "tallaMax: %s"%(tallaMax)
```

```
            if 'nuevo' in main_url:
                novedades='TRUE'
            else:
                novedades='FALSE'


            var_sql = """insert into crawlerVenca
(Client,Category,Brand,SubBrand,Description,PriceBeforeDiscount,ActualPrice,
URL,Link,MinSize,MaxSize,CodeProd,ID_Presentation,Execution_Time,Top20,Crawl
_Day,Crawl_Date,NEW,#colors) values
(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"""
            cursor.execute(var_sql,
(client,categoria,brand,subcategoria,nombreP,precioRealP,precioP,main_url,li
nkP,tallaMin,tallaMax,codigoP,current_cont,fecha_ejecucion,top,timeIni,timeP
,novedades,ncolores))
            cursor.commit()
            print "current cont %s/%s"%(current_cont,productos)

        except Exception,e:
            f=open(filename,"a")
            #write error to file:
            f.write("Error occured:\n" + current_url + " <------> " + str(e)
+ "\n") # some message specific to this iteration (page) should be added
here...
            #close error log file:
            f.close()
            fallos.add(current_url)


        current_cont += -1
        contP += 1
        print "Productos %s/%s"%(contP,productos)
        if contP == productos:
            final += 1
            print "Escaparate crawleado con exito"

    contE += 1
    print "Crawleado escaparate %s/%s"%(contE,escaparates)
    print "Con exito: %s/%s"%(final,escaparates)

print "Crawler de %s ha finalizado correctamente"%(brand)
print "Lista de fallos: %s"%(fallos)

cursor.close()
```

C&A

```
# -*- coding: utf-8 -*-
"""

@author: joelsanchez
"""

import time
import urllib2
import urllib
import re
import pyodbc
import requests
from ordered_set import *
from bs4 import BeautifulSoup
from datetime import datetime
pmName = 'ordered_set'
pm = __import__(pmName)
i = datetime.now()

headers = {'User-agent': 'Mozilla/5.0'}
#conectamos con el servidor SQL donde subiremos los datos
cnxn = pyodbc.connect('DRIVER={SQL Server Native Client
10.0};SERVER=KOSMOS;DATABASE=MKT_CRAWLER;UID=[user];PWD=[password]')

cursor = cnxn.cursor()

#Funcion para saber si un string contiene numeros, en algunas webs nos
sirve para comprobar si un link es de producto o no
def hasNumbers(inputString):
    return bool(re.search(r'\d', inputString))

fecha_error = time.strftime("%Y%m%d")
fallos = set()
new_urls = set()
fecha_ejecucion = time.strftime("%Y%m%d %H:%M:%S")
timeIni = datetime.now()
cont = 0
topMax = 20
final = 0
client = 'Venca'
brand = 'CA'

filename = "S:/Consultas_python/CrawlerVenca/Errores/" + fecha_error + "_"
+ brand + "_Errors.txt"



urls = {"https://www.c-and-a.com/es/es/shop/mujer","https://www.c-and-
a.com/es/es/shop/promocion-mujer"}
#url = "http://www2.hm.com"
word = '/shop/'
allP = "?pageSize=all"
sinDuplicados = OrderedSet()
```

```
while urls:
    current_url = urls.pop()
    #nos quedamos con su nombre ya que sera informacion para la base de
datos
    print "Crawling current_url : %s"%(current_url)
    #sacamos su codigo HTML
    webpage = requests.get(current_url, headers=headers)
    htmltext = webpage.content
    soup= BeautifulSoup(htmltext, "html.parser")
    #recorremos la URL y extraemos todos los links, filtramos por los de
producto mirando si contiene HTML y numeros, y lo guardamos en un set, para
no tener repetidos y recorrer luego
    for tag in soup.findAll('a', href=True):
        if hasNumbers(tag['href'])==False and word in tag['href']:
            new_url = tag['href']+allP
            new_urls.add(new_url)
    #print urls
escaparates = len(new_urls)
contE = 0
#bucle de las urls previamente sacadas del SQL, hasta que no se recorran
todas, seguimos
while new_urls:
    try:

        current_cont = 0
        #sacamos la primera URL a recorrer
        current_url = new_urls.pop()
        #nos quedamos con su nombre ya que sera informacion para la base de
datos
        main_url = current_url
        print "Crawling current_url : %s"%(current_url)
        #sacamos su codigo HTML
        webpage = requests.get(current_url, headers=headers)
        htmltext = webpage.content
        soup= BeautifulSoup(htmltext, "html.parser")
        sinDuplicados = OrderedSet()
        sinDuplicados.clear()
        #contador = OrderedSet()
        #contador.clear()
        word = 'html'
        #recorremos la URL y extraemos todos los links, filtramos por los de
producto mirando si contiene HTML y numeros, y lo guardamos en un set, para
no tener repetidos y recorrer luego
        for tag in soup.findAll('a', href=True):
            product = hasNumbers(tag['href'])
            if product is True and word in tag['href']:
                sinDuplicados.add(tag['href'])
                cont +=1
                #contador.add(cont)
        cont = 0
                #print sinDuplicados
        current_cont = len(sinDuplicados)
        productos = len(sinDuplicados)
        contP = 0
```

```
    except Exception:
        pass
    #bucle para recorrer los productos que hemos extraido previamente
    while sinDuplicados:
        try:
            #leemos la  primera URL de sinDuplicados y extraemos su codigo
HTML
            word = 'productImageZoomLayer'
            current_url = sinDuplicados.pop()
            webpage = requests.get(current_url, headers=headers)
            htmltext = webpage.content
            soup= BeautifulSoup(htmltext)
            print "Crawling", main_url
            #si el codigo HTML contiene la palabra word, quiere decir que es
un producto y por lo tanto seguimos
            if word in htmltext:
                timeP = datetime.now()
                print "Parsing", current_url
                #current_cont = contador.pop()
                if current_cont <= topMax:
                    top = 'TRUE'
                else:
                    top = 'FALSE'
                #sacamos el nombre de la categoría
                categoria = soup.findAll('li')[7].getText()
                categoria = categoria.replace("\r","")
                categoria = categoria.replace("\n","")
                categoria = " ".join(categoria.split())
                #sacamos el nombre de la subcategoria
                subcategoria = soup.findAll('li')[9].getText()
                subcategoria = subcategoria.replace("\r","")
                subcategoria = subcategoria.replace("\n","")
                subcategoria = " ".join(subcategoria.split())
                #sacamos el nombre del producto
                nombreP = soup.find('h1', itemprop='name').getText() #nombre
producto
                nombreP = nombreP.replace("\r","")
                nombreP = nombreP.replace("\n","")
                nombreP = nombreP.replace(",",".")
                nombreP = " ".join(nombreP.split())

                precioP = soup.find('span', itemprop='price').getText()
#precio venta producto
                precioP = precioP.replace("\r","")
                precioP = precioP.replace("\n","")
                precioP = precioP.replace(",",".")
                precioP = precioP.replace(" ", "")
                precioP = precioP[0:-2]

                try:
                    precioRealP = soup.find('div', { 'class' :
"old"}).getText()
                    precioRealP = precioRealP.replace("\r","")
                    precioRealP = precioRealP.replace("\n","")
                    precioRealP = precioRealP.replace(",",".")
                    precioRealP = precioRealP.replace(" ", "")
                    precioRealP = precioRealP[0:-2]
```

```
except Exception:
                    precioRealP = precioP

                linkP = current_url #link directo al producto
                #codigo del producto
                codigoP = soup.find('div', { "class" : "detailsTabContent
detailsTabContentDetail1 active"})['data-productid']

                word = 'productFlag blue'
                if word in htmltext: #comprobamos si el codigo html tiene el
producto marcado como novedad
                     novedades='TRUE'
                else:
                     novedades='FALSE'

             #C&A tiene una url diferente para cada color diferente que
tiene en cada articulo,
             #por lo tanto no se puede saber a simple vista el numero de
colores, y por eso pongo
             #el color en el que estamos, luego podriamos hacer un count de
las veces que aparece
             #su codigo para saber el numero de colores por ejemplo
                color = soup.find('div', { "class" : "colorLine"})
                color = color.find('h2').getText()
                color = color.replace("\r","")
                color = color.replace("\n","")
                color = color.replace(",",".")
                color = " ".join(color.split())
                color = color.replace("Color: ","")

                #buscamos en el codgio HTML por las tallas minimas y
maximas, en caso de ser complementos o derivados ponemos talla unica
                list_of_words = htmltext.split()
                next_word =
list_of_words[list_of_words.index('{"customerSizeNumber":') + 1]
                tallaMin = next_word.replace('"',"")
                tallaMin = tallaMin.replace(",","")
                if tallaMin == "":
                    tallaMin =
list_of_words[list_of_words.index('{"customerSizeNumber":') + 7]
                    tallaMin = tallaMin.replace('"',"")
                    tallaMin = tallaMin.replace(",","")

                next_word = list_of_words[list_of_words.index('}],') - 5]
                tallaMax = next_word.replace('"',"")
                tallaMax = tallaMax.replace(",","")
                if tallaMax == 'E' or tallaMax == 'A' or tallaMax == 'B' or
tallaMax == 'C' or tallaMax == 'D':
                    tallaMax = list_of_words[list_of_words.index('}],') - 6]
                    tallaMax = tallaMax.replace('"',"")
                    tallaMax = tallaMax.replace(",","")
                if tallaMin == "Talla":
                    tallaMin = 'Unica'
                    tallaMax = 'Unica'
```

```
                print "Categoria: %s - Subcategoria: %s - Nombre: %s -
            Precio: %s - Precio sin descontar: %s - Link: %s - Codigo: %s -
            Novedad: %s - Color: %s - Talla minima: %s - Talla
            maxima: %s"%(categoria,subcategoria,nombreP,precioP,precioRealP,
            linkP,codigoP,novedades,color, tallaMin, tallaMax)
                var_sql = """insert into crawlerVenca
(Client,Category,Brand,SubBrand,Description,NEW,PriceBeforeDiscount,ActualPr
ice,URL,Link,MinSize,MaxSize,CodeProd,ID_Presentation,Execution_Time,Top20,C
rawl_Day,Crawl_Date) values (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"""
                cursor.execute(var_sql,
(client,categoria,brand,subcategoria,nombreP,novedades,precioRealP,precioP,m
ain_url,linkP,tallaMin,tallaMax,codigoP,current_cont,fecha_ejecucion,top,tim
eIni,timeP))
                cursor.commit()
                print "current cont %s/%s"%(current_cont,productos)

        except Exception,e:
            f=open(filename,"a")
            #write error to file:
            f.write("Error occured:\n" + current_url + " <------> " + str(e)
+ "\n") # some message specific to this iteration (page) should be added
here...
            #close error log file:
            f.close()
            fallos.add(current_url)


        current_cont += -1
        contP += 1
        print "Productos %s/%s"%(contP,productos)
        if contP == productos:
            final += 1
            print "Escaparate crawleado con exito"

    contE += 1
    print "Crawleado escaparate %s/%s"%(contE,escaparates)
    print "Con exito: %s/%s"%(final,escaparates)


print "Crawler de %s ha finalizado correctamente"%(brand)
print "Lista de fallos %s"%(fallos)
cursor.close()
```

H&M

```
# -*- coding: utf-8 -*-
"""

@author: joelsanchez
"""
import sys
import time
import urllib
import pyodbc
import requests
from ordered_set import *
from selenium import webdriver
from bs4 import BeautifulSoup
from datetime import datetime
pmName = 'ordered_set'
pm = __import__(pmName)

headers = {'User-agent': 'Mozilla/5.0'}

cnxn = pyodbc.connect('DRIVER={SQL Server Native Client
10.0};SERVER=KOSMOS;DATABASE=MKT_CRAWLER;UID=[user];PWD=[password]')

cursor = cnxn.cursor()

fecha_error = time.strftime("%Y%m%d")
fallos = set()
fecha_ejecucion = time.strftime("%Y%m%d %H:%M:%S")
timeIni = datetime.now()
urls = {"http://www2.hm.com/es_es/mujer.html"}
url = "http://www2.hm.com"
word = 'mujer/'


connected = False
while connected == False:
    try:
        driver = webdriver.Chrome()#'C:/chromedriver.exe'
        time.sleep(20)
        current_url = urls.pop()
        driver.get(current_url)
        time.sleep(60)
        html = driver.execute_script("return
document.documentElement.innerHTML;")
        soup= BeautifulSoup(html)
        connected = True
    except Exception:
        pass


soup= BeautifulSoup(html)
```

```
#recorremos la URL y extraemos todos los links, filtramos por los de
producto mirando si contiene HTML y numeros, y lo guardamos en un set, para
no tener repetidos y recorrer luego
for tag in soup.findAll('a', href=True):
    if word in tag['href']:
        new_url = url + tag['href']
        urls.add(new_url)

current_cont = 0
final = 0
topMax = 20
client = 'Venca'
brand = 'HM'
filename= "S:/Consultas_python/CrawlerVenca/Errores/" + fecha_error + "_" +
brand + "_Errors.txt"

escaparates = len(urls)
contE = 0
while urls:


    current_url = urls.pop()
    main_url = current_url

    print "Crawling current_url : %s"%(current_url)
    sinDuplicados = OrderedSet()
    sinDuplicados.clear()

    try:

        driver.get(current_url);
        lastHeight = driver.execute_script("return
document.body.scrollHeight")
        while True:
            time.sleep(1)
            driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")
            time.sleep(2)
            newHeight = driver.execute_script("return
document.body.scrollHeight")

            html2 = driver.execute_script("return
document.documentElement.innerHTML;")
            soup= BeautifulSoup(html2)
            word = 'productpage'
            for tag in soup.findAll('a', href=True):
                if word in tag['href'] and '_jcr_content/product.quickbuy'
not in tag['href']:
                    new_url = url + tag['href']
                    #print new_url
                    sinDuplicados.add(new_url)

            time.sleep(1)
            if newHeight == lastHeight:
                break
            lastHeight = newHeight
        html2 = driver.execute_script("return
document.documentElement.innerHTML;")
```

```
    except Exception:

        driver.quit()
        sys.exit()

    soup= BeautifulSoup(html2)
    word = 'productpage'

    for tag in soup.findAll('a', href=True):
        if word in tag['href'] and '_jcr_content/product.quickbuy' not in
tag['href']:
            new_url = url + tag['href']
            #print new_url
            sinDuplicados.add(new_url)

    current_cont = len(sinDuplicados)
    productos = len(sinDuplicados)
    contP = 0
    while sinDuplicados:
        try:
            word = 'productpage'
            current_url = sinDuplicados.pop()
            webpage = requests.get(current_url, headers=headers)
            htmltext = webpage.content
            soup= BeautifulSoup(htmltext)
            print "Crawling", main_url
            print "Parsing", current_url

            timeP = datetime.now()
            #current_cont = contador.pop()
            if current_cont <= topMax:
                top = 'TRUE'
            else:
                top = 'FALSE'

          #Aparte del nombre del producto no podemos saber su categoría ni
subcategoria a simple vista
            nombreP = soup.find('h1', {"class" : "product-item-
headline"}).getText() #nombre producto
            nombreP = nombreP.replace("\r","")
            nombreP = nombreP.replace("\n","")
            nombreP = nombreP.replace(",",".")
            nombreP = " ".join(nombreP.split())

            categoria = nombreP.partition(' ')[0]

            try:
                precioP = soup.find('span', { 'class' : "price-
value" }).getText()
                precioP = precioP.replace("\r","")
                precioP = precioP.replace("\n","")
                precioP = precioP.replace("\t","")
                precioP = precioP.replace(" ","")
                precioP = precioP.replace(",",".")
                precioP = precioP.encode('utf8','ignore')
                precioP = precioP[:precioP.index("€")]
                print "Precio: %s"%(precioP)
```

```
                list_of_words = htmltext.split()
                precioRealP =
list_of_words[list_of_words.index("'priceValue':") + 1]

                if precioRealP == 'false,' or precioRealP == "'',":
                    precioRealP = precioP
                    precioRealP = precioRealP[:-1]
                else:
                    precioRealP = precioRealP.replace("'","")
                    precioRealP = precioRealP.replace(",",".")
                    precioRealP = precioRealP[:-1]

                print "Precio sin descontar: %s"%(precioRealP)
            except Exception:
                precioP = soup.find('span', { "class" : "price-
value" }).getText() #precio venta producto
                precioP = precioP.replace("\r","")
                precioP = precioP.replace("\n","")
                precioP = precioP.replace(",",".")
                precioP = precioP.replace(" ", "")
                precioP = " ".join(precioP.split())
                precioP = precioP[:-1]

                precioRealP = soup.find('small', { 'class' : "price-value-
original" })
                if precioRealP == None:
                    precioRealP = precioP
                else:
                    precioRealP = soup.find('small', { 'class' : "price-
value-original" }).getText()
                    precioRealP = precioRealP.replace("\r","")
                    precioRealP = precioRealP.replace("\n","")
                    precioRealP = precioRealP.replace("\t","")
                    precioRealP = precioRealP.replace(" ","")
                    precioRealP = precioRealP.replace(",",".")
                    precioRealP = precioRealP.encode('utf8','ignore')
                    precioRealP = precioRealP[:precioRealP.index("€")]
                    print "Precio sin descontar: %s"%(precioRealP)

            linkP = current_url

            list_of_words = htmltext.split()
            codigoP = list_of_words[list_of_words.index("'baseProductCode'")
- 1]
            codigoP = codigoP[14:]
            codigoP = codigoP.replace("'","")
            codigoP = codigoP.replace(",","")

            word = 'name="product-color"'
            ncolores = htmltext.count(word)

            try:
                tallaMin = soup.find('div', { 'class' : "detailbox
detailbox-value  unavailable-size"}).getText()

                tallaMax = soup.findAll('div', { 'class' : "detailbox
detailbox-value  unavailable-size"})
                last_div = None
```

```
            for last_div in tallaMax:pass #bucle para recorrer todas las
      tallas y coger la ultima
                if last_div:
                    tallaMax = last_div.getText()
            except AttributeError:
                tallaMin = 'Unica'
                tallaMax = tallaMin

            if tallaMin == '' or tallaMax=='':
                tallaMin = 0
                tallaMax = 0

            if 'novedades' in main_url: #comprobamos si el codigo html tiene
el producto marcado como novedad
                 novedades='TRUE'
            else:
                 novedades='FALSE'

            var_sql = """insert into crawlerVenca
(Client,Category,Brand,SubBrand,Description,PriceBeforeDiscount,ActualPrice,
URL,Link,MinSize,MaxSize,CodeProd,ID_Presentation,Execution_Time,Top20,Crawl
_Day,Crawl_Date,NEW) values (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"""
            cursor.execute(var_sql,
(client,categoria,brand,categoria,nombreP,precioRealP,precioP,main_url,linkP
,tallaMin,tallaMax,codigoP,current_cont,fecha_ejecucion,top,timeIni,timeP,no
vedades))
            cursor.commit()

            print "Categoria: %s - Subcategoria: %s - Nombre: %s -
Precio: %s - Precio Real: %s - Link: %s - Codigo: %s - Colores: %s - Talla
minima: %s - Talla
maxima: %s"%(categoria,categoria,nombreP,precioP,precioRealP,linkP,codigoP,n
colores,tallaMin,tallaMax)
            print "current cont %s/%s"%(current_cont,productos)

        except Exception,e:
            f=open(filename,"a")
            #write error to file:
            f.write("Error occured:\n" + current_url + " <------> " + str(e)
+ "\n") # some message specific to this iteration (page) should be added
here...
            #close error log file:
            f.close()
            fallos.add(current_url)

        current_cont += -1
        contP += 1
        print "Productos %s/%s"%(contP,productos)
        if contP == productos:
            final += 1
            print "Escaparate crawleado con exito"

    contE += 1
    print "Crawleado escaparate %s/%s"%(contE,escaparates)
    print "Con exito: %s/%s"%(final,escaparates)

print "Crawler de %s ha finalizado correctamente"%(brand)
print "Lista de fallos: %s"%(fallos)
driver.quit()
cursor.close()
```

Kiabi

```
# -*- coding: utf-8 -*-
"""

@author: joelsanchez
"""
import time
import urllib2
import urllib
import re
import pyodbc
import requests
from ordered_set import *
from bs4 import BeautifulSoup
from datetime import datetime
pmName = 'ordered_set'
pm = __import__(pmName)
#i = datetime.now()

headers = {'User-agent': 'Mozilla/5.0'}

cnxn = pyodbc.connect('DRIVER={SQL Server Native Client
10.0};SERVER=KOSMOS;DATABASE=MKT_CRAWLER;UID=[user];PWD=[password]')

cursor = cnxn.cursor()

def hasNumbers(inputString):
    return bool(re.search(r'\d', inputString))


urls = {"http://www.kiabi.es/"}
fecha_error = time.strftime("%Y%m%d")
url = "http://www.kiabi.es"
fecha_ejecucion = time.strftime("%Y%m%d %H:%M:%S")
timeIni = datetime.now()
fallos = set()
cont = 0
topMax = 20
client = 'Venca'
brand = 'Kiabi'
filename= "S:/Consultas_python/CrawlerVenca/Errores/" + fecha_error + "_"
+ brand + "_Errors.txt"

word = '-mujer_'
allP = '?pn=0'

final = 0

current_url = urls.pop()
#nos quedamos con su nombre ya que sera informacion para la base de datos
print "Crawling current_url : %s"%(current_url)
#sacamos su codigo HTML
webpage = requests.get(current_url, headers=headers)
htmltext = webpage.content
soup= BeautifulSoup(htmltext, "html.parser")
```

```
#recorremos la URL y extraemos todos los links, filtramos por los de
producto mirando si contiene HTML y numeros, y lo guardamos en un set, para
no tener repetidos y recorrer luego
for tag in soup.findAll('a', href=True):
    #if word in tag['href'] and "_P" not in tag['href'] and "http" not in
tag['href']:
    if 'mujer' in tag['href']:
        new_url = url + tag['href'] + allP
        urls.add(new_url)

escaparates = len(urls)
contE = 0
sinDuplicados = OrderedSet()
sinDuplicados.clear()
contador = OrderedSet()
contador.clear()
while urls:

    try:
        current_url = urls.pop()
        main_url = current_url
        print "Crawling current_url : %s"%(current_url)
        webpage = requests.get(current_url, headers=headers)
        htmltext = webpage.content
        soup= BeautifulSoup(htmltext, "html.parser")


        sinDuplicados = OrderedSet()
        sinDuplicados.clear()
        contador = OrderedSet()
        contador.clear()

        for tag in soup.findAll('a', href=True):
            product = bool(re.search('P[0-9]{6}', tag['href']))
            if product == True:
                new_url = url + tag['href']
                #print new_url
                sinDuplicados.add(new_url)
                cont +=1
                contador.add(cont)
        cont = 0
        current_cont = len(sinDuplicados)
        productos = len(sinDuplicados)
        contP = 0
    except Exception:
        pass
            #print sinDuplicados
    while sinDuplicados:

        try:
            current_url = sinDuplicados.pop()
            webpage = requests.get(current_url, headers=headers)
            htmltext = webpage.content

            soup= BeautifulSoup(htmltext)
            print "Crawling", main_url

            print "Parsing", current_url
            timeP = datetime.now()
            #current_cont = contador.pop()
```

```
            if current_cont <= topMax:
                top = 'TRUE'
            else:
                top = 'FALSE'

            try:
                #buscamos categoria
                categoria = soup.find('a', rel="v:url")
                if categoría == None:
                    categoria3 = 'Desconocida'
                    categoria4 = 'Desconocida'
                else:
                    categoria2 = categoria.findNext('a', rel="v:url")
                    categoria3 = categoria.findNext('a',
rel="v:url").getText() #categoria
                    categoria4 = categoria2.findNext('a',
rel="v:url").getText() #subcategoria
            except Exception:
                categoria4 = "None"
                categoria3 = "None"
            print "Categoria: %s - Subcategoria: %s"%(categoria3,
categoria4)
                #buscamos nombre prodcuto
            nombreP = soup.find('h1', { "class" : "breadcrumb" })
            if nombreP == None:
                nombreP = 'Desconocido'
            else:
                nombreP = soup.find('h1', { "class" :
"breadcrumb" }) .getText()
                nombreP = nombreP.encode('utf-8')
                nombreP= nombreP.replace('˜','')
                nombreP = nombreP.replace('”','')#nombre producto
            print "Nombre: %s"%(nombreP)
            try:
                #buscamos precio de venta producto
                precioP = soup.find_all('meta',
itemprop='price')[0]['content'] #precio venta producto
                precioP = precioP.replace(',','.')
                precioP = float(precioP)
                if precioP == None:
                    precioP = '0'
            except IndexError:
                precioP = soup.find('title').getText()
                precioP = precioP.split('-')
                precioP = precioP.pop(len(precioP)-1)
                precioP = precioP[:-2]
                precioP = precioP.replace(",",".")

            try:

                #buscamos el precio 'tachado' del producto
                precioRealP = soup.find('span', { "class" : "listPrice
price" }).getText() #precio sin descontar (en caso de oferta, aparecera
tachado en la web)
                precioRealP = precioRealP.replace("\r","")
                precioRealP = precioRealP.replace("\n","")
                precioRealP = precioRealP.replace(" ","")
                if precioRealP == "":
                    precioRealP = precioP
```

```
                else:
                    precioRealP = soup.find('span', { "class" : "listPrice
price" }).getText()
                    precioRealP = precioRealP.replace("\r","")
                    precioRealP = precioRealP.replace("\n","")
                    precioRealP = precioRealP.replace(",",".")
                    precioRealP = precioRealP.encode('ascii','ignore')
                    precioRealP = precioRealP[0:4]
                    precioRealP = float(precioRealP) #precio sin descontar
(en caso de oferta, aparecera tachado en la web)

            except AttributeError:
                list_html = htmltext.split('"')
                precioRealP = list_html[list_html.index('listPrix') + 2]
            if precioRealP == "":
                precioRealP = precioP
            print "Precio: %s - Precio sin
descontar: %s"%(precioP,precioRealP)

            linkP = current_url #link al producto
            print "Link: %s"%(linkP)

            try:
                codigoP = soup.find('meta', { 'property' :
"og:url"})['content'] #codigo producto
                codigoP = codigoP[codigoP.index('_P')+2:]
                if 'C' in codigoP:
                    codigoP = codigoP[:codigoP.index('C')]
                #codigoP = int(re.search(r'\d+', codigoP).group())
                #codigoP = codigoP.replace("[","")
                #codigoP = codigoP.replace("]","")
                #codigoP = map(int, re.findall(r'\d+', codigoP))
            except Exception:
                list_url = current_url.split('/')
                findCodigo = list_url[len(list_url)-1]
                try:
                    codigoP =
findCodigo[findCodigo.index('_P')+2:findCodigo.index('#')]
                except Exception:
                    codigoP =
findCodigo[findCodigo.index('_P')+2:findCodigo.index('C')]

            print "Codigo: %s"%(codigoP)


            word='["detail"] = "novedades"'
            if word in htmltext or 'novelties' in htmltext: #comprobamos si
el codigo html tiene el producto marcado como novedad
                novedades='TRUE'
            else:
                novedades='FALSE'
            word = 'colorPrixTTC'

            ncolores = htmltext.count(word) #numero de colores del articulo
            tallaMin = soup.find('div', { "class" : "sizes_section" })
            if tallaMin == None:
                tallaMin2 = 'Desconocida'
```

```
            else:
                tallaMin2 = tallaMin.find('span', {"class" :
"available"}).getText() #talla minima
                tallaMax = tallaMin.findAll('span', {"class" : "available"})
                last_div = None
                for last_div in tallaMax:pass #bucle para recorrer todas las
tallas y coger la ultima
                if last_div:
                    tallaMax = last_div.getText()


            print "current cont %s/%s"%(current_cont,productos)

            var_sql = """insert into crawlerVenca
(Client,Category,Brand,SubBrand,Description,NEW,PriceBeforeDiscount,ActualPr
ice,URL,Link,#Colors,MinSize,MaxSize,CodeProd,ID_Presentation,Execution_Time
,Top20,Crawl_Day, Crawl_Date) values
(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"""
            cursor.execute(var_sql,
(client,categoria3,brand,categoria4,nombreP,novedades,precioRealP,precioP,ma
in_url,linkP,ncolores,tallaMin2,tallaMax,codigoP,current_cont,fecha_ejecucio
n,top,timeIni,timeP))
            cursor.commit()
        except Exception,e:
            f=open(filename,"a")
            #write error to file:
            f.write("Error occured:\n" + current_url + " <------> " + str(e)
+ "\n") # some message specific to this iteration (page) should be added
here...
            #close error log file:
            f.close()
            fallos.add(current_url)


        current_cont += -1
        contP += 1
        print "Productos %s/%s"%(contP,productos)
        if contP == productos:
            final += 1
            print "Escaparate crawleado con exito"

    contE += 1
    print "Crawleado escaparate %s/%s"%(contE,escaparates)
    print "Con exito: %s/%s"%(final,escaparates)


print "Crawler de %s ha finalizado correctamente"%(brand)
print "Lista de fallos: %s"%(fallos)
cursor.close()
```

Lefties

```
# -*- coding: utf-8 -*-
"""

@author: joelsanchez
"""

import time
import re
import pyodbc
import requests
from ordered_set import *
from bs4 import BeautifulSoup
from datetime import datetime
from selenium import webdriver
pmName = 'ordered_set'
pm = __import__(pmName)
#i = datetime.now()
#Funcion para saber si un string contiene numeros, en algunas webs nos
sirve para comprobar si un link es de producto o no
def hasNumbers(inputString):
    return bool(re.search(r'\d', inputString))

#headers para simular que entramos desde Mozilla y no nos de 'Acces
denied'
headers = {'User-agent': 'Mozilla/5.0'}

#conectamos a SQL
cnxn = pyodbc.connect('DRIVER={SQL Server Native Client
10.0};SERVER=KOSMOS;DATABASE=MKT_CRAWLER;UID=[user];PWD=[password]')

cursor = cnxn.cursor()

fecha_error = time.strftime("%Y%m%d")
fecha_ejecucion = time.strftime("%Y%m%d %H:%M:%S")
timeIni = datetime.now()
cont = 0
final = 0
topMax = 20
errores = 0
fallos = set()
client = 'Venca'
brand = 'Lefties'
filename= "S:/Consultas_python/CrawlerVenca/Errores/" + fecha_error + "_"
+ brand + "_Errors.txt"

word = '/woman/'

connected = False
while connected == False:

    try:
        driver = webdriver.Chrome()#'C:/chromedriver.exe'
        time.sleep(10)
        urls = {'http://www.lefties.com/es/'}
        current_url = urls.pop()
        driver.get(current_url)
        time.sleep(10)
```

```
      driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")
        time.sleep(2)

      html2 = driver.execute_script("return
document.documentElement.innerHTML;")
        soup= BeautifulSoup(html2)
        connected = True

    except Exception:
        pass

#recorremos la URL y extraemos todos los links, filtramos por los de
producto mirando si contiene HTML y numeros, y lo guardamos en un set, para
no tener repetidos y recorrer luego
for tag in soup.findAll('a', href=True):
    if (word in tag['href'] or '/zapatos-mujer/' in tag['href']) or
'sportswear' in tag['href']:
        product = bool(re.search('p[0-9]{3}', tag['href']))
        if '?NC=1' not in tag['href'] and product == False:
            if "http" in tag['href']:
                urls.add(tag['href'])
            else:
                new_url = "https://www.lefties.com" + tag['href']
                urls.add(new_url)
river.quit()

escaparates = len(urls)
escaparates = len(urls)
contE = 0
#bucle de escaparates, mientras queden links sigue haciendo
while urls:

    #sacamos el primer escaparate
    current_url = urls.pop()
    #nos quedamos con su link
    main_url = current_url
    print "Crawling current_url : %s"%(current_url)
    #entramos desde Mozilla y leemos la web y cogemos su codigo html
    webpage = requests.get(current_url, headers=headers)
    htmltext = webpage.content
    soup= BeautifulSoup(webpage.content)
    word = 'html'
    sinDuplicados = OrderedSet()
    sinDuplicados.clear()
    contador = OrderedSet()
    contador.clear()
    for tag in soup.findAll('a', href=True):
        product = bool(re.search('p[0-9]{3}', tag['href']))
        if product == True:
            if 'http' not in tag['href'] and 'tarjeta-regalo' not in
tag['href']:
                new_url = 'https:'+tag['href']
                sinDuplicados.add(new_url)
            else:
                sinDuplicados.add(tag['href'])
            cont +=1
            #contador.add(cont)
    cont = 0
```

```
    current_cont = len(sinDuplicados)
    productos = len(sinDuplicados)
    contP = 0

    while sinDuplicados:

        try:
            #cogemos el primer producto del set
            current_url = sinDuplicados.pop()
            linkP = current_url
            print "Crawling", main_url

            timeP = datetime.now()
            #leemos la web y su codigo html, simulando que somos Mozilla
            webpage = requests.get(current_url, headers=headers)
            htmltext = webpage.content
            soup= BeautifulSoup(webpage.content)
            print "Parsing", current_url

            #current_cont = contador.pop()
            if current_cont <= topMax:
                top = 'TRUE'
            else:
                top = 'FALSE'

            try:
                #sacamos categoria
                categoria = soup.find('title').getText()
                categorías = categoria.split('-')
                categoría = categorías [2]
                subcategoria = categorías[1]
                categoría = categoria.encode('utf-8')
                categoría = categoria.replace('\t','')
                categoría = categoria.replace('  ','')
                subcategoria = subcategoria.encode('utf-8')
                subcategoria = subcategoria.replace('\t','')
                subcategoria = subcategoria.replace('  ','')
            except Exception:
                categoria = "None"
                subcategoria = "None"
            print "Categoria: %s"%(categoria)
            print "Subcategoria: %s"%(subcategoria)


            try:
                #nombre producto
                nombreP = categorías [0]
                nombreP = nombreP.encode('utf-8')
                nombreP = nombreP.replace('\t','')
                nombreP = nombreP.replace('  ','')
            except Exception:
                nombreP = soup.find_all('meta', { 'property' :
"og:title"})[0]['content']
                nombres = nombreP.split('-')
                nombreP = nombres[0]
                nombreP = nombreP.encode('utf-8')
                nombreP = nombreP.replace('\t','')
                nombreP = nombreP.replace('  ','')
            print "Nombre: %s"%(nombreP)
```

```
            #precio producto
            list_htmls = htmltext.split('"')

            precioP = list_htmls[list_htmls.index('pricecurrency')-2]
            try:
                precios=precioP.split('-')
                precioP = precios[1]
            except Exception:
                precioP = list_htmls[list_htmls.index('pricecurrency')-2]
            precioRealP = precioP

            print "Precio sin descontar: %s"%(precioRealP)


            print "Precio: %s"%(precioP)
                #link directo al producto

            try:
                #codigo producto
                codigoP = list_htmls[list_htmls.index('productId')+2]
            except Exception:
                list_url = current_url.split('-')
                codigoP = list_url[len(list_url)-1]
                codigoP = codigoP[codigoP.index('p')+1:codigoP.index('.')]
            print "Codigo: %s"%(codigoP)

            if 'new' in main_url:
                novedades = 'TRUE'
            else:
                novedades = 'FALSE'

            print "Novedades: %s"%(novedades)

            #nmero de colores

            ncolores = 0
            print "Colores: %s"%(ncolores)

            tallaMin = 0
            tallaMax= 0
            print "Talla Min: %s"%(tallaMin)
            print "Talla Max: %s"%(tallaMax)

            var_sql = """insert into crawlerVenca
(Client,Category,Brand,SubBrand,Description,NEW,PriceBeforeDiscount,ActualPr
ice,URL,Link,MinSize,MaxSize,CodeProd,ID_Presentation,Execution_Time,#Colors
,Top20,Crawl_Day, Crawl_Date) values
(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"""
            cursor.execute(var_sql,
(client,categoria,brand,subcategoria,nombreP,novedades,precioRealP,precioP,m
ain_url,linkP,tallaMin,tallaMax,codigoP,current_cont,fecha_ejecucion,ncolore
s,top,timeIni,timeP))
            cursor.commit()
            print "current cont %s/%s"%(current_cont,productos)
```

```
        except Exception,e:
            f=open(filename,"a")
            #write error to file:
            f.write("Error occured:\n" + current_url + " <------> " + str(e)
+ "\n") # some message specific to this iteration (page) should be added
here...
            #close error log file:
            f.close()
            fallos.add(current_url)
            errores += 1

        current_cont += -1
        contP += 1
        print "Productos %s/%s"%(contP,productos)
        if contP == productos:
            final += 1
            print "Escaparate crawleado con exito"

    contE += 1
    print "Crawleado escaparate %s/%s"%(contE,escaparates)
    print "Con exito: %s/%s"%(final,escaparates)



print "Crawler de %s ha finalizado correctamente"%(brand)
print "Lista de fallos %s"%(fallos)
cursor.close()
```

Mango

```
# -*- coding: utf-8 -*-
"""

@author: joelsanchez
"""
import sys
import time
import urllib
import pyodbc
import requests
import ctypes
from ordered_set import *
from selenium import webdriver
from bs4 import BeautifulSoup
from datetime import datetime
pmName = 'ordered_set'
pm = __import__(pmName)
#i = datetime.now()
headers = {'User-agent': 'Mozilla/5.0'}

cnxn = pyodbc.connect('DRIVER={SQL Server Native Client
10.0};SERVER=KOSMOS;DATABASE=MKT_CRAWLER;UID=[user];PWD=[password]')

cursor = cnxn.cursor()

"""
sql = "SELECT URL from mkt.dbo.url_input where Brand = 'Mango'"
urls = set()
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Fetch all the rows in a list of lists.
   results = cursor.fetchall()
   for row in results:
      new_urls = {row[0]}
      urls.update(new_urls)
      # Now print fetched result
      print "URL to crawl=%s" %(urls)

except:
   print "Error: unable to fecth data"
"""
headers = {'User-agent': 'Mozilla/5.0'}

SetCursorPos = ctypes.windll.user32.SetCursorPos
mouse_event = ctypes.windll.user32.mouse_event

def left_click(x, y, clicks=1):
  SetCursorPos(x, y)
  for i in xrange(clicks):
   mouse_event(2, 0, 0, 0, 0)
   mouse_event(4, 0, 0, 0, 0)
fecha_error = time.strftime("%Y%m%d")
fecha_ejecucion = time.strftime("%Y%m%d %H:%M:%S")
timeIni = datetime.now()
cont = 0
finale = 0
```

```
topMax = 20
errores = 0
fallos = set()
client = 'Venca'
brand = 'Mango'
filename= "S:/Consultas_python/CrawlerVenca/Errores/" + fecha_error + "_" +
brand + "_Errors.txt"

#urls = {"http://shop.mango.com/ES/mujer"}

url = "http://shop.mango.com/"
# This will get the initial html - before javascript
word = 'mujer/'

connected = False
while connected == False:

    try:
        driver = webdriver.Chrome()#'C:/chromedriver.exe'
        time.sleep(10)
        urls = {'http://shop.mango.com/ES/mujer'}
        current_url = urls.pop()
        driver.get(current_url)
        time.sleep(10)
        driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")
        time.sleep(2)
        html2 = driver.execute_script("return
document.documentElement.innerHTML;")
        soup= BeautifulSoup(html2)
        connected = True

    except Exception:
        pass

for tag in soup.findAll('a', href=True):
#print tag['href']
    if word in tag['href'] and "html" not in tag['href'] and "http" in
tag['href'] and "/edits" not in tag['href'] and '-campaign' not in
tag['href']:

        new_url = tag['href'] + '?prov=8'
        if new_url != 'http://shop.mango.com/ES/mujer/prendas?prov=8' and
new_url != 'http://shop.mango.com/ES/mujer/accesorios?prov=8':
            urls.add(new_url)
print urls

escaparates = len(urls)
contE = 0
while urls:

    current_url = urls.pop()
    main_url = current_url
    print "Crawling current_url : %s"%(current_url)
    sinDuplicados = OrderedSet()
    sinDuplicados.clear()
    try:
        driver.get(current_url)
        time.sleep(5)
```

```
        try:
            element = driver.find_element_by_xpath("//div[@class='icon
closeModal icon__close']")
            element.click()
        except Exception:
            pass
        lastHeight = driver.execute_script("return
document.body.scrollHeight")
        while True:
            driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")
            time.sleep(1)
            newHeight = driver.execute_script("return
document.body.scrollHeight")
            time.sleep(1)
            html2 = driver.execute_script("return
document.documentElement.innerHTML;")
            soup= BeautifulSoup(html2)
            #links = set()
            word = 'ES/p0'

            #contador = OrderedSet()
            #contador.clear()

            for tag in soup.findAll('a', href=True):
                if word in tag['href']:
                    if url in tag['href']:
                        sinDuplicados.add(tag['href'])
                    else:
                        new_url = url + tag['href']
                        sinDuplicados.add(new_url)
            time.sleep(1)
            if newHeight == lastHeight:
                break
            lastHeight = newHeight
            html2 = driver.execute_script("return
document.documentElement.innerHTML;")
    except Exception:
        pass

    soup= BeautifulSoup(html2)
    #links = set()
    word = 'ES/p0'

    #contador = OrderedSet()
    #contador.clear()

    for tag in soup.findAll('a', href=True):
        if word in tag['href']:
            if url in tag['href']:
                sinDuplicados.add(tag['href'])
            else:
                new_url = url + tag['href']
                sinDuplicados.add(new_url)
            cont +=1
            #contador.add(cont)
    cont = 0
            #print sinDuplicados
```

```
    current_cont = len(sinDuplicados)
    productos = len(sinDuplicados)
    contP = 0
    while sinDuplicados:

        try:
            current_url = sinDuplicados.pop()
            print "Crawling", main_url
            print "Parsing", current_url

            timeP = datetime.now()
            #current_cont = contador.pop()
            if current_cont <= topMax:
                top = 'TRUE'
            else:
                top = 'FALSE'

            webpage = requests.get(current_url, headers=headers)
            htmltext = webpage.content
            soup= BeautifulSoup(htmltext)
            print "Parsing", current_url
            cont += 1

            list_url = current_url.split('/')
            try:
                categoría = list_url[6]
                subcategoria = list_url[7]

            except Exception:
                categorías = list_url[len(list_url)-1].split('-')
                categoría = categorías[0]
                subcategoria = categoría
            print "Categoria: %s"%(categoría)
            print "Subcategoria: %s"%(subcategoria)

            try:
                nombreP = soup.find('div', { 'itemprop' : "name"}).getText()
                nombreP = nombreP.replace("\r","")
                nombreP = nombreP.replace("\n","")
                nombreP = nombreP.replace('\t','')
                nombreP = nombreP.replace('  ','')
            except Exception:
                nombreP = list_url[len(list_url)-2]
                nombreP = nombreP.replace('-',' ')
            print "Nombre: %s"%(nombreP)

            codigoP =
current_url[current_url.index('id=')+3:current_url.index('_')]
            print "Codigo: %s"%(codigoP)

            linkP = current_url

            try:
                list_html=htmltext.split('"')
                precioP = list_html[list_html.index('price') + 1 ]
                precioP = precioP.replace(',','')
                precioP = precioP.replace(':','')
            except Exception:
```

```
                precioP = soup.find('span', { 'itemprop' :
"price"}).getText()
                precioP = precioP.encode('utf-8')
                list_precioP=precioP.split(',')
                if len(list_precioP) > 2:
                    #precioP = precioP.encode('utf-8')
                    precioP = precioP[precioP.index('€')+3:-4]
                    precioP = precioP.replace(",",".")
                else:
                    precioP = soup.find('span', { 'itemprop' :
"price"}).getText()
                    precioP = precioP.replace(",",".")
                    precioP = precioP.encode('utf-8')
                    precioP = precioP[:precioP.index('€')]
            print "Precio: %s"%(precioP)

            precioRealP = soup.find('span', { 'itemprop' :
"price"}).getText()
            precioRealP = precioRealP.replace(",",".")
            precioRealP = precioRealP.encode('utf-8')
            precioRealP = precioRealP[:precioRealP.index('€')]
            print "Precio Real: %s"%(precioRealP)

            if precioRealP == '0':
                precioRealP = precioP


            word = 'class="exclusive"'
            if word in htmltext:
                novedades = 'TRUE'
            else:
                novedades = 'FALSE'

            print "Novedad: %s"%(novedades)


            colores = 'name="colors_'
            ncolores = htmltext.count(colores)

            print "Colores: %s"%(ncolores)

            tallas = soup.find_all('input', {'class' :
"inputOcultoColor"})[0]['value']
            list_of_tallas = tallas.split()
            final = len(list_of_tallas)
            tallaMin = list_of_tallas[0]
            tallaMax = list_of_tallas[final-1]
            tallaMax = tallaMax.encode('utf-8')
            word = tallaMax[4:]

            if word == 'ca':
                tallaMin = 'Unica'
                tallaMax = 'Unica'
            else:
                word2 = '#'
                if word2 in tallaMin:
```

```
                    tallaMin =
            tallaMin[tallaMin.index('|'):tallaMin.index('#')]
                    tallaMin = tallaMin.split('|')
                    finalMin = len(tallaMin)
                    tallaMin = tallaMin[finalMin - 1]
                    tallaMax = tallaMax.split('|')
                    finalMax = len(tallaMax)
                    tallaMax = tallaMax[finalMax - 1]
                else:
                    tallaMin = tallaMin.split('|')
                    finalMin = len(tallaMin)
                    tallaMin = tallaMin[finalMin - 1]
                    tallaMax = tallaMax.split('|')
                    finalMax = len(tallaMax)
                    tallaMax = tallaMax[finalMax - 1]


            print "Tallas Min: %s Talla Max: %s"%(tallaMin,tallaMax)
            var_sql = """insert into crawlerVenca
(Client,Category,Brand,SubBrand,Description,NEW,PriceBeforeDiscount,ActualPr
ice,URL,Link,MinSize,MaxSize,CodeProd,ID_Presentation,Execution_Time,#Colors
,Top20,Crawl_Day, Crawl_Date) values
(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"""
            cursor.execute(var_sql,
(client,categoria,brand,subcategoria,nombreP,novedades,precioRealP,precioP,m
ain_url,linkP,tallaMin,tallaMax,codigoP,current_cont,fecha_ejecucion,ncolore
s,top,timeIni,timeP))
            cursor.commit()
            print "current cont %s/%s"%(current_cont,productos)
        except Exception,e:
            f=open(filename,"a")
            #write error to file:
            f.write("Error occured:\n" + current_url + " <------> " + str(e)
+ "\n") # some message specific to this iteration (page) should be added
here...
            #close error log file:
            f.close()
            fallos.add(current_url)



        current_cont += -1
        contP += 1
        print "Productos %s/%s"%(contP,productos)
        if contP == productos:
            finale += 1
            print "Escaparate crawleado con exito"

    contE += 1
    print "Crawleado escaparate %s/%s"%(contE,escaparates)
    print "Con exito: %s/%s"%(finale,escaparates)


print "Crawler de %s ha finalizado correctamente"%(brand)
print "Numero de errores: %s"%(errores)
print "Lista de fallos: %s"%(fallos)
driver.quit()
cursor.close()
```

Shana

```
# -*- coding: utf-8 -*-
"""

@author: joelsanchez
"""
import sys
import urllib
import urllib2
import time
import re
import pyodbc
import requests
from selenium import webdriver
from ordered_set import *
from bs4 import BeautifulSoup
from datetime import datetime
pmName = 'ordered_set'
pm = __import__(pmName)
timeIni = datetime.now()

headers = {'User-agent': 'Mozilla/5.0'}

def hasNumbers(inputString):
    return bool(re.search(r'\d', inputString))

#conectamos a SQL
cnxn = pyodbc.connect('DRIVER={SQL Server Native Client
10.0};SERVER=KOSMOS;DATABASE=MKT_CRAWLER;UID=[user];PWD=[password]')

cursor = cnxn.cursor()
"""
#seleccionamos la lista de links de escaparates a recorrer de una lista de
SQL
sql = "SELECT URL from mkt.dbo.url_input where Brand = 'Shana'"
urls = set()
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Fetch all the rows in a list of lists.
   results = cursor.fetchall()
   #para cada fila de SQL cogemos la informacion (links) y creamos un set
de links
   for row in results:
      new_urls = {row[0]}
      urls.update(new_urls)
      # Now print fetched result
      print "URL to crawl=%s" %(urls)

except:
   print "Error: unable to fecth data"
"""
fecha_error = time.strftime("%Y%m%d")
fecha_ejecucion = time.strftime("%Y%m%d %H:%M:%S")
fallos = set()
visited = set()
```

Analizando a la competencia

```
#sinDuplicados = set()
cont = 0
topMax = 20
final = 0
client = 'Venca'
brand = 'Shana'
filename= "S:/Consultas_python/CrawlerVenca/Errores/" + fecha_error + "_" +
brand + "_Errors.txt"

urls = {'http://www.shana.com/es/31000-ropa-
online','http://www.shana.com/es/33000-new-
in','http://www.shana.com/es/31010-camisetas'}
url = "http://www.shana.com/es/"

escaparates = len(urls)
contE = 0

while urls:
    #sacamos el primer escaparate
    current_url = urls.pop()
    #nos guardamos el link del escaparate para añadirlo a SQL al final
    main_url = current_url
    print "Crawling current_url : %s"%(current_url)
    webpage = requests.get(current_url, headers=headers)
    htmltext = webpage.content
    soup= BeautifulSoup(htmltext)
    sinDuplicados = OrderedSet()
    sinDuplicados.clear()
    contador = OrderedSet()
    contador.clear()

    for tag in soup.findAll('a', href=True):
        product = bool(re.search('/[0-9]{4}-', tag['href']))
        if product == True:
            sinDuplicados.add(tag['href'])
            cont +=1
            contador.add(cont)
    cont = 0
    current_cont = len(sinDuplicados)
    productos = len(sinDuplicados)
    contP = 0

    while sinDuplicados:
        try:
                current_url = sinDuplicados.pop()
                webpage = requests.get(current_url, headers=headers)
                htmltext = webpage.content
                soup= BeautifulSoup(htmltext)

                print "Crawling", main_url
                print "Parsing", current_url
                timeP = datetime.now()
                #current_cont = contador.pop()
                if current_cont <= topMax:
                    top = 'TRUE'
                else:
                    top = 'FALSE'

                new_main = main_url.split('/')
                new_sub = current_url.split('/')
```

```
                #buscamos categoría, subcategoria y nombre producto
                categoría = new_main.pop(4)
                subcategoria = new_sub.pop(4)
                nombreP = soup.find('title').getText()
                nombreP = nombreP.split('-')
                nombreP = nombreP.pop(0)

                print "Categoria: %s"%(categoría)
                print "Subcategoria: %s"%(subcategoria)
                print "Nombre: %s"%(nombreP)

                precioP = soup.find('span', { 'itemprop' :
"price" }).getText()
                precioP = precioP[2:]
                precioP = precioP.replace(',','.')

                """
                precioP = soup.find('p', { 'class' : "price" }).getText()
                precioP = precioP.replace("\r","")
                precioP = precioP.replace("\n","")
                precioP = precioP.replace(" ","")
                precioP = precioP.replace(",",".")
                precioP = precioP.encode('utf8','ignore')
                precioP = precioP[:precioP.index("€")]"""

                print "Precio: %s"%(precioP)


                precioRealP = soup.find('span', { 'id' :
"old_price_display" })
                try:
                    precioRealP = soup.find('span', { 'id' :
"old_price_display" }).getText()
                except Exception:
                    precioRealP = precioP

                if precioRealP == '':
                    precioRealP = precioP
                else:
                    precioRealP = precioRealP[2:]
                    precioRealP = precioRealP.replace(',','.')
                print "Precio sin descontar: %s"%(precioRealP)

                linkP = current_url

                codigoP = soup.find('input', { 'name' :
"id_product"})['value'] #codigo producto
                print "Codigo: %s"%(codigoP)


                if 'novedades' in main_url:
                    novedades='TRUE'
                else:
                    novedades='FALSE'
                print "Novedad: %s"%(novedades)

                ncolores = htmltext.count('class="color-pordefecto"')
                ncolores = ncolores - 1
                print "Colores: %s"%(ncolores)
```

```
                html_text = htmltext.split('"')
                tallaMin=html_text[html_text.index('attribute_radio') + 7]
                try:
                    tallaMin =
tallaMin[tallaMin.index('<span>')+6:tallaMin.index('</span>')]
                except Exception:
                    tallaMin = "Agotada"

                tallaMax=html_text[html_text.index('box-cart-bottom') - 1]
                try:
                    tallaMax =
tallaMax[tallaMax.index('<span>')+6:tallaMax.index('</span>')]
                except Exception:
                    tallaMax = "Agotada"
                print "Talla Min: %s"%(tallaMin)
                print "Talla Max: %s"%(tallaMax)

                var_sql = """insert into crawlerVenca
(Client,Category,Brand,SubBrand,Description,NEW,PriceBeforeDiscount,ActualPr
ice,URL,Link,#Colors,MinSize,MaxSize,CodeProd,ID_Presentation,Execution_Time
,Top20,Crawl_Day, Crawl_Date) values
(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"""
                cursor.execute(var_sql,
(client,categoria,brand,subcategoria,nombreP,novedades,precioRealP,precioP,m
ain_url,linkP,ncolores,tallaMin,tallaMax,codigoP,current_cont,fecha_ejecucio
n,top,timeIni,timeP))
                cursor.commit()
                print "current cont %s/%s"%(current_cont,productos)

        except Exception,e:
            f=open(filename,"a")
            #write error to file:
            f.write("Error occured:\n" + current_url + " <------> " + str(e)
+ "\n") # some message specific to this iteration (page) should be added
here...
            #close error log file:
            f.close()
            fallos.add(current_url)


        current_cont += -1
        contP += 1
        print "Productos %s/%s"%(contP,productos)
        if contP == productos:
            final += 1
            print "Escaparate crawleado con exito"

    contE += 1
    print "Crawleado escaparate %s/%s"%(contE,escaparates)
    print "Con exito: %s/%s"%(final,escaparates)


print "Crawler de %s ha finalizado correctamente"%(brand)
print "Lista de fallos: %s"%(fallos)

cursor.close()
```

Stradivarius

```
# -*- coding: utf-8 -*-
"""

@author: joelsanchez
"""
import os
import time
import re
import pyodbc
import requests
from ordered_set import *
from selenium import webdriver
from bs4 import BeautifulSoup
from datetime import datetime
pmName = 'ordered_set'
pm = __import__(pmName)
i = datetime.now()

cnxn = pyodbc.connect('DRIVER={SQL Server Native Client
10.0};SERVER=KOSMOS;DATABASE=MKT_CRAWLER;UID=[user];PWD=[password]')

cursor = cnxn.cursor()

#Funcion para saber si un string contiene numeros, en algunas webs nos
sirve para comprobar si un link es de producto o no
def hasNumbers(inputString):
    return bool(re.search(r'\d', inputString))

headers = {'User-agent': 'Mozilla/5.0'}
fecha_ejecucion = time.strftime("%Y%m%d %H:%M:%S")
timeIni = datetime.now()
fecha_error = time.strftime("%Y%m%d")
fallos = set()
cont = 0
topMax = 20
final = 0
client = 'Venca'
brand = 'Stradivarius'
filename= "S:/Consultas_python/CrawlerVenca/Errores/" + fecha_error + "_"
+ brand + "_Errors.txt"

urls = {'http://www.stradivarius.com/es/mujer/ropa/punto-c1317531.html',
'http://www.stradivarius.com/es/mujer/ropa/camisas-c1317522.html' }

escaparates = len(urls)
contE = 0
#bucle de las urls previamente sacadas del SQL, hasta que no se recorran
todas, seguimos
while urls:

    #sacamos la primera URL a recorrer
    current_url = urls.pop()
    #nos quedamos con su nombre ya que sera informacion para la base de
datos
    main_url = current_url
```

```
    print "Crawling current_url : %s"%(current_url)
    #sacamos su codigo HTML, simulando que somos Mozilla
    webpage = requests.get(current_url, headers=headers)
    htmltext = webpage.content
    soup= BeautifulSoup(webpage.content)
    word = 'html'
    sinDuplicados = OrderedSet()
    sinDuplicados.clear()
    contador = OrderedSet()
    contador.clear()
    #recorremos la URL y extraemos todos los links, filtramos por los de
producto mirando si contiene HTML y numeros, y lo guardamos en un set, para
no tener repetidos y recorrer luego
    for tag in soup.findAll('a', href=True):
        product = hasNumbers(tag['href'])
        if product is True and word in tag['href'] and '*' not in
tag['href']:
            sinDuplicados.add(tag['href'])
            cont +=1
            contador.add(cont)
    cont = 0
    #bucle para recorrer los productos que hemos extraido previamente
    current_cont = len(sinDuplicados)
    productos = len(sinDuplicados)
    contP = 0
    while sinDuplicados:
        try:

            #leemos la  primera URL de sinDuplicados y extraemos su codigo
HTML
            word = '-c'
            current_url = sinDuplicados.pop()
            print "Crawling", main_url
            timeP = datetime.now()
            #si el codigo HTML contiene la palabra word, quiere decir que es
un producto y por lo tanto seguimos
            if word in current_url:
                webpage = requests.get(current_url, headers=headers)
                htmltext = webpage.content
                soup= BeautifulSoup(webpage.content)
                print "Parsing", current_url

                #current_cont = contador.pop()
                if current_cont <= topMax:
                    top = 'TRUE'
                else:
                    top = 'FALSE'


                #sacamos el nombre de la categoría y subcategoria
                list_of_words = current_url.split('/')
                categoría = list_of_words[list_of_words.index('es') + 1]
                subcategoria = list_of_words[list_of_words.index('es') + 2]
                print "Categoria: %s -
Subcategoria: %s"%(categoria,subcategoria)

                try:
```

```
                    #sacamos el nombre del producto
                    nombreP = soup.find_all('meta', { 'name' :
"title" })[0]['content']

                    print "Nombre: %s"%(nombreP)
                except IndexError:
                    nombreP = soup.find('title').getText()
                    list_n = nombreP.split('-')
                    nombreP = list_n[0]
                    print "Nombre: %s"%(nombreP)

                #precio producto
                """list_p = htmltext.split(':')
                precioP = list_p[list_p.index('"Offer",\r\n\t\t"price"') +
1]
                precioP = precioP[:precioP.index(',')]
                precioP = precioP.replace('"',"")
                precioRealP = precioP
                if len(precioP) > 6:
                    try:
                        precioP = precioP.split('-')
                        precioP = precioP[0]
                        precioRealP = precioP
                    except Exception:
                        pass
                """
                list_htmls = htmltext.split('"')
                precioP = list_htmls[list_htmls.index('price') + 2]
                try:
                    precioP = precioP.split('-')[0]
                except Exception:
                     precioP = list_htmls[list_htmls.index('price') + 2]
                precioRealP = precioP
                print "Precio: %s"%(precioP)

                #link directo al producto
                linkP = current_url

                #codigo producto
                list_c = htmltext.split(' ' and ';')
                codigoP = list_c[list_c.index('\r\n\tinditex.iCountryCode =
"ES"') + 5]
                codigoP = codigoP.replace("\r","")
                codigoP = codigoP.replace("\n","")
                codigoP = codigoP[codigoP.index('=') + 1:]
                codigoP = codigoP.replace(" ","")
                print "Codigo: %s"%(codigoP)

                #comprobamos si es novedad
                if 'nuevo' in main_url:
                    novedades = 'TRUE'
                else:
                    novedades = 'FALSE'
                try:
```

```
                    #insertamos los datos en la tabla SQL
                    var_sql = """insert into crawlerVenca
(Client,Category,Brand,SubBrand,Description,NEW,PriceBeforeDiscount,ActualPr
ice,URL,Link,CodeProd,ID_Presentation,Execution_Time,Top20,Crawl_Day,
Crawl_Date) values (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"""
                    cursor.execute(var_sql,
(client,categoria,brand,subcategoria,nombreP,novedades,precioRealP,precioP,m
ain_url,linkP,codigoP,current_cont,fecha_ejecucion,top,timeIni,timeP))
                    cursor.commit()

                except Exception,e:
                    f=open(filename,"a")
                    #write error to file:
                    f.write("Error occured:\n" + current_url + " <------> "
+ str(e) + "\n") # some message specific to this iteration (page) should be
added here...
                    #close error log file:
                    f.close()

                print "current cont %s/%s"%(current_cont,productos)

        except Exception,e:
            f=open(filename,"a")
            #write error to file:
            f.write("Error occured:\n" + current_url + " <------> " + str(e)
+ "\n") # some message specific to this iteration (page) should be added
here...
            #close error log file:
            f.close()
            fallos.add(current_url)


        current_cont += -1
        contP += 1
        print "Productos %s/%s"%(contP,productos)
        if contP == productos:
            final += 1
            print "Escaparate crawleado con exito"

    contE += 1
    print "Crawleado escaparate %s/%s"%(contE,escaparates)
    print "Con exito: %s/%s"%(final,escaparates)

print "Crawler de %s ha finalizado correctamente"%(brand)
print "Lista de fallos: %s"%(fallos)
cursor.close()
```

Zara

```
   # -*- coding: utf-8 -*-
"""

@author: joelsanchez
"""

import time
import re
import os
import pyodbc
import requests
from ordered_set import *
from bs4 import BeautifulSoup
from datetime import datetime
from selenium import webdriver
pmName = 'ordered_set'
pm = __import__(pmName)
#i = datetime.now()
#Funcion para saber si un string contiene numeros, en algunas webs nos
sirve para comprobar si un link es de producto o no
def hasNumbers(inputString):
    return bool(re.search(r'\d', inputString))

#headers para simular que entramos desde Mozilla y no nos de 'Acces
denied'
headers = {'User-agent': 'Mozilla/5.0'}

#conectamos a SQL
cnxn = pyodbc.connect('DRIVER={SQL Server Native Client
10.0};SERVER=KOSMOS;DATABASE=MKT_CRAWLER;UID=[user];PWD=[password]')

cursor = cnxn.cursor()

fecha_error = time.strftime("%Y%m%d")
fecha_ejecucion = time.strftime("%Y%m%d %H:%M:%S")
timeIni = datetime.now()
cont = 0
final = 0
topMax = 20
errores = 0
fallos = set()
client = 'Venca'
brand = 'Zara'
filename= "S:/Consultas_python/CrawlerVenca/Errores/" + fecha_error + "_"
+ brand + "_Errors.txt"

word = '/mujer/'

connected = False
while connected == False:

    try:
        driver = webdriver.Chrome()#'C:/chromedriver.exe'
        time.sleep(10)
        urls = {'https://www.zara.com/es/'}
        current_url = urls.pop()
        driver.get(current_url)
```

```
        time.sleep(10)
        driver.execute_script("window.scrollTo(0,
document.body.scrollHeight);")
        time.sleep(2)
        html2 = driver.execute_script("return
document.documentElement.innerHTML;")
        soup= BeautifulSoup(html2)
        connected = True

    except Exception:
        pass

#recorremos la URL y extraemos todos los links, filtramos por los de
producto mirando si contiene HTML y numeros, y lo guardamos en un set, para
no tener repetidos y recorrer luego
for tag in soup.findAll('a', href=True):
    if (word in tag['href'] or '/trf/' in tag['href']) and 'tarjeta-regalo'
not in tag['href']:
        product = bool(re.search('p[0-9]{6}', tag['href']))
        if product == False:
            if "https" in tag['href']:
                urls.add(tag['href'])
            else:
                new_url = "https:" + tag['href']
                urls.add(new_url)
driver.quit()

escaparates = len(urls)
contE = 0
#bucle de escaparates, mientras queden links sigue haciendo
while urls:

    #sacamos el primer escaparate
    current_url = urls.pop()
    #nos quedamos con su link
    main_url = current_url
    print "Crawling current_url : %s"%(current_url)
    #entramos desde Mozilla y leemos la web y cogemos su codigo html
    webpage = requests.get(current_url, headers=headers)
    htmltext = webpage.content
    soup= BeautifulSoup(webpage.content)
    word = 'html'
    sinDuplicados = OrderedSet()
    sinDuplicados.clear()
    contador = OrderedSet()
    contador.clear()
    for tag in soup.findAll('a', href=True):
        product = bool(re.search('p[0-9]{3}', tag['href']))
        if product == True:
            if 'https:' not in tag['href'] and 'tarjeta-regalo' not in
tag['href']:
                new_url = 'https:'+tag['href']
                sinDuplicados.add(new_url)
            else:
                sinDuplicados.add(tag['href'])
            cont +=1
            #contador.add(cont)
    cont = 0
    current_cont = len(sinDuplicados)
    productos = len(sinDuplicados)
    contP = 0
```

```
     #bucle parar recorrer los productos del escaparate, mientras queden
productos en ese escaparate sigue recorriendo
    while sinDuplicados:
        try:
            word = '-c'
            #cogemos el primer producto del set
            current_url = sinDuplicados.pop()
            linkP = current_url
            print "Crawling", main_url
            #no se nos va a colar ningun link aqui con la busqueda
exhaustiva que hemos realizado previamente, pero por si acaso miramos que el
link contenga -c
            if word in current_url:

                timeP = datetime.now()
                #leemos la web y su codigo html, simulando que somos Mozilla
                webpage = requests.get(current_url, headers=headers)
                htmltext = webpage.content
                soup= BeautifulSoup(webpage.content)
                print "Parsing", current_url

                #current_cont = contador.pop()
                if current_cont <= topMax:
                    top = 'TRUE'
                else:
                    top = 'FALSE'

                try:
                    #sacamos categoría
                    categoría = soup.find('li', itemtype="http://data-
vocabulary.org/Breadcrumb").getText()
                    categoría = categoria.replace("\r","")
                    categoría = categoria.replace("\n","")
                    categoría = categoria.replace(">","")
                    categoría = categoria.encode('utf-8')
                    categoría = categoria.replace('\t','')
                    categoría = categoria.replace('  ','')


                    #sacamos subcategoria
                    subcategoria = soup.find('li', itemtype="http://data-
vocabulary.org/Breadcrumb")
                    subcategoria = subcategoria.findNext('li',
itemtype="http://data-vocabulary.org/Breadcrumb").getText()
                    subcategoria = subcategoria.replace("\r","")
                    subcategoria = subcategoria.replace("\n","")
                    subcategoria = subcategoria.replace(">","")
                    subcategoria = subcategoria.encode('utf-8')
                    subcategoria = subcategoria.replace('\t','')
                    subcategoria = subcategoria.replace('  ','')

                except Exception:
                    categoría = "None"
                    subcategoria = "None"
                print "Categoria: %s"%(categoría)
                print "Subcategoria: %s"%(subcategoria)

                try:
```

```
                    #nombre producto
                    nombreP = soup.find('h1').getText()
                    nombreP = nombreP.replace("\r","")
                    nombreP = nombreP.replace("\n","")
                    nombreP = nombreP.replace("\t", "")
                    #nombreP = nombreP.encode('utf-8')
                    nombreP = nombreP.replace('\t','')
                    nombreP = nombreP.replace('  ','')
                except Exception:
                    nombreP = soup.find_all('meta', { 'name' :
"description"})[0]['content']
                    #nombreP = nombreP.encode('utf-8')
                    nombreP = nombreP.replace('\t','')
                    nombreP = nombreP.replace('  ','')
                print "Nombre: %s"%(nombreP)
                #precio producto
                list_htmls = htmltext.split('"')
                try:
                    precioP = soup.find_all('span', { 'class' :
"price" })[0]['data-price']
                    precioP = precioP[:-3]
                    precioP = precioP.replace("\t","")
                    precioP = precioP.replace(",",".")
                    precioRealP = precioP
                except IndexError:
                    try:
                        precioP = soup.find_all('span', { 'class' :
"sale"})[0]['data-price']
                        precioP = precioP.replace(",",".")
                        precioP = precioP[:-5]
                        precioRealP = soup.find_all('span', { 'class' :
"line-through"})[0]['data-price']
                        precioRealP = precioRealP.replace(",",".")
                        precioRealP = precioRealP[:-5]
                    except Exception:
                        precioP = list_htmls[list_htmls.index('price')+1]
                        precioP = precioP.replace(':','')
                        precioP = precioP.replace(',','')
                        precioP = precioP[:-2] + '.' + precioP[-2:]
                        try:
                            precioRealP =
list_htmls[list_htmls.index('price') + 3]
                            precioRealP = precioRealP.replace(':','')
                            precioRealP = precioRealP.replace(',','')
                            precioRealP = precioRealP[:-2] + '.' +
precioRealP[-2:]
                            if hasNumbers(precioRealP) == False:
                                precioRealP = precioP
                        except Exception:
                            precioRealP = precioP
                print "Precio sin descontar: %s"%(precioRealP)


                print "Precio: %s"%(precioP)
                    #link directo al producto

                try:
```

```
                    #codigo producto
                    codigoP = soup.find_all('input', { 'name' :
"parentId"})[0]['value']
                except Exception:
                    list_url = current_url.split('-')
                    codigoP = list_url[len(list_url)-1]
                    codigoP =
codigoP[codigoP.index('p')+1:codigoP.index('.')]
                print "Codigo: %s"%(codigoP)

                #comprobamos si el escaparate marcaba el producto como
novedad
                mainpage = requests.get(main_url, headers=headers)
                soup2= BeautifulSoup(mainpage.content)
                try:
                    li = soup2.find('li', id=iden)
                    new = li.find('span' , { 'class' : "label-new"})
                    if new == None:
                        novedades = "FALSE"
                    else:
                        novedades = 'TRUE'
                except Exception:
                    novedades='FALSE'

                #nmero de colores
                colores = 'name="color"'
                ncolores = htmltext.count(colores)
                print "Colores: %s"%(ncolores)

                #comprobamos las tallas minima y maxima, unica en caso de no
tener tallas
                sinTallas = soup.find('span' , {'class' : "size-name"})
                if sinTallas == None:
                    tallaMin = 'Unica'
                    tallaMax = 'Unica'
                else:
                    tallaMin = soup.find('span' , {'class' : "size-
name"}).getText()
                    tallaMin = tallaMin.replace("\t","")
                    tallaMin = tallaMin.replace("\n","")
                    tallaMin = tallaMin.replace("\r","")
                    tallaMin = tallaMin.replace(" ","")
                    tallaMin = tallaMin + ""


                    tallaMax = soup.findAll('span' , {'class' : "size-
name"})
                    last_div = None
                    for last_div in tallaMax:pass #bucle para recorrer todas
las tallas y coger la ultima
                    if last_div:
                        tallaMax = last_div.getText()
                        tallaMax = tallaMax.replace("\t","")
                        tallaMax = tallaMax.replace("\n","")
                        tallaMax = tallaMax.replace("\r","")
                        tallaMax = tallaMax.replace(" ","")
                print "Talla Min: %s"%(tallaMin)
                print "Talla Max: %s"%(tallaMax)
```

```
                var_sql = """insert into crawlerVenca
            (Client,Category,Brand,SubBrand,Description,NEW,PriceBeforeDisco
            unt,ActualPrice,URL,Link,MinSize,MaxSize,CodeProd,ID_Presentatio
            n,Execution_Time,#Colors,Top20,Crawl_Day, Crawl_Date) values
            (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"""
                cursor.execute(var_sql,
(client,categoria,brand,subcategoria,nombreP,novedades,precioRealP,precioP,m
ain_url,linkP,tallaMin,tallaMax,codigoP,current_cont,fecha_ejecucion,ncolore
s,top,timeIni,timeP))
                cursor.commit()
                print "current cont %s/%s"%(current_cont,productos)

        except Exception,e:
            f=open(filename,"a")
            #write error to file:
            f.write("Error occured:\n" + current_url + " <------> " + str(e)
+ "\n") # some message specific to this iteration (page) should be added
here...
            #close error log file:
            f.close()
            fallos.add(current_url)
            errores += 1

        current_cont += -1
        contP += 1
        print "Productos %s/%s"%(contP,productos)
        if contP == productos:
            final += 1
            print "Escaparate crawleado con exito"

    contE += 1
    print "Crawleado escaparate %s/%s"%(contE,escaparates)
    print "Con exito: %s/%s"%(final,escaparates)



print "Crawler de %s ha finalizado correctamente"%(brand)
print "Lista de fallos %s"%(fallos)
cursor.close()
```