Books2Scrape

# Objetivo

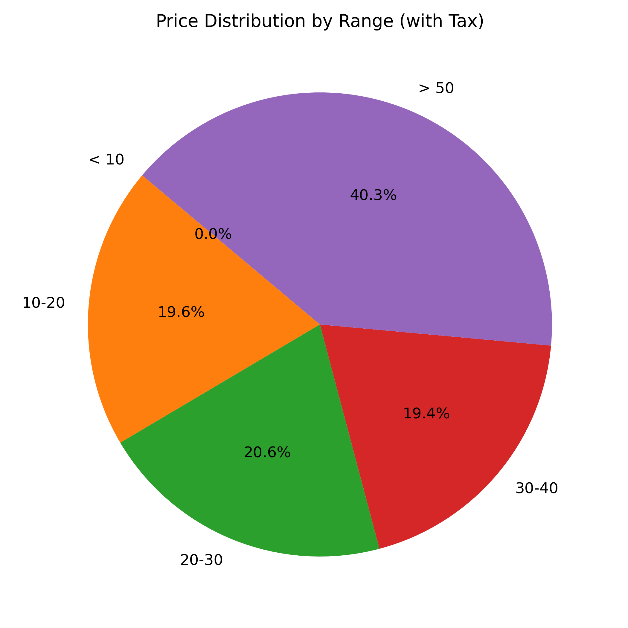
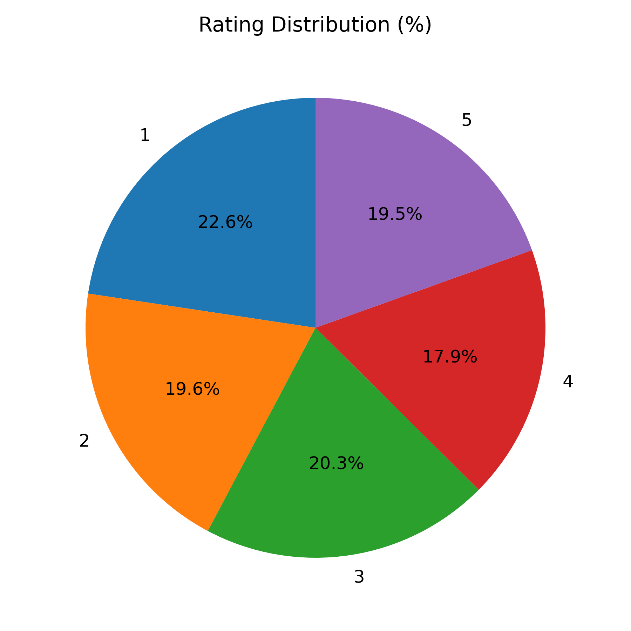
Realizar una aplicación que “scrapee” los datos de la página http://books.toscrape.com/ con el fin de procesar los mismos, para crear informes y graficas. Para la creación de esta herramienta, se tuvo en cuenta una gran parte de las utilidades que nombra el ejercicio, enumerándolas:

* Webscraping: la información se obtuvo mediante extracción de datos vía webscraping, utilizando la librería Scrapy para la obtención de la mayoría de la información de los libros, y BeautifulSoup4 para las categorías. Se utilizo Scrapy debido a su velocidad, y al ser una página de pocos recursos visuales o de backend, lo hace aún mejor, su escalabilidad y funciones. Utilice BeautifulSoup4 para obtener las categorías ya que era más fácil para obtener unas pocas líneas de información.
* Limpieza y transformación: Se utilizo Pandas para realizar la limpieza y transformación de datos, debido a su simpleza a la hora del manejo de los mismos. Es una librería eficiente, rápida, flexible, autocorrectiva, y de gran legibilidad. A futuro para datasets más pesados se podría usar otras librerías como Polar, Spark o su idioma nativo (NumPy)
* Reportes y agregación: Para la hora de crear graficas se utilizó MatplotLib, ya que te permite mostrar graficas sobre código, flexible y customizable, y Seaborn esta para creación de gráficos mas relacionados con la parte de estadísticas. Luego para la creación de agregación o “pivot tables” se utilizo pandas, por los motivos nombrados anteriormente.
* Estrategia contra bloqueos de datos: Si bien la pagina no contaba con un antibot, se tomaron ciertas medidas para no ser bloqueado, entre ellas:
  + User-agents: cada pedido de información a la pagina puede usar aleatoriamente uno de tres agentes declarados, además del propio.
  + Proxies: cada pedido de información a la página puede usar aleatoriamente uno de los multiples proxies declarados, además del propio.
  + Intentos: Cada URL tiene 2 intentos, lo que le permite hacer dos combinaciones de proxy + user-agent
  + Retardo: Tiene seteado por default entre 0.1 y 0.2 segundos, además de esto tiene activado la configuración “AUTOTHROTLE” lo que le permite elegir el retraso conveniente entre un pedido y otro (configurado entre 0.1 y 1).
  + Optimización: con todos los ajustes de seguridad anteriormente nombrados, el proceso se hizo exponencialmente mas lento, pasando de ~1 minuto, a más de 1 hora, por lo que se habilitaron, configuración de multiproceso de pedidos, y pedidos en paralelo.
* Desafíos: Realmente mi mayor desafío fue no conocer la librería Scrapy, al ser algo diferente a lo que estoy acostumbrado tuve que aprender los fundamentos rápidamente, y entender como era el manejo interno de clases y configuraciones de la misma. En cuanto a los datos, no hubo grandes desafíos, para obtener mas datos para trabajar, realizo un segundo webscraping de cada URL de libro, para luego crear un archivo final con más información, que luego será utilizado para crear reportes.
* Limpieza y transformación: Para la misma se utilizo pandas, ya que es una librería simple, pero muy eficiente, de fácil legibilidad, compatibilidad y autocorrectiva en algunos casos. A futuro con una medida mas grande de datos se podría migrar a Polar
* Capacidad de procesamiento y escalabilidad: Para un mejor rendimiento se evito el uso de bucles innecesarios, para procesar información se utilizó código vectorizado, para evitar crear archivos de 0 se creo una función que agrega las nuevas líneas al archivo final, utilizando el campo “UPC”. El código este habilitado para ser escalable al usar rutas relativas, código modularizado, un proceso de instalación sencillo, y ajustes optimizados para una correcta extracción, permiten que el código se ejecute relativamente rápido, y que además sea fácilmente tweakable.
* Buenas prácticas: Como se nombró en este informe, se utilizaron varias buenas practicas a la hora de hacer este código, entre ellas: código modularizado, manejo de errores, rutas relativas, ajustes personalizados, limpieza de datos, manejo correcto de clases, código con comentarios cortos pero útiles, y se evita código repetido
* Documentación: Se creo un repositorio en GitHub con instrucciones sobre como descargar, instalar dependencias, y ejecutar el código.

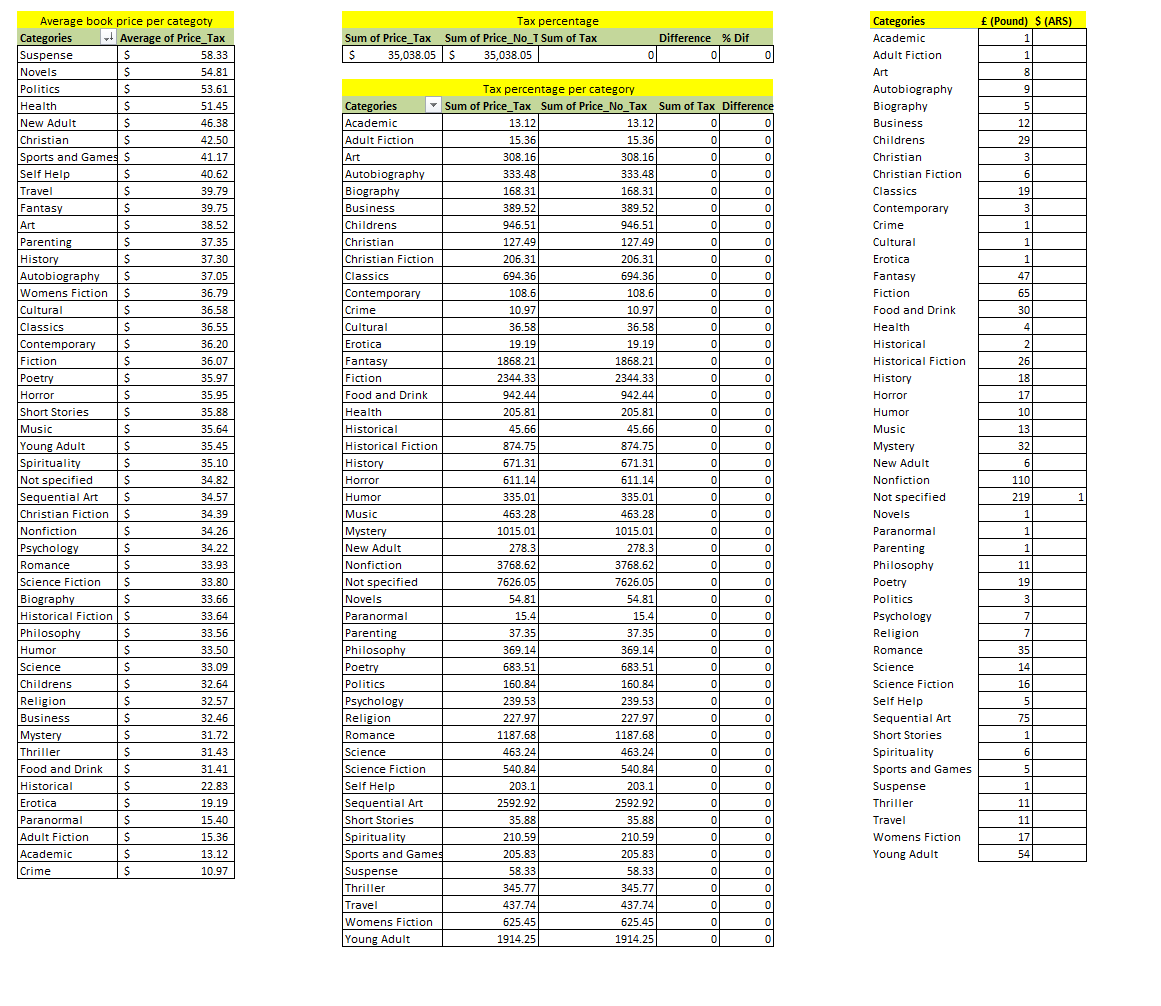
# Resultados

Al realizar la extracción, limpieza y preparación de datos obtuvo un archivo crudo final, el cual posee los siguientes campos UPC, Category, Title, Currency, Price\_Tax ,Price\_No\_Tax, Tax, Stock\_availability, Stock\_quantity y URL. Con estos se crearon múltiples reportes entre ellos:

* Promedio de precio por categoría
* Mejores libros económicos
* Mejores libros caros
* Mejores libros con poco stock
* Mejores categorías con poco stock
* Cantidad de libros por categoría
* Dos graficas que se adjuntan a continuación:



Nota: Si se hubiese tenido información más variada, se podrían haber generado mas reportes, entre ellos los siguientes ejemplos



# Mejoras y siguientes pasos

Al tener un tiempo acotado, creo que hay ciertas cosas del proyecto que se podrían haber mejorado, entre ellas la principal seria implementar una base de datos de algún tipo (en el caso de ser algo masivo, se recomendaría un Spark o similares), como se nombró anteriormente, utilizar polar o numpy para mayor eficiencia, agregar mas control de error tanto para Airflow como para el código en jupyter notebook, en vez de crear reportes en Excel haría que ejecutara una api de powerBI para actualizar los datos y mostrar los reportes alli