Escuela Técnica Superior de Ingeniería Informática

Ingeniería Informática. Ingeniería del Software

TRABAJO DE FIN DE GRADO

Visualización de Datos en la Ingeniería del Software

Autor/es:
Daniel Carpio Camacho

Tutor/es:
José Ángel Galindo Duarte

Primera Convocatoria

Curso 2018/2019

Resumen

La elaboración de gráficas para la representación de datos es una práctica muy habitual en la actualidad. Y a pesar de la gran cantidad de situaciones en las que están presentes, gran cantidad de personas cometen errores que afectan a cómo vemos y entendemos dicha representación. Estos pueden ser el uso de determinados colores o incorrectas representaciones de los datos. La idea de presentar los datos de una forma más "elaborada" concluye en gráficos complejos que resultan incomprensibles, haciendo que no aporten nada a la hora de mostrar datos.

Si comprobamos el uso que se les da a los gráficos en la web, el problema no parece ir a mejor. Existen numerosas librerías que te permiten crear gráficas en páginas web, pero como veremos, no todos ellos te permiten mostrar los datos de forma óptima. Entre estas librerías existen algunas bastante conocidas, por ejemplo amCharts, conocida por crear gráficos bastante llamativos pero que no cumplen con algunas conclusiones que veremos en este trabajo. Por otro lado, tenemos aquellas que sí que te permiten hacerlo de manera correcta pero que, al disponer de una amplia configuración, a la hora de que un desarrollador vaya a utilizarlo puede hacerlo de manera incorrecta. En este caso podemos destacar Google Charts, que proporcionan una configuración enorme, pero los usuarios fallan al usarla.

Para solventar este problema, estudiaremos cómo se deben realizar las gráficas de manera correcta en base a trabajos anteriores como "Show me the numbers" de Stephen Few y posteriormente implementaremos una librería en JavaScript que nos permite añadir gráficas a nuestra página web. Pero, a diferencia de las demás librerías, estás gráficas tendrá la mayoría de dichas configuraciones ya establecidas facilitando que se puedan comprender los datos de manera más correcta.

En este documento detallaremos más a fondo cómo se han elaborado estás gráficas, tanto los detalles previamente explicados como la implementación en código de las mismas. Además, veremos cómo usarse en el desarrollo de páginas web, incluyendo una aplicación web donde se usan.

Igualmente, se mostrará la implementación de un plugin en WordPress

para facilitar el uso de esta librería en este sistema.

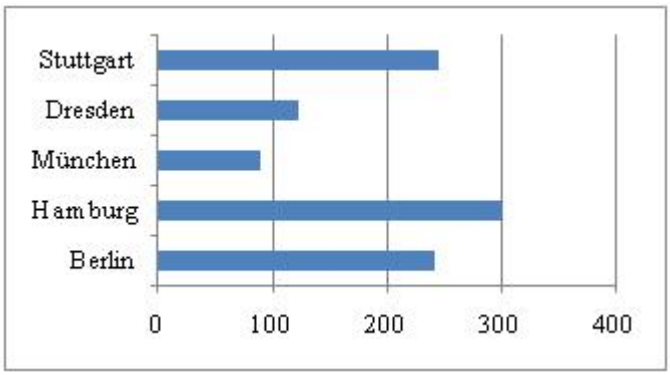
Índice

Chapter 1

Prefacio

1.1 Introducción

Uno de las técnicas más usadas para mostrar información de una forma simple y directa es mediante el uso de diferentes gráficos. Existen una multitud de tipos bastante amplia de ellos y cada uno puede aportar la misma información de una forma distinta, destacando diferentes aspectos. Entre estos podemos destacar los gráficos de barras o los gráficos circulares, ampliamente conocidos por todo el mundo.



Así mismo, un mismo gráfico se puede realizar de formas distintas, dado que podemos variar el color, los márgenes, la fuente, el número de datos, el rango en los valores en los ejes, etc. Modificar cualquiera de estos valores hace que la gráfica se contemple de manera distinta, incluso más de lo que pueda parecer a primera vista.

Ni todos los tipos de gráficos ni todas las configuraciones son aceptables. En esto entran detalles de cómo las personas comprendemos los datos que nos llegan a través de los ojos. Instintivamente nuestra mirada tiende a apuntar

hacia colores más llamativos. O las personas no somos capaces de comparar áreas con la misma facilidad que con la que comparamos distancias.

Además, debemos destacar la presencia de personas con discapacidad visual. Pues debemos destacar problemas como el daltonismo. El hecho de usar unos colores u otros puede afectar gravemente para ellos, pues dos colores que a se ven distintos, para una persona con daltonismo se puede ver prácticamente iguales. Y si en la gráfica que se está mostrando existe una alta dependencia de los colores, sería muy importante fijarse en qué colores se están usando.

Detalles como estos pueden hacer que directamente se deba rechazar varios tipos de gráficos y varias configuraciones de otros tipos.

1.2 Motivación

Actualmente, la forma más común de mostrar datos de una forma sencilla consiste en la representación en gráficos. Esto permite mostrarlos de una manera sencilla, y como además de que existen gran cantidad de maneras distintas de hacer esto, permitiendo enfocarnos en distintos aspectos de los mismos.

No obstante, a pesar de ser esto, se cometen distintos errores a la hora de crear estas representaciones, lo que provoca que no se comprenda el contenido del mismo, haciendo que dicha gráfica sea totalmente inútil.

Además, debemos destacar un problema que se produce en determinadas personas. En concreto al 10% de todos los hombres y 1% de las mujeres: el daltonismo. La elección de colores es importante para que todas las personas puedan verlo de manera correcta. Con frecuencia se usan determinados colores para que vayan en conjunto a la estética de la página, sin tener en cuenta en si dichos colores se verían correctamente para personas con dicha discapacidad visual.

Por ello, este trabajo recogerá un determinado número de consejos y advertencias que se deben cumplir para ayudar a transmitir de manera correcta la información deseada.

También, cuando observando las distintas librerías gráficas que se encuentran disponibles en la web, vemos que a pesar de ser muy completas y personalizables, apenas aportan dichas advertencias e incluso que permiten al usuario cometer errores fácilmente detectables a nivel de código. Por ello crearemos una librerías en JavaScript que nos permitirá crear gráficas en la web siguiendo dichas advertencias, siempre que nos sea posible.

Para comprobar cómo se vería dicha librería en un entorno real, crearemos una aplicación web donde se usará dicha librería con datos reales, donde

consumiremos una API para obtener dichos datos y poder mostrarlos por pantalla.

Además, para facilitar el uso de la misma, añadiremos la opción de que se pueda añadir fácilmente a WordPress mediante un plugin, para que se pueda usar de manera más cómoda.

1.3 Objetivos

El objetivo de este proyecto es desarrollar una librería que permita al desarrollador web añadir gráficos de una forma sencilla. Además, estos gráficos deben cumplir ciertas normas para que su elaboración sea correcta.

Para mostrar el uso de estas gráficas en un proyecto real, se desarrollará una aplicación web en Django que utilice la librería. Esta aplicación web además estará desplegada en Heroku. No obstante, para facilitar el despliegue de esta aplicación web, estará también disponible en un contenedor de Docker y subido a Docker Hub, de tal manera que solo haya falta escribir un par de comandos para poder lanzarlo.

Además, para facilitar el uso de la librería, se implementará un plugin en WordPress que permita utilizarse de manera sencilla.

En paralelo a los objetivos principales ya comentados, existen una serie de objetivos de carácter docente que se explican a continuación.

1.3.1 Objetivos docentes

1. Mejorar la búsqueda de soluciones a un problema que se pueda encontrar dentro de la ingeniería del software.
2. Mejorar la planificación y seguimiento de proyectos informáticos.
3. Mejorar el cálculo de presupuestos para proyectos informáticos.
4. Afianzar el uso de los lenguajes de programación JavaScript, Python y PHP.
5. Estudio y uso de la librería Data-Driven Documents o D3Js.
6. Estudio y manejo del framework Django.
7. Despliegue de aplicaciones web en servicios PaaS, en este caso Heroku.
8. Aumentar los conocimientos de estudio y consumo de APIs REST.
9. Creación de un proyecto en WordPress.

10. Estudio y creación de plugins en WordPress.

11. Crear y creación de imágenes Docker, así como creación de ficheros Dockerfile y docker-compose.

Chapter 2

Materias relacionadas

2.1 Herramientas de gestión y desarrollo

Durante el desarrollo del proyecto, nos ayudaremos de distintas herramientas que nos ayudarán durante toda la planificación e implementación.

Para la planificación usaremos el software Planner. Esta herramienta, de codigo abierto y disponible tanto para sistemas operativos Window como tipo UNIX, nos permite detallar y planificar proyectos, así como poder visualizar de una manera cómoda toda la línea temporal de la misma.

Para el control del tiempo, usaremos Toggl, donde nos irá generando un registro de cuántas horas se han invertido en qué tarea y cuándo se han invertido. Nos será muy útil para poder comparar los datos reales con la planificación antes comentada.

Para el control del código, nos ayudaremos de Git, una herramienta de control de versiones, que usaremos en un repositorio que tendremos alojado en GitHub.

En determinadas partes que comentaremos más adelante, nos ayudaremos de Docker para el despliegue de aplicaciones, usando imágenes como MySQL o WordPress.

Como editor de código, usaremos Visual Studio Code, editor de código desarrollado por Microsoft disponible tanto en Windows como Linux y Mac.

2.2 Tecnologías

Durante el desarrollo de este proyecto, se usarán las tecnologías:

- Javascript: Lenguaje de programación orientado a objeto, usado principalmente en el lado del cliente para añadir mejoras a la interfaz de usuario.

- Python: Lenguaje de programación que destaca principalmente por su código legible. Posee una enorme cantidad de librerías que ayuda enormemente al desarrollo de software.
- PHP: Lenguaje de programación cuyo uso se centra en el lado del servidor para generar páginas web con contenido dinámico.
- Data-Driven Documents: Librería para JavaScript que nos permite tratar datos para así crear de una manera sencilla gráficos en SVG en páginas web.
- Django: Framework de código abierto, escrito en Python, para crear aplicaciones web de una manera sencilla y rápida.
- Heroku: Plataforma que nos proporciona servidores web para poder alojar nuestras aplicaciones.
- Git: Software de control de versiones donde podremos registrar todos los cambios que hagamos en nuestro código.
- GitHub API: API proporcionada por GitHub que nos muestra datos de los repositorios alojados en su web.
- Docker: Software que nos permite empaquetar aplicaciones dentro de contenedores para poder ejecutarse en distintas máquinas de manera cómoda.
- WordPress: Gestor de contenido enfocado a crear páginas web de cualquier tipo. Actualmente tiene una gran presencia en la web.

2.3 Participantes en el proyecto

Para este proyecto, únicamente habrá un desarrollador: Daniel Carpio Camacho, estudiante de Ingeniería Informática - Ingeniería del Software de la Universidad de Sevilla.

2.4 Glosario de términos

- Gráfica: Representación visual de datos numéricos que permite mostrar la relación.
- Visualización de datos: Búsqueda e interpretación de un conjunto de datos que permita obtener información útil para el usuario.

- Librería: Conjunto de funciones con una interfaz bien definida cuya finalidad es ser usada en otros software.
- PaaS: Platform as a Service. Servicios en la nube que proporcionan equipos informáticos para desplegar aplicaciones web, entre otras muchas funciones.
- Wrapper: Método cuya función es llamar a un segundo método, pero simplificando el uso de esta.

Chapter 3

Planificación

3.1 Objetivos funcionales del proyecto

1. Obj-01: Estudiar distintos tipos de gráficos existentes y determinar cuáles son sus formas correctas de representarlas. Definiremos el diseño que usaremos para nuestras librería, tras estudiar qué factores afectan para la interpretación de las mismas.

2. Obj-02: Determinar la estructura de los datos de entrada para el uso de la librería. Definiremos, usando el formato JSON, qué estructura deberán tener los datos de entrada a la hora de representarlos, intentando que sean los más sencillos y coherentes para el usuario.

3. Obj-03: Estudio de la librería D3Js. Estudiaremos el funcionamiento y herramientas que nos proporciona la librería D3 para poder crear los gráficos.

4. Obj-04: Creación de la librería en JavaScript para la visualización correcta de los datos. Con los conocimientos obtenidos de la librería en el objetivo 03, crearemos las gráficas con los diseños obtenidos en el objetivo 01.

5. Obj-05: Testeo de la librería. Realizaremos determinadas pruebas para poder comprobar que la librería funcione de la manera esperada.

6. Obj-06: Creación de una aplicación en Django que utilice la librería anteriormente. Esta aplicación utilizará la librería para poder visualizar datos obtenidos mediante la API REST que ofrece GitHub.

7. Obj-07: Estudio de la creación de plugins en WordPress. Estudiaremos cómo funciona WordPress y sus plugins, para poder crear el nuestro y poder usar nuestra librería fácilmente.

8. Obj-08: Creación de un plugin en WordPress para la incorporación de la librería a una página web. Con los conocimientos obtenidos en el objetivo anterior, crearemos dicho plugin.

9. Obj-09: Estudio de Docker. Estudiaremos la tecnología Docker, así como su funcionamiento y qué herramienta nos proporciona.

10. Obj-10: Subir la aplicación implementada en Django a un contenedor en Docker Hub. Para poder facilitar el despliegue de la aplicación, crearemos un contenedor Docker con la aplicación y la subiremos a Docker Hub.

11. Obj-11: Despligue en Heroku. Estudiaremos la plataforma y cómo podremos desplegar aplicaciones web. Después procederemos a desplegar dicha aplicación en Heroku.

12. Documentación: Documentación del TFG.

3.2 Planificación temporal

Las actividades se han programado en periodos de trabajo, cuya duración se han establecidos en función de la carga de trabajo.

La duración de las actividades son:

1. Obj-01: Del 15/10/2018 al 05/11/2018

2. Obj-02: Del 05/11/2018 al 12/11/2018

3. Obj-03: Del 12/11/2018 al 26/11/2018

4. Obj-04: Del 26/11/2018 al 17/12/2018

5. Obj-05: Del 17/12/2018 al 24/12/2018

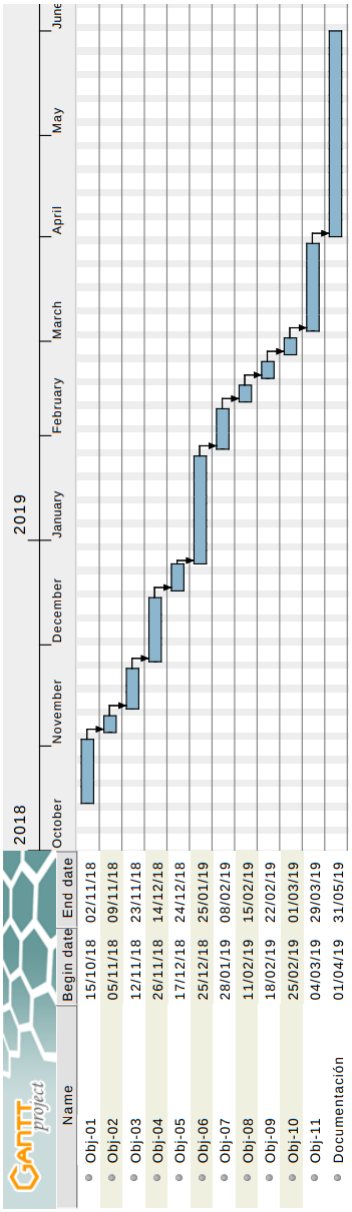
6. Obj-06: Del 07/01/2019 al 28/01/2019

7. Obj-07: Del 28/01/2019 al 11/02/2019

8. Obj-08: Del 11/02/2019 al 18/02/2019

9. Obj-09: Del 18/02/2019 al 25/02/2019

10. Obj-10: Del 25/02/2019 al 04/03/2019
11. Obj-11: Del 04/03/2019 al 01/04/2019
12. Documentación: Del 01/04/2019 al 31/05/2019



3.3 Presupuesto

Con estos datos, podremos estimar un presupuesto para todo el proyecto. Aquí podemos ver los datos con los que haremos los cálculo:

Detalle	Valor
Horas del proyecto	300 horas
Salario por hora	30€/h
Desarrolladores en el equipo	1
Equipo	800€
Seguridad Social	28.3%
Tiempo de amortización	3 años

Con estos datos, realizaremos el presupuesto que a continuación detallaremos.

Dado que el sueldo por hora a un desarrollador es de 30€/h, tenemos un único desarrollador y el proyecto durará 300 horas, podemos ver que el sueldo total a pagar en total será de 9000€.

A estos 9000€ debemos añadirles el coste en impuestos a la Seguridad Social, que en la actualidad se encuentra al 28.3%. Añadiendo esto, nos encontramos con que se irían 12552.30€ en personal, de los cuales 3552.30€ serían los impuestos.

Para nuestro equipo, tenemos un total de 3 años de amortización, lo que se traduce en 36 meses. Para ello, siguiendo una amortización lineal, tendríamos un total de 22.22€ al mes, lo que serían en total 222.20€ al final del proyecto como coste de amortización.

Con todo esto, el presupuesto total de nuestro proyecto sería:

Detalle	**Valor**
Sueldos	9000.00€
Seguridad Social	3552.30€
Amortización de equipos	222.20€
Total (sin IVA)	12774.50€
Total	15457.15€

Chapter 4

Estudio previo

4.1 Introducción

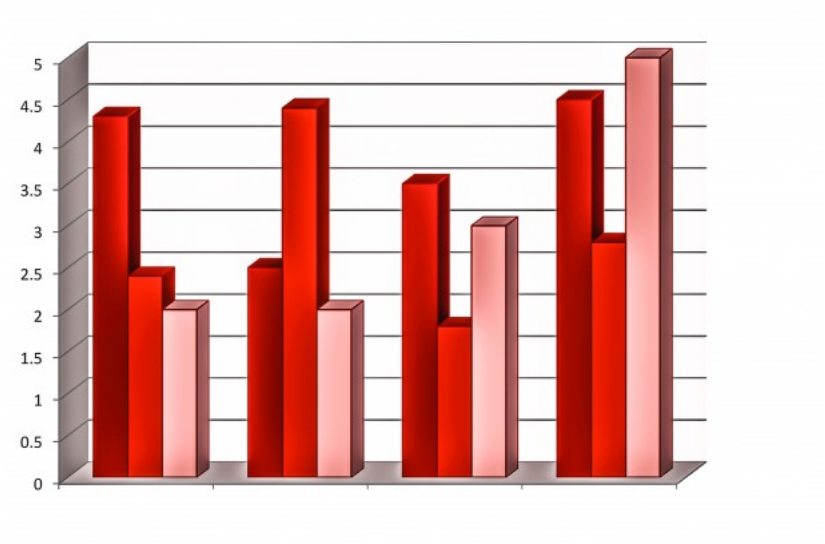
Antes de empezar la implementación de nuestro proyecto, realizaremos un estudio sobre cómo debemos realizar las gráficas para conseguir nuestros objetivos. Para ello, usaremos como referencias los trabajos de Show Me the Numbers [1], Visualización en la ingeniería del software. Aplicación a SonarQube[2] o How to Become the MacGyver of Data Visualizations [3].

Con la ayuda de la información de estos trabajos comentados, redactaremos las bases de nuestra implementación. Además de esto, ampliaremos la información con datos relativos a detalles que proporcionarán ayuda a personas con discapacidad visual.

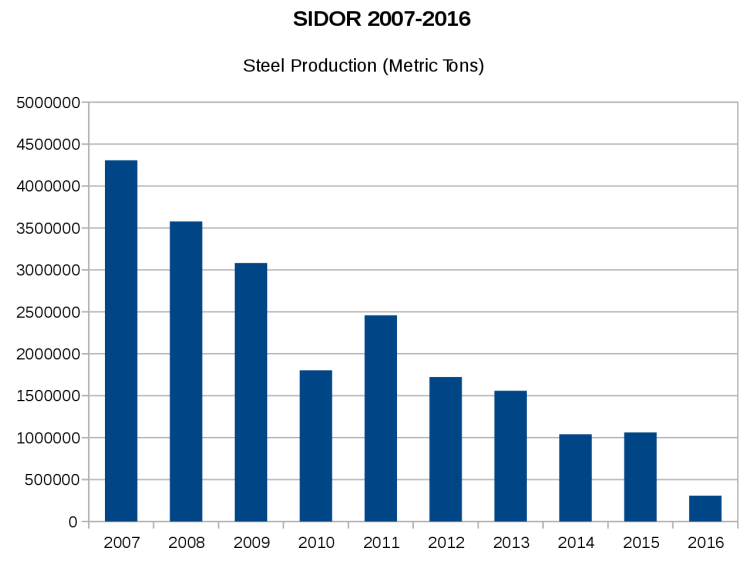
4.2 Gráficas 3D

Este error se encuentra presente en todas las representaciones que incluye barras, que como veremos más adelante, existen varios.

La idea de convertir los rectángulos de la representación 2D en ortoedros en una representación 3D no aporta nada sino confundir a la hora de comparar el valor de la altura de la figura con el valor del eje.



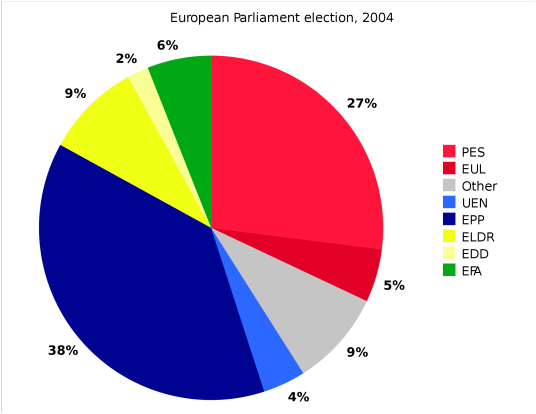
Por ello, toda representación que encontraremos será en 2D para evitar la confusión que pueda ocasionar.



4.3 Gráficas de área

Esto podrá sorprender a más de una persona, pero todas las gráficas cuya comparación se base en áreas debería de reducirse lo máximo posible.

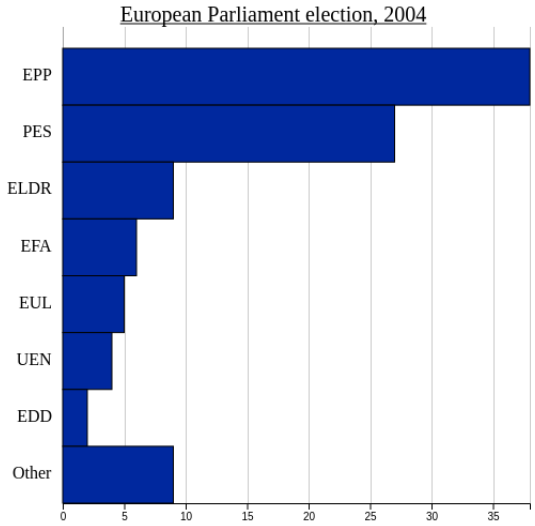
Entre ellos van incluidos los gráficos de área, gráficos circulares o gráficos de donuts.



Para comprender esto debemos entender que el ojo humano funciona mejor comparando líneas que comparando áreas.

Si nos fijamos en la gráfica anterior, podemos apreciar la dificultad de comparar los valores que representan el 5% y el 6% del total. Entonces, ¿de verdad es necesario usar un gráfico circular?

A continuación vamos a mostrar otro gráfico que representan los mismos datos, pero esta vez usaremos un gráfico de barras horizontales.



Como podemos apreciar, se pueden comparar mejor los valores, incluso esas diferencias tan poco notables en las gráficas pastel.

Por esto, no es nada recomendable el uso de gráficos que se basen en comparar áreas.

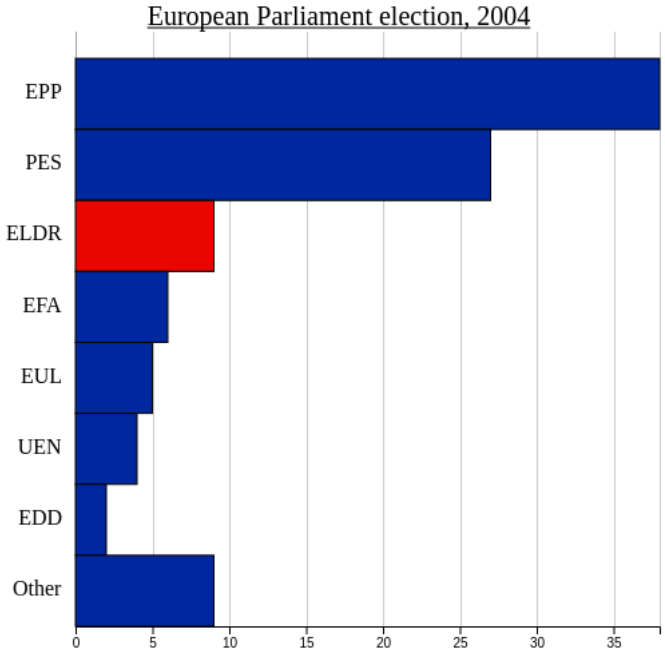
4.4 Uso de colores

El uso adecuado de colores es un tema que los desarrolladores web no suelen tener en cuenta. En la actualidad, casi un 10% de los hombres y el 1% de las mujeres sufren daltonismo [5].

Este problema es independiente al tipo de gráfica que se utilice, lo que provoca que se pueda cometer con más frecuencia. Es por ello que la elección de colores para esta librería estará pensado para que se pueda ver correctamente.

No obstante, también se presenta la opción de que se pueda añadir los colores definidos por el usuario. Es comprensible que se quieran usar unos determinados colores, ya sea por los propios colores de la página y tal.

Si se quieren usar colores definidos por el usuario, es altamente recomendable que estos colores sean de la misma paleta. Eso se debe al grado de impacto que un usuario recibe al percibir los colores. Si tenemos dos colores, uno más llamativo que el otro, a esa persona siempre le llamará la atención el color llamativo, distrayendo la vista del otro.

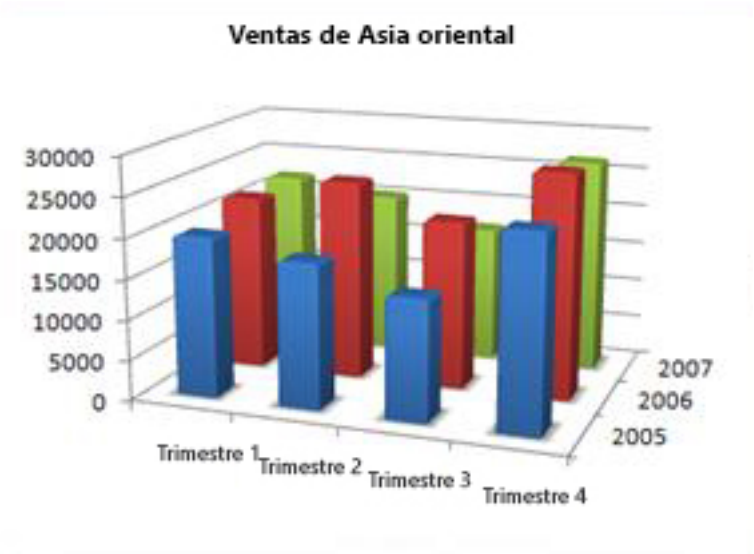


Como se puede apreciar, destaca mucho más la barra roja, tanto por ser el de color diferente como que el color rojo es más llamativo que el azul.

Por ello, recomendamos usar algunas herramientas, como Coolors [7], ya que te permite escoger paletas de colores y te deja visualizarlas para los ocho tipos distintos de daltonismo.

4.5 Principio KISS: Keep it simple, stupid!

El principio KISS es término ampliamente utilizado para describir un principio de diseño. Este acrónimo define que un sistema funciona mejor si es simple. No hace falta complicarlo innecesariamente.



¿Es necesario representar los datos así? ¿Se puede considerar que los datos de la última fila se aprecian correctamente?

Para la creación de gráficas, cuanto más simple, más fácil de comprender es. Y ese es nuestro principal objetivo, después de todo.

4.6 Gráficos de barras

Sin lugar a dudas, se trata de la gráfica más conocida de todas. Consiste en representar los valores numéricos mediante rectángulos cuyas longitudes se encuentran en proporción.

Este tipo de gráficas tiene dos formas distintas de representarse: en horizontal o en vertical. A priori, uno podría pensar que no habría diferencias entre ambas, pero en ciertos casos sí que existen diferencias notables. La diferencia se encuentra en la longitud de la etiqueta que acompaña el rectángulo. ¿Qué ocurre si esa etiqueta es demasiado larga? Si usamos gráficos de barras verticales, las etiquetas acabarían superponiéndose unas a otras.

Algunas personas optan por inclinar la etiqueta 45º o con poner las etiquetas a distintas alturas pero, ¿es necesario? Lo único que consigue es que

sea más complicado de leer. Si simplemente se pusiesen las barras horizontales, se podría permitir que las etiquetas sean más largas sin causar ningún problema.

Uno de los errores que más puede afectar es el de no poner correctamente los rangos en los valores. Una gráfica de barra siempre debe mostrar los datos con respecto al cero. Es la mejor forma de ver la diferencia entre dos valores. En el caso de que no se usase, se puede llegar a pensar que existe una diferencia más grande de la que realmente hay.

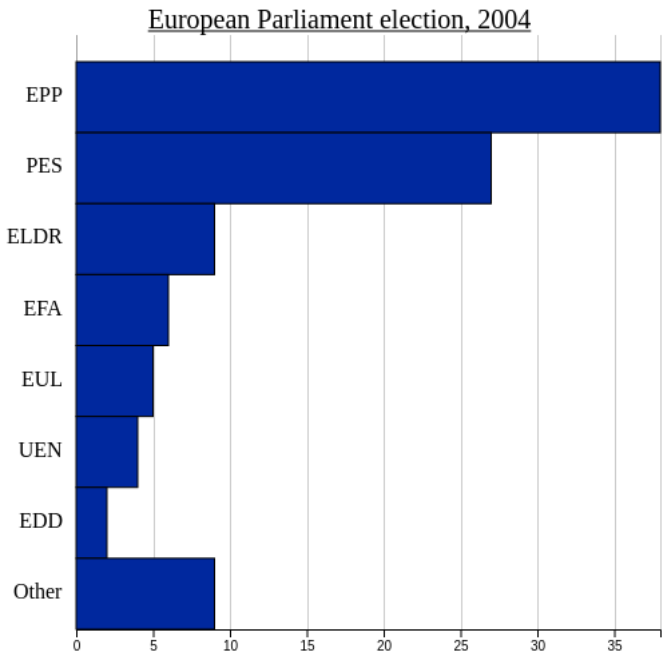
Otro detalle que es importante es la distancia que debe existir entre las barras. Lo ideal sería ponerlas entre 1:0.5 y 1:1.5 con respecto a la anchura de las barras.

También existe la posibilidad de no dejar espacio. Esto se utilizaría si los valores son porcentajes. Es una recomendación, pero no una obligación.

Es importante usar un color liso sin más. Ni usar patrones ni diseños que molesten a la vista. Ni colocar bordes alrededor de las barras que no aporten información ninguna.

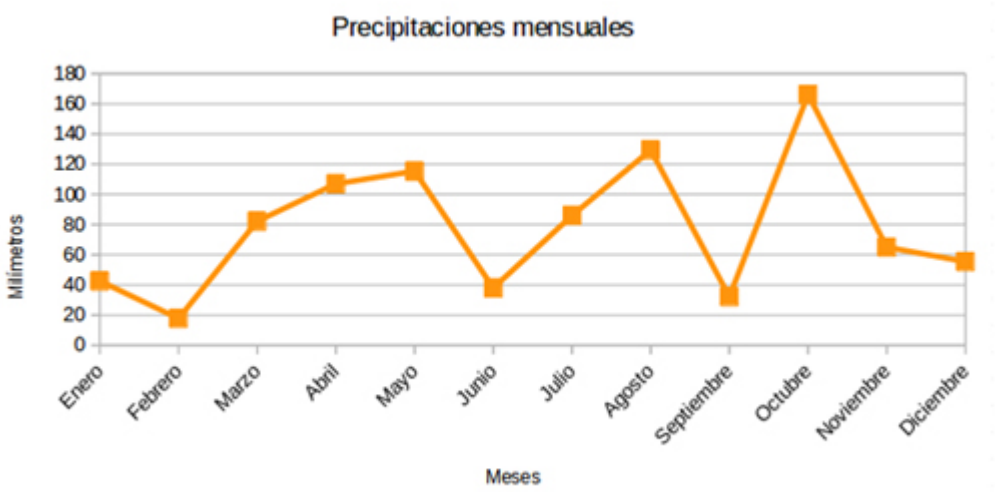
Cuanto más simple sea, más cómodo y sencillo de entender.

En nuestra librería ofreceremos tanto horizontales como verticales, con y sin espacios entre barras. Este será un ejemplo de la implementación en la librería, que anteriormente ya se había mostrado.



4.7 Gráficos de líneas

Los gráficos de líneas son la mejor representación para indicar cambios a lo largo de un eje temporal, que suele ser el eje X.

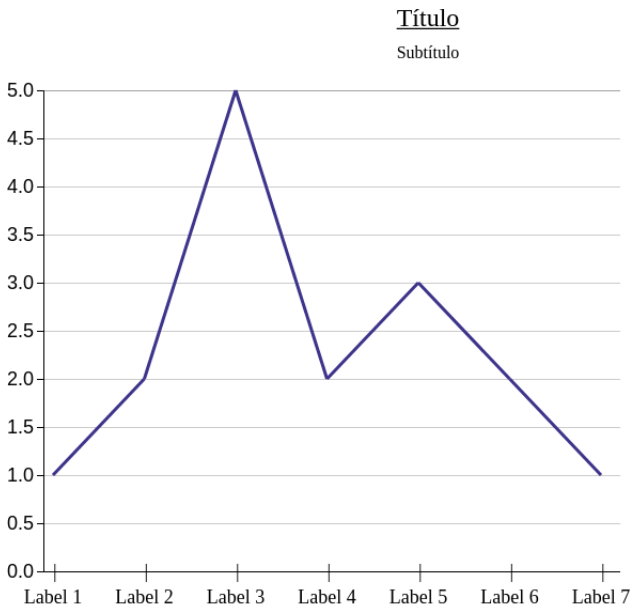


No obstante, existen errores que debemos evitar a toda costa.

Uno de los fallos más comunes es no considerar los gráficos de líneas como puntos unidos por líneas rectas. Se consideraría incorrecto gráficos que, cerca de los puntos, curvan las líneas para que no se generen picos en los puntos.

En esta clase de gráficos, también se puede generar problemas de superposiciones. Si un tramo de la línea se superpusiese a otro tramo y solo mostráramos un color, se podría pensar que: o bien se han superpuesto, o bien en ese tramo de la línea faltan los datos de la línea que queda por debajo. Esto se puede solucionar haciendo la línea de arriba discontinua, que deje ver los colores de abajo, o simplemente indicándolo claramente.

En nuestra librería, se presentará esta representación.



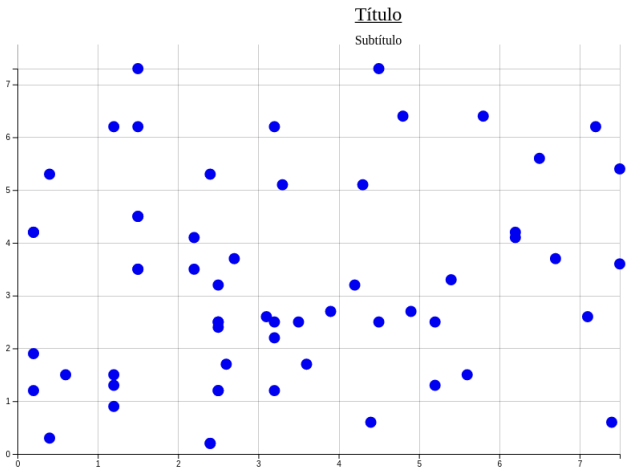
4.8 Gráfico de puntos

Los gráficos de puntos usan figuras para indicar los valores. Son útiles para representar los valores respecto a dos ejes y descubrir si tienen alguna correlación.

En este caso, también podemos encontrar dos tipos distintos de gráficas: si están agrupados o si no lo están. Agrupar puntos en distintos grupos que se representarán con distintos colores o formas ayuda en tareas de comparación.

Al igual que el diagrama de barras, la importancia de los ejes es importante, y en todo momento se debe mostrar con respecto al cero.

Los diagramas de puntos son bastante útiles para mostrar grandes cantidades de datos, pero también hay que tener cuidado con mostrar demasiados. Un gráfico lleno de puntos no aporta nada y solamente confunde.

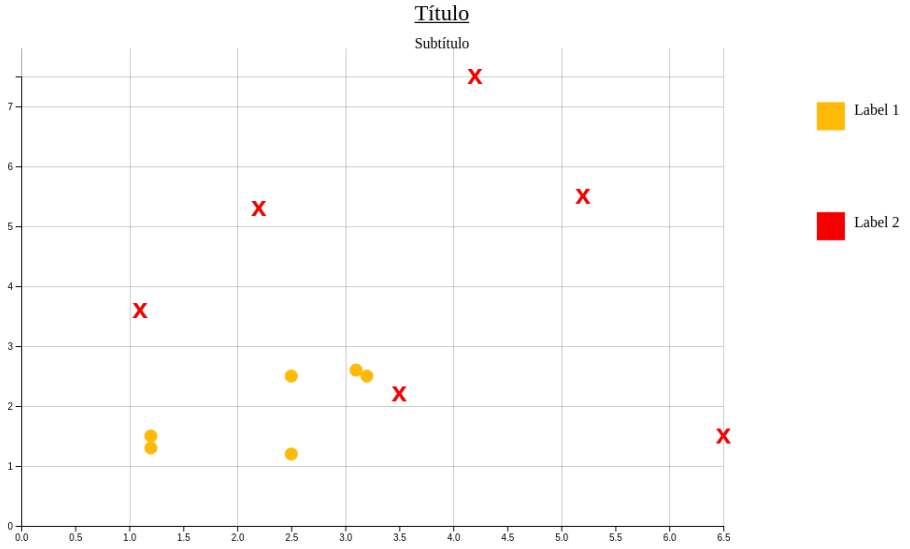


Además, hay que tener cuidado con los puntos que se superponen, pues puede parecer que dos o más puntos parezca solo uno.

En el caso de que los puntos se agrupen, hay que hacer formas y colores que sean claramente diferenciables. Además, al igual que con los no agrupados, hay que tener cuidado con las superposiciones.

Para esto no hay una forma correcta de hacerlo, sino que depende más de la cantidad de datos y su dispersión.

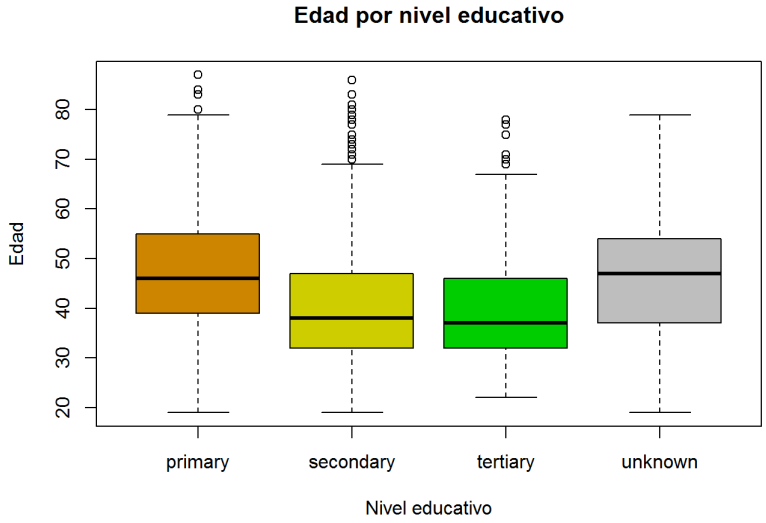
En la librería, se presenta esta representación.



4.9 Diagrama de caja

Posiblemente el más desconocido de todos los demás. Su estructura puede parecer más compleja. No obstante, la idea que transmite es, dado un conjunto de datos, dar el máximo y el mínimo, los cuartiles y los valores atípicos.

Estos diagramas, como se puede ver, proveen gran cantidad de datos en un diagrama reducido.

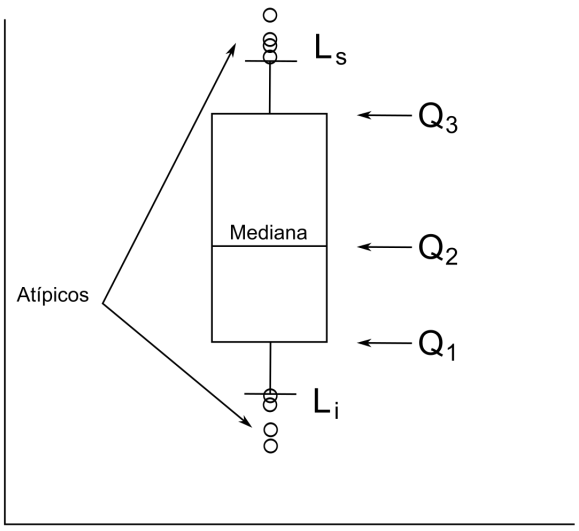


Como vemos en la imagen anterior, estos diagramas se pueden descomponer en tres elementos: círculos, unas líneas y una caja.

Estos círculos son los que corresponden con valores atípicos Las líneas, representan los valores desde los extremos hasta el cuartil 1 y 3, respectivamente.

La caja representa todos los valores entre el cuartil 1 y 3. Además, si se mira más detenidamente, habrá una línea en el centro. Esta línea representará la mediana.

Esta imagen ahora resumirá todo lo comentado.



Debido a su estructura, podemos ver que tiene una gran similitud con el diagrama de barras. Es por ello que la prácticamente todos los fallos que se encuentran en este tipo de diagramas sean similares. No obstante, si tratamos específicamente esta clase de diagramas, debemos remarcar el principalmente las líneas que salen de la caja. Debemos asegurarnos de que estas líneas proveen una buena visualización, sobre todo cuando el diagrama es demasiado pequeño. A fin de cuentas, se trata de una línea muy delgada, por lo que puede perderse fácilmente si no tenemos en cuenta el tema el tamaño del diagrama.

Además, hay que tener cuidado con los valores atípicos. Si se encuentran demasiado alejados de la caja, esto dejará muy poco espacio para este último, haciendo que apenas se aprecien los valores.

4.9.1 Cálculo de los valores

En el caso de que tengamos un conjunto de datos y queramos calcular los valores antes comentados, se pueden realizar fácilmente.

A continuación ponemos un ejemplo de cómo se calcularía en Python:

```
from numpy import percentile

def calculo(values):
    quartilesMedian = percentile(values, [25, 50, 75])
    iqr = quartilesMedian[2] - quartilesMedian[0]
    outliers = [x for x in values \
    if x > quartilesMedian[2] + \
    1.5*iqr or x < quartilesMedian[0] \
    - 1.5*iqr]
    values = [x for x in values \
    if x not in outliers]
    return {
        "q1": quartilesMedian[0],
        "median": quartilesMedian[1],
        "q3": quartilesMedian[2],
        "outliers": outliers,
        "min": min(values),
        "max": max(values)
    }
```

Chapter 5

Estructura de los gráficos en formato JSON

5.1 Introducción

Ahora que ya sabemos como serán visualmente nuestras gráficas, vamos a ver cómo poder daremos nuestra la información que vamos a mostrar a nuestra librería. Para ello, debido a su sencillez, usaremos un formato JSON para transmitir los datos, siguiendo las estructuras que veremos a continuación.

En las imágenes anteriores se han mostrado cómo quedarían las gráficas. No obstante, también ofreceremos parámetros opcionales si se quiere personalizar más. Estos parámetros hacen referencia a qué colores utilizar en la gráfica. Si se desea usar estos parámetros, debemos recordar la gran importancia de usar colores correctos para las gráficas, como comentamos anteriormente.

Hemos definido la estructura de la manera que sea más cómoda al usuario y todos tendrán un patrón común: un título, un subtítulo y un array que contendrá los datos que mostrar, que variará en función del tipo de gráfica a mostrar.

5.2 Gráfico de barras

Este JSON cubre tanto los gráficos de líneas horizontales como verticales. Entre sus parámetros, se encuentran los siguientes:

1. "title": Opcional. Hace referencia al título, que se encuentra situado en la parte superior de la gráfica.

2. "subtitle": Opcional. Hace referencia al subtítulo, que se encuentra justo debajo del título. Tiene un tamaño menor que el título.

3. "desc": Opcional. Descripción de la gráfica. No es visible, pero ayuda en algunos lectores de pantalla.

4. "data": Datos que se mostrará en la gráfica. Para cada dato, hay que indicar la etiqueta, el valor y un color (opcional).

5. "color": Opcional. Se incluye dentro de cada elemento del array "data". Se usa para seleccionar el color manualmente.

6. "highlight": Opcional. Como comentamos anteriormente, se puede usar un color diferente al resto para hacerlo destacar. Se proporciona este parámetro para que, en el caso de que no especifiquemos los colores a mano, se pueda destacar algún o algunos valores de una forma más sencilla. Recibe un boolean como parámetro.

```
{
    "title": "Título",
    "subtitle": "Subtítulo",
    "desc": "Descripción",
    "data": [
        {
            "label": "Et1",
            "value": 1,
            "color": "#FFFFFF"
        },{
            "label": "Et2",
            "value": 2,
            "highlight":true
        },{
            "label": "Et3",
            "value": 5
        }
    ]
}
```

5.3 Gráfico de líneas

Entre sus parámetros se encuentran:

1. "title": Opcional. Hace referencia al título, que se encuentra situado en la parte superior de la gráfica.

2. "subtitle": Opcional. Hace referencia al subtítulo, que se encuentra justo debajo del título. Tiene un tamaño menor que el título.

3. "desc": Opcional. Descripción de la gráfica. No es visible, pero ayuda en algunos lectores de pantalla.

4. "labels": Indica el texto que aparece en el eje X en cada valor de los datos a mostrar.

5. "data": Datos que se mostrará en la gráfica. Para cada dato, hay que indicar la etiqueta, un color (opcional) y un conjunto de valores para mostrar.

6. "color": Opcional. Se incluye dentro de cada elemento del array "data". Se usa para seleccionar el color manualmente

```
{
    "title": "Título",
    "subtitle": "Subtítulo",
    "desc": "Descripción",
    "labels": ["Paso 1", "Paso 2", "Paso 3"],
    "data": [
        {
            "label": "Línea 1",
            "values": [1,2,1]
        },{
            "label": "Línea 2",
            "values": [2,2,1]
        },{
            "label": "Línea 3",
            "values": [3,1,2]
        },{
            "label": "Línea 4",
            "values": [1,2,3]
        }
    ]
}
```

5.4 Gráfico de puntos

1. "title": Opcional. Hace referencia al título, que se encuentra situado en la parte superior de la gráfica.

2. "subtitle": Opcional. Hace referencia al subtítulo, que se encuentra justo debajo del título. Tiene un tamaño menor que el título.

3. "desc": Opcional. Descripción de la gráfica. No es visible, pero ayuda en algunos lectores de pantalla.

4. "data": Datos que se mostrará en la gráfica. Para cada conjunto de datos, se debe indicar una etiqueta que nombre a ese conjunto, un color (opcional) y los valores x e y de los mismos.

5. "color": Opcional. Se incluye dentro de cada elemento del array "data". Se usa para seleccionar el color manualmente.

```
{
    "title": "Título",
    "subtitle": "Subtítulo",
    "desc": "Descripción",
    "data": [
        {
            "label": "Etiqueta 1",
            "values": [
                {
                    "x": 1,
                    "y": 1
                }, {
                    "x": 2,
                    "y": 2
                }, {
                    "x": 3,
                    "y": 3
                }
            ],
            "color": "#FE2A3B"
        }, {
            "label": "Etiqueta 2",
            "values": [
                {
                    "x": 1,
                    "y": 1
                }, {
                    "x": 4,
                    "y": 4
                }, {
                    "x": 2,
                    "y": 3
                }
            ]
        }
    ]
}
```

5.5 Gráficos de cajas

Esta estructura se puede usar tanto para gráficos verticales como horizontales.

1. "title": Opcional. Hace referencia al título, que se encuentra situado en la parte superior de la gráfica.

2. "subtitle": Opcional. Hace referencia al subtítulo, que se encuentra justo debajo del título. Tiene un tamaño menor que el título.

3. "desc": Opcional. Descripción de la gráfica. No es visible, pero ayuda en algunos lectores de pantalla.

4. "data": Datos que se mostrará en la gráfica. Para cada valor, se debe indicar una etiqueta que nombre, un color (opcional) y un conjunto de valores que represente la mediana, el primer y tercer cuartil, el máximo y mínimo y los valores atípicos (estos últimos pueden no indicarse si no hubiese).

5. "color": Opcional. Se incluye dentro de cada elemento del array "data". Se usa para seleccionar el color manualmente.

6. "colorBorder": Opcional. Se incluye dentro de cada elemento del array "data". Se usa para seleccionar el color del borde y de la línea de la mediana de la caja manualmente.

7. "highlight": Opcional. Como comentamos anteriormente, se puede usar un color diferente al resto para hacerlo destacar. Se proporciona este parámetro para que, en el caso de que no especifiquemos los colores a mano, se pueda destacar algún o algunos valores de una forma más sencilla. Recibe un boolean como parámetro.

```
{
    "title": "Título",
    "subtitle": "Subtitulo",
    "desc": "Descripción",
    "data": [
        {
            "label": "Etiqueta 1",
            "values": {
                "median": 3,
                "q1": 2,
                "q3": 8,
                "min": 1,
                "max": 10
            }
        }, {
            "label": "Etiqueta 2",
            "values": {
                "median": 5,
                "q1": 4,
                "q3": 6,
                "min": 3,
                "max": 7,
                "outliers": [1,10]
            }
        }, {
            "label": "Etiqueta 3",
            "values": {
                "median": 5,
                "q1": 4,
                "q3": 6,
                "min": 3,
                "max": 7
            }
        }
    ]
}
```

5.6 Gráficos de barras agrupados

Esta estructura se puede usar tanto para gráficos verticales como horizontales.

1. "title": Opcional. Hace referencia al título, que se encuentra situado en la parte superior de la gráfica.

2. "subtitle": Opcional. Hace referencia al subtítulo, que se encuentra justo debajo del título. Tiene un tamaño menor que el título.

3. "desc": Opcional. Descripción de la gráfica. No es visible, pero ayuda en algunos lectores de pantalla.

4. "legend": Estos nombres son los que vendrán indicados en la leyenda del gráfico.

5. "data": Datos que se mostrará en la gráfica. Cada conjunto de datos se indicará con un nombre, que aparecerá en el eje correspondiente y los valores de los datos. Opcionalmente se le puede añadir un conjunto de colores para pintar esos datos.

6. "highlight": Opcional. Como comentamos anteriormente, se puede usar un color diferente al resto para hacerlo destacar de una forma más sencilla. Recibe un boolean como parámetro.

7. "color": Opcional. A diferencia del resto, este no iría dentro de data, sino otro parámetro en la base del JSON. Se indicaría en un array de strings los colores que usarán cada barra del grupo, en el mismo orden.

8. "colorHighlight": Opcional. A diferencia del resto, este no iría dentro de data, sino otro parámetro en la base del JSON. Se indicaría en un array de strings los colores que usarán cada barra del grupo, en el mismo orden, siempre y cuando estos datos tengan el parámetro highlight a true.

```
{
    "title": "Título",
    "subtitle": "Subtítulo",
    "legend": ["Valor 1","Valor 2","Valor 3"],
    "data": [
        {
            "label": "Etiqueta 1",
            "values": [1,2,2]
        }, {
            "label": "Etiqueta 2",
            "values": [2,3,1],
            "highlight": true
        }, {
            "label": "Etiqueta 3",
            "values": [1,5,1]
        }
    ]
}
```

5.7 Gráficos de cajas agrupados

Esta estructura se puede usar tanto para gráficos verticales como horizontales.

1. "title": Opcional. Hace referencia al título, que se encuentra situado en la parte superior de la gráfica.

2. "subtitle": Opcional. Hace referencia al subtítulo, que se encuentra justo debajo del título. Tiene un tamaño menor que el título.

3. "desc": Opcional. Descripción de la gráfica. No es visible, pero ayuda en algunos lectores de pantalla.

4. "legend": Estos nombres son los que vendrán indicados en la leyenda del gráfico.

5. "data": Datos que se mostrará en la gráfica. Cada conjunto de datos se indicará con un nombre, que aparecerá en el eje correspondiente y los valores de los datos, que deben incluir la mediana, los cuartiles 1 y 3, el máximo, el mínimo y valores atípicos si hubiese. Opcionalmente se le puede añadir un conjunto de colores para pintar esos datos.

6. "highlight": Opcional. Como comentamos anteriormente, se puede usar un color diferente al resto para hacerlo destacar de una forma más sencilla. Recibe un boolean como parámetro.

7. "color": Opcional. A diferencia del resto, este no iría dentro de data, sino otro parámetro en la base del JSON. Se indicaría en un array de strings los colores que usarán cada barra del grupo, en el mismo orden.

8. "colorBorder": Opcional. Se indicaría en un array de string los colores de los bordes que usarán cada barra del grupo, en el mismo orden.

9. "colorHighlight": Opcional. Se indicaría en un array de strings los colores que usarán cada barra del grupo, en el mismo orden, siempre y cuando estos datos tengan el parámetro highlight a true.

10. "colorHighlightBorder": Opcional. Se indicaría en un array de strings los colores de los bordes que usarán cada barra del grupo, en el mismo orden, siempre y cuando estos datos tengan el parámetro highlight a true.

```
{
    "title": "Título",
    "subtitle": "Subtítulo",
    "legend": ["Valor 1", "Valor 2", "Valor 3"],
    "data": [
        {
            "label": "Etiqueta 1",
            "values": [
                {
                    "median": 5,
                    "q1": 4,
                    "q3": 6,
                    "min": 1,
                    "max": 7
                }, {
                    "median": 5,
                    "q1": 4,
                    "q3": 6,
                    "min": 3,
                    "max": 7,
                    "outliers": [1,10]
                }, {
                    "median": 3.5,
                    "q1": 3,
                    "q3": 4,
                    "min": 2,
                    "max": 9,
                }
            ]
        }, {
            "label": "Etiqueta 2",
            "values": [ --- ]
        }
    ]
}
```

Chapter 6

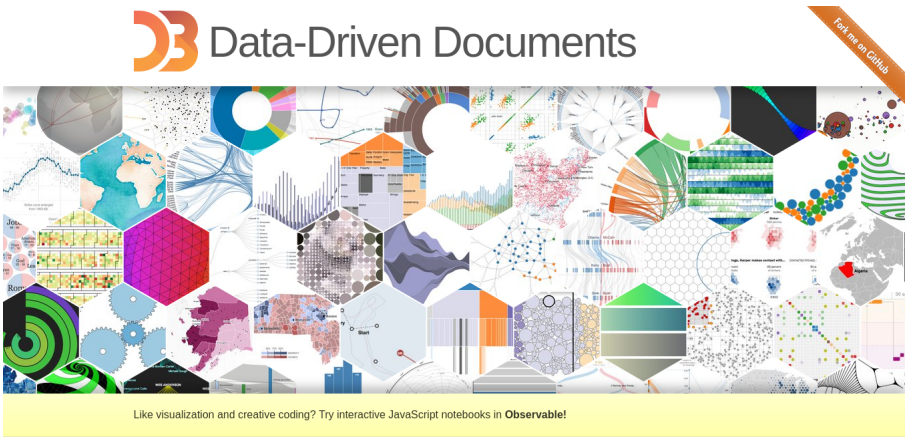
Librería JavaScript

6.1 Introducción

Con las estructuras de datos ya definidas, crearemos una librería que nos permita crear las gráficas en nuestras páginas web. Para ello, usaremos la librería D3Js para crearlo. Esta librería nos permite trabajar con datos a gran velocidad. Además, al ser una librería que se ejecuta en cliente, nos permite desplazar toda la carga de trabajo fuera de los servidores, haciendo que no haya más carga de trabajo al cargar la página web.

Esta librería además nos otorga diferentes funciones con las que ayudarnos a crear las gráficas. Entre estas funciones se incluye la de crear los ejes o la de ajustar los valores en función del ancho en píxeles que dispongamos.

Si vemos en su página principal, podemos ver la gran capacidad de formas que podemos generar, así como el código de las mismas disponible en GitHub, lo que facilita bastante su aprendizaje.



6.2 Uso de la librería

Antes que nada, será necesario que añadamos las librerías a nuestra página. Esto se hará como cualquier otro fichero JavaScript, añadiendo las etiquetas script en nuestro HTML. No obstante, será necesario que previamente a esto añadamos la librería D3Js. Esto se puede hacer mediante el CDN que ellos mismos proporcionan en su página, o descargándolo en nuestro servidor.

Todo método que creemos en nuestra librería seguirá el mismo modo de funcionamiento. En nuestra página web añadiremos un div con alguna propiedad para poder seleccionarlo. Se recomienda encarecidamente que esta propiedad sea un id. Posteriormente, mediante código JavaScript, llamaremos al método de nuestra librería de la gráfica que queramos, le indicaremos la id anteriormente definida, el ancho y alto de la gráfica y el JSON que hemos definido anteriormente. Esto hará que, una vez cargada la página, se añada dentro del div un SVG con el ancho y alto definido. Y este SVG tendrá dentro todas las líneas, figuras y textos de nuestra gráfica.

Otra gran ventaja de la librería D3Js es que como acabamos de mencionar, podemos generar los gráficos en SVG. La gran ventaja de este formato es que, a diferencia de una imagen, se basan en vectores y permiten que sea altamente ampliable sin perder resolución. Aun así, debemos evitar que el usuario tenga que hacer zoom para poder apreciar correctamente la gráfica.

6.3 Funciones creadas

La librería JS creada traerá una función para cada tipo de gráfica. Todas ellas recibirán como parámetro el selector con el mismo formato que en archivos CSS. Es decir, en el caso de las id, se le añade el símbolo # delante. Como segundo y tercer parámetro recibirán el ancho y el alto respectivamente, y como último parámetro recibirán un JSON con una de las estructuras explicadas anteriormente.

Las funciones creadas son:

- function horizontalBarGraph(id, width, height, data);

Esta función nos permite crear un gráfico de barras horizontales, dejando un espacio entre las barras.

- function horizontalBarGraphNoSpace(id, width, height, data);

Esta función nos permite crear un gráfico de barras horizontales pero, a diferencia de la anterior, esta no deja ningún espacio entre las barras.

- function verticalBarGraph(id, width, height, data);

Esta función nos permite crear un gráfico de barras verticales, dejando un espacio entre las barras.

- function verticalBarGraphNoSpace(id, width, height, data);

Esta función nos permite crear un gráfico de barras verticales pero, a diferencia de la anterior, esta no deja ningún espacio entre las barras.

- function dotChart(id, width, height, data);

Esta función nos creará un gráfico de puntos, permitiendo agruparlos en hasta cuatros grupos distintos con diferente forma y color.

- function lineChart(id, width, height, data);

Esta función nos permite crear un gráfico de líneas, permitiéndonos crear distintas líneas de colores distintos.

- function horizontalBoxPlot(id, width, height, data);

Esta función nos permitirá crear gráficos de caja horizontales.

- function verticalBoxPlot(id, width, height, data);

Esta función nos permite crear gráficos de cajas verticales.

- function groupedHorizontalBarChart(id, width, height, data);

Esta función nos permite crear gráficos de barras horizontales, agrupando valores para comparar mejor, incluyendo una leyenda a la derecha.

- function groupedVerticalBarChart(id, width, height, data);

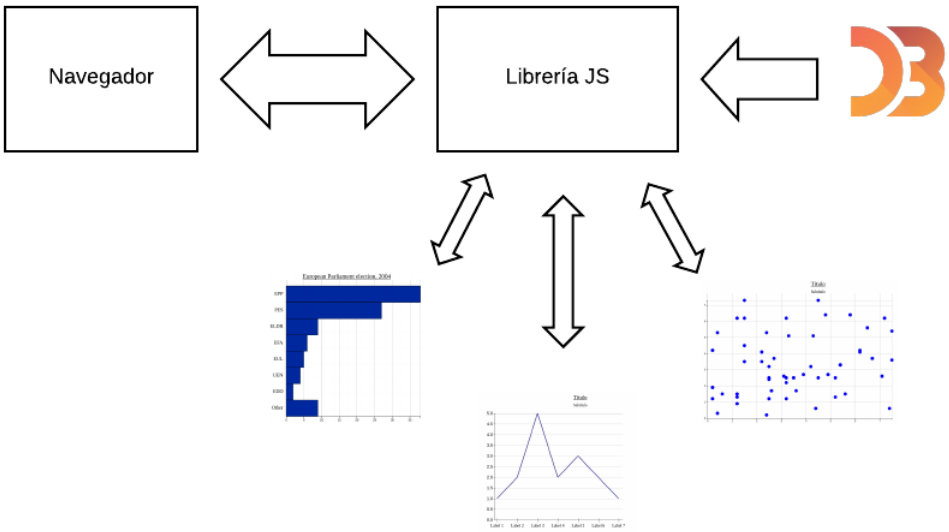
Esta función nos permite crear gráficos de barras verticales, agrupando valores para comparar mejor, incluyendo una leyenda a la derecha.

- function groupedHorizontalBoxPlot(id, width, height, data);

Esta función nos permite crear gráficos de cajas horizontales, pero esta vez, los valores estarán agrupados para poder realizar comparaciones de manera más cómoda.

- function groupedVerticalBoxPlot(id, width, height, data);

Esta función nos permite crear gráficos de cajas verticales, pero esta vez, los valores estarán agrupados para poder realizar comparaciones de manera más cómoda.


Navegador
Librería JS

Chapter 7

Creación de una aplicación Django

7.1 Introducción

Para probar la librería, crearemos una aplicación web donde añadiremos algunos gráficos de los creados anteriormente. Estos gráficos reflejarán los datos sacados de una API que consumiremos. El objetivo de esta aplicación consiste en mostrar información relativa a un repositorio de GitHub, mostrando datos como el número de líneas de código por lenguaje o los commits de los usuarios.

Para crear esta aplicación usaremos el framework Django, debido a la gran facilidad para crear páginas web y la comodidad del lenguaje de programación Python.

La API que usaremos será la propia que proporciona GitHub. Toda la información al respecto se puede encontrar en su página web [4]. Esta API proporciona bastantes datos de un repositorio dado, siempre y cuando sea público. No obstante, debido a limitaciones de la propia API, se limita el número máximo de peticiones por hora. Cuando eso ocurra, se devolverá un error 500 al intentar acceder.

7.2 Funcionamiento

Dado un repositorio, donde únicamente necesitamos el nombre del usuario y el nombre del repositorio, llamaremos a la API. Este repositorio debe ser público para que podamos leer los datos.

En la parte de back-end, únicamente leeremos los datos de la API, obtendremos los datos deseados y formaremos el JSON. Acto seguido, devolver-

emos una template donde ya estará la gráfica preparada a la espera de los datos.

7.3 Llamada a la API

Usando la librería requests de Python, realizaremos llamadas a la API de GitHub. Dicha API se encuentra localizada en:

https://api.github.com

Y podemos obtener datos de los repositorios consultando:

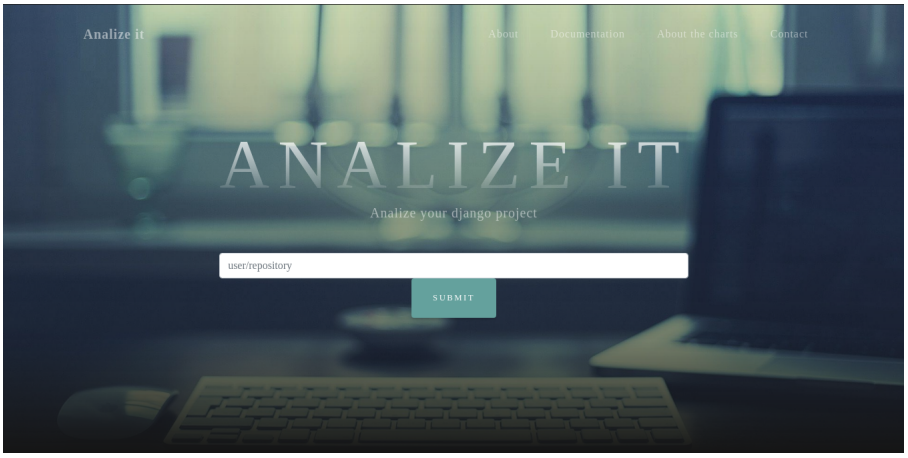
https://api.github.com/repos/*owner*/*repository*

Una vez aquí, podemos ver una gran cantidad de información, así como otros enlaces a los que podemos acceder si queremos buscar más información al respecto. Toda la documentación de la misma viene proporcionada en su página web [4].

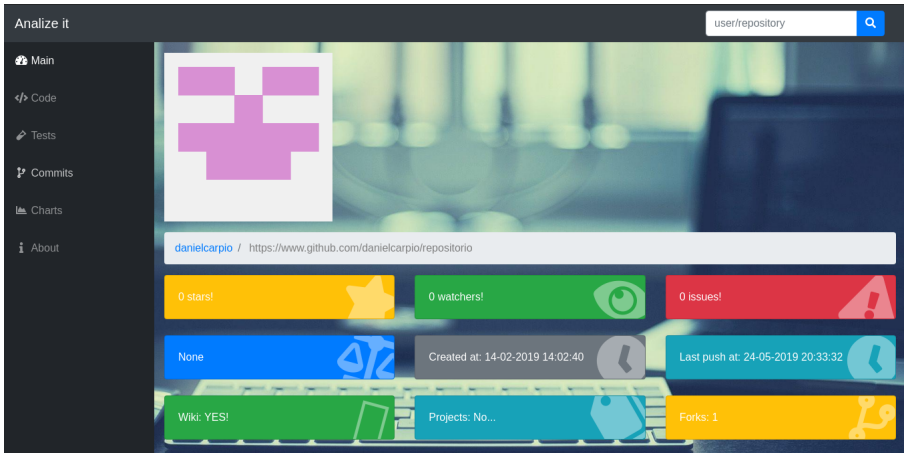
No obstante, hay que destacar que la versión gratuita, solo permite realizar un total de 60 llamadas por hora.

7.4 Partes de la aplicación

Las imágenes mostradas a continuación se han hecho analizando un proyecto software real en el cual se estuvo trabajando anteriormente a este proyecto. Nuestra aplicación empieza con un index que proporciona datos sobre la aplicación, así como un input donde escribir el nombre del repositorio deseado.

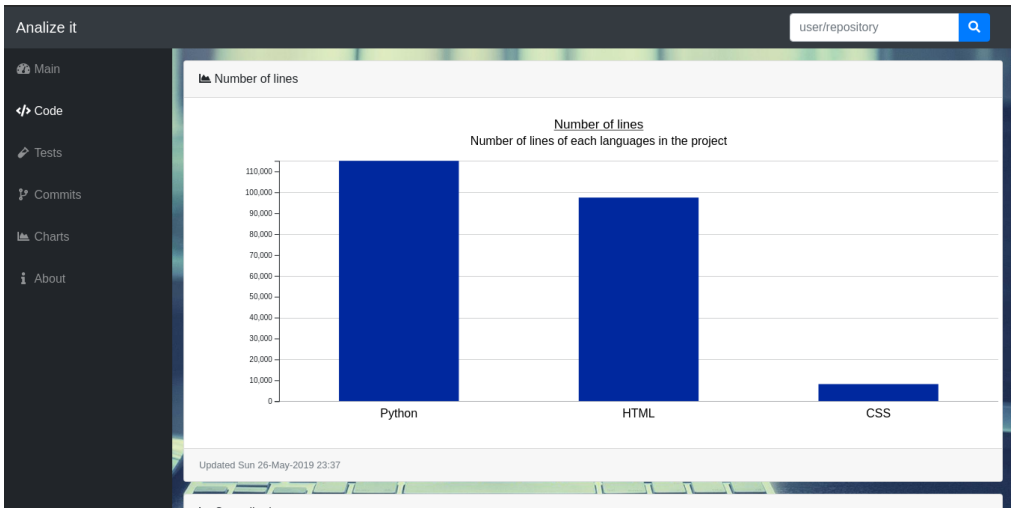


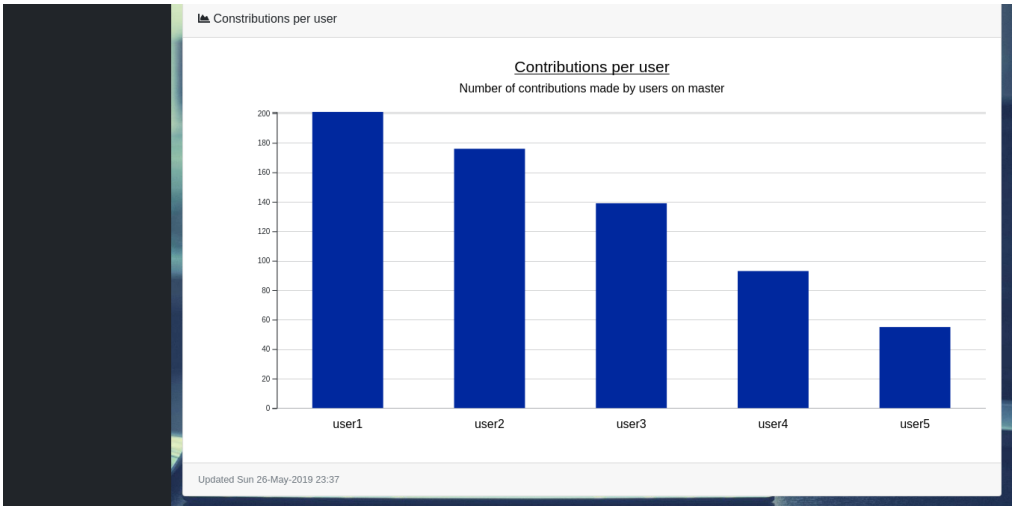
Una vez pongamos un repositorio con un proyecto dado, nos mostrará información sobre el mismo, como la fecha de creación o la licencia que usa.



En la pestaña de la izquierda, podemos ver distintas opciones. Entre ellas, la primera sería Main, que es donde nos encontramos. La segunda sería Code. En esta página, veremos información sobre el código presente en el repositorio.

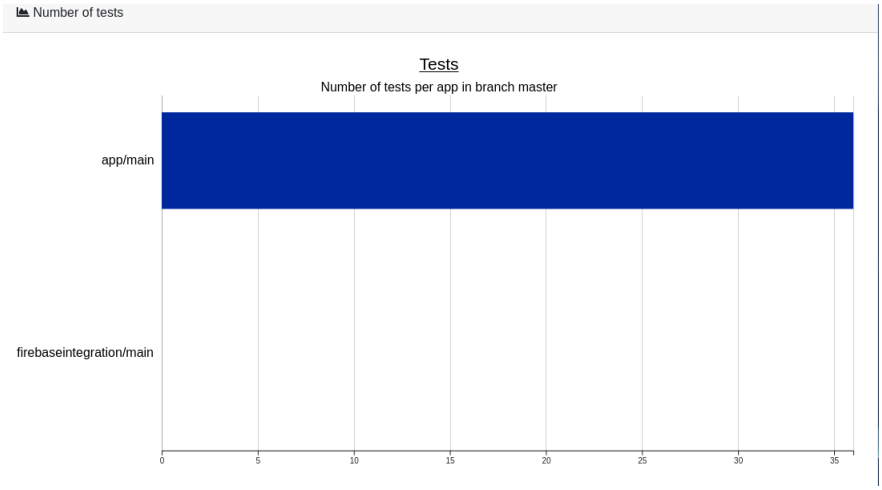
Esta vista tendrá dos gráficas presentes, que representarían el número de líneas de código por lenguaje presente en el mismo y el número de contribuciones hechas por usuario.

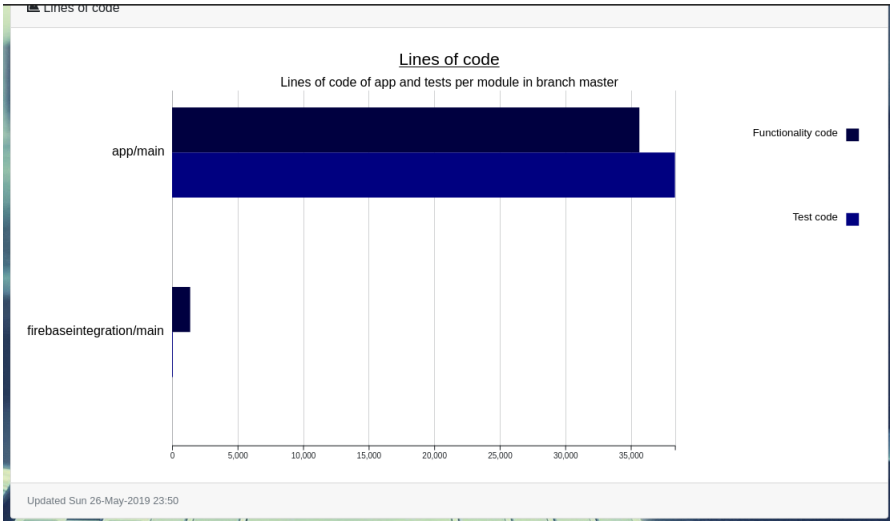




En la tercera pestaña presente en la aplicación es Tests. Esta pestaña únicamente funciona si el repositorio aloja un proyecto Django.

Esta aplicación muestra el número de tests que existen en la aplicación, así como el número de líneas de código funcional como el número de líneas de tests. Además, te muestra el ratio entre ambas para poder apreciar si se considera aceptable. Este último es totalmente subjetivo, pero indicamos con color rojo si el ratio es 1:0, amarillo si es 1:1 o verde si es superior.





Ratio table

App name	Ratio	Functional lines of code	Test lines of code
app/main	1:1	35653	38368
firebaseintegration/main	1:0	1383	60
App name	Ratio	Functional lines of code	Test lines of code

Updated Sun 26-May-2019 23:48

7.5 Instalación

Para lanzar la aplicación en local, instalaremos los paquetes que aparecen en requirements.txt de la siguiente forma:

```
$ pip install -r requirements.txt
```

Una vez instalado, nos desplazamo a la carpeta del proyecto, y ejecutamos los siguientes comandos:

```
$ python manage.py migrate
$ python manage.py runserver
```

Una vez esto, ya podemos acceder a la aplicación accediendo a:

```
https://localhost:8000/
```

Chapter 8

Despliegue en Heroku

8.1 Introducción

Para ayudarnos en el desarrollo de la aplicación, veremos cómo desplegar la aplicación en un servidor. Para esto, usaremos Heroku. Debido a su bajo coste y a que nos proporciona herramientas para facilitar el despliegue de aplicaciones en Django, consideramos que es una buena opción. Además, para subir los proyectos en Heroku, se usa Git, haciendo que sea muy cómodo modificar el contenido del mismo.

El único problema que tenemos con Heroku es que, si el servidor se duerme por inactividad, tarda un poco en arrancar. No obstante, no es más de 10 segundos, así que lo consideramos aceptable para nuestro caso.

8.2 Despliegue

Empezamos enlazando nuestro repositorio con el repositorio que nos ofrece Heroku. Esto lo hacemos con el siguiente comando:

```
heroku git:remote -a "app name"
```

Entre las herramientas que nos ofrece Heroku, tenemos la librería django_heroku. Con esta librería, que se instala con pip, se configurará automáticamente la base de datos PostgreSQL. Únicamente tenemos que añadir a nuestro settings.py:

```
import django_heroku
django_heroku.settings(locals())
```

Debemos especificar qué versión de Python usar. Esto lo haremos creando un fichero runtime.txt con la versión deseada:

```
python-3.7.1
```

También debemos darle las instrucciones para desplegar la aplicación. Para esto, creamos un fichero llamado Procfile y le añadimos las instrucciones:

```
release: sh -c 'cd app && python manage.py migrate'
web: sh -c 'cd app && gunicorn app.wsgi --log-file -'
```

Con esto, ya solo debemos hacer un push al repositorio de Heroku con estos archivos junto a la app y ya se desplegaría.

En nuestro caso, la aplicación se encontraría desplegada en:

```
https://analizeit.herokuapp.com/
```

Chapter 9

Contenedor Docker

9.1 Introducción

A día de hoy, Docker nos ofrece una comodidad a la hora de desarrollar software que no podemos dejar pasar. Debido a la gran comodidad que nos ofrece Docker para desplegar fácilmente aplicaciones, así como la gran popularidad que ha adquirido, su ligereza y velocidad, consideramos de gran utilidad crear una imagen de la aplicación antes creada y subirlo a Docker Hub para evitar todos los pasos anteriores de instalación en local.

Además de conseguir evitar todos estos pasos durante la instalación de nuestra aplicación, conseguimos también evitar tener que instalar las dependencias, pues ya todas vienen previamente instaladas dentro de la imagen, y no es necesario instalarlas en nuestros ordenadores personales si se diese el caso.

9.2 Creación de la imagen

Para conseguir esto, crearemos una imagen usando un Dockerfile. En este archivo, añadiremos las instrucciones que seguirá Docker para crear la imagen.

Para nuestro caso, nuestro Dockerfile es el siguiente:

```
FROM python:alpine
MAINTAINER danielcarpio

RUN apk add postgresql-dev gcc python3-dev musl-dev

WORKDIR /app

ADD . /app
RUN pip install -r requirements.txt

WORKDIR /app/analizeIt

RUN python3 manage.py migrate

CMD ["sh", "-c", "python3 manage.py runserver 0.0.0.0:8000"]
```

Este Dockerfile copiará nuestro proyecto en un contenedor, instalará los requisitos y lanzará nuestro proyecto. Todo esto se lanza sobre el sistema operativo Alpine.

Hecho esto, ya solo habría que subirlo a Docker Hub con el comando push y ya estaría a disposición de todo el mundo.

9.3 Desplegar el proyecto

Esta imagen se encuentra actualemente subida en Docker Hub. Por ello, la instalación será mucho más sencillo que instalarlo a mano. Simplemente tendremos que descargarnos la imagen ejecutando:

```
$ docker pull danielcarpio/analizeit
```

Y una vez descargada, lo ejecutamos lanzando:

```
$ docker run -it danielcarpio/analizeit
```

Con esto, mientras tengamos la terminal abierta, podremos acceder a nuestro proyecto accediendo a:

```
https://localhost:8000
```

Chapter 10

Plugin de WordPress

10.1 Introducción

WordPress se considera el medio más común para crear páginas web a día de hoy. Según W3Techs, WordPress se usa en casi el 35% de todas las páginas web [6]. Muchas empresas y desarrolladores usan WordPress para crear sus páginas de una manera sencilla y rápida. Además WordPress proporciona plantillas y plugins, que añaden más funcionalidades a las páginas web, haciéndolas más cómodas a más personas.

Es por ello que consideramos que sería buena idea adaptar esta librería para que se pueda usar fácilmente en la misma. Para conseguir esto, crearemos un plugin que se puede añadir a cualquier página web y mostrar gráficas en las mismas. Los objetivos que vamos a perseguir en esta parte del proyecto será crear un servidor en WordPress, crear el plugin y probar su uso en el mismo.

10.2 Creación del servidor en WordPress

Para la creación del servidor en WordPress, como nos encontramos en un entorno de desarrollo y queremos agilizar la creación del mismo, volveremos a usar Docker, ya que existen contenedores oficiales que nos ahorran mucho tiempo en la configuración, en especial con la base de datos MySQL.

A diferencia del anterior, esta vez necesitaremos dos imágenes: una para el servidor de WordPress y otra para el servidor MySQL. Para esto, usaremos otra de las herramientas que nos ofrece Docker: crearemos un archivo docker-compose donde podemos establecer la configuración deseada con las imágenes antes comentadas y posteriormente ejecutar Docker con ese archivo. Automáticamente se creará el sistema tantas veces como queramos.

El archivo docker-compose que usaremos será este:

```
version: '3.1'

services:
    wordpress:
        image: wordpress
        restart: always
        ports:
            - 8080:80
        environment:
            WORDPRESS_DB_HOST: db
            WORDPRESS_DB_USER: exampleuser
            WORDPRESS_DB_PASS: examplepass
            WORDPRESS_DB_NAME: exampledb

    db:
        image: mysql:5.7
        restart: always
        environment:
            MYSQL_DATABASE: exampledb
            MYSQL_USER: exampleuser
            MYSQL_PASSWORD: examplepass
            MYSQL_RANDOM_ROOT_PASSWORD: '1'
```

Con esto, únicamente tendremos que usar el comando:

```
$ docker-compose up
```

Y ya se lanzará nuestro servidor WordPress en:

```
https://localhost:8080
```

10.3 Funcionamiento del plugin

Nuestro plugin constará de un único fichero PHP con dos funciones.

La primera ella modifica el contenido del header de nuestra página. Lo que hace es añadir contenido al mismo. En este caso, lo que hará será añadir dos etiquetas scripts, una haciendo referencia al CDN de la librería D3Js y la otra con nuestra librería.

La segunda función modifica el contenido de la entrada que estemos escribiendo. Para esto, mediante una expresión regular, buscaremos en el contenido una línea que tenga una estructura que posteriormente comentaremos,

y la modificaremos por un div con una id vacío y por un script que llame a la función deseada de la librería.

La estructura que buscaríamos es la siguiente:

[−nombre_grafica#ancho#alto#JSON−]

El nombre de la gráfica puede ser:

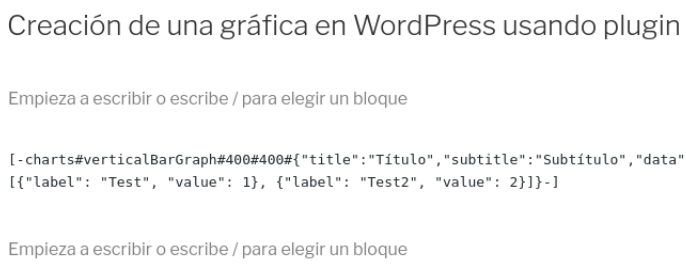
1. horizontalBarGraph
2. verticalBarGraph
3. horizontalBarGraphNoSpace
4. verticalBarGraphNoSpace

que son 4 de las gráficas que ofrecemos en nuestra librería. El ancho, alto y JSON funcionan de la misma manera que hemos estado viendo hasta ahora.

10.4 Gráficas en una entrada del WordPress

Para probar este plugin, en el entorno que hemos creado escribiremos una entrada simple, que solo tendrá el título y la gráfica.

El contenido del mismo, antes de que el plugin realice la modificación es el siguiente:



Cuando estemos en la vista previa o hayamos publicado la entrada, esa línea se modificará por la gráfica deseada con los datos dados:

Chapter 11

Conclusiones

Tras todo este camino que hemos recorrido, hemos podido comprobar la gran importancia de hacer gráficas de manera correcta para poder mostrar los datos correctamente. Hemos podido comparar gráficas con errores, comparándolo con una correcta para poder comprobar el efecto que tienen y cuáles son las recomendaciones a seguir para no cometer esos errores.

A fin de cuentas, podemos apreciar que el mayor error que se comete en la visualización de datos es no detenerse a comprobar qué se ha creado, puesto que parte de los errores comentados se podrían haber visto si se hubiese detenido mirar. Otros errores sí que se nos pueden escapar más a primera vista, pero tras todos estos consejos y recomendaciones, el diseño de las gráficas mejorará notablemente.

No obstante, hay que remarcar que hay muchísimas formas de representar los datos y que a fin de cuentas, es la persona detrás de esto quién se encarga de representar estos datos de la mejor forma posible. Aunque hayamos tratado las gráficas más comunes, existen muchas más, y es trabajo de esta persona elaborar de manera correcta la visualización de las mismas según la situación en la que se encuentre y el objetivo que desee conseguir.

También hemos creado una librería para ayudarnos a crear las gráficas. Esta librería te ayuda en parte a la construcción de las mismas siguiendo las pautas comentadas. Aun así, debemos comentar que existen muchas más en el mercado, y que no son para nada de mala calidad. Como hemos repetido muchas veces en este documento, depende del objetivo, de los datos y sobre todo, de la persona que se encuentre delante del ordenador creando la gráfica.

También hemos puesto en práctica esta librería en una aplicación web. Esta aplicación creada en Django interacciona con GitHub y presenta los datos, ya sean simplemente el número de estrellas que tiene en GitHub así como el número de líneas de código del proyecto, de una forma amigable y cómoda hacia una persona.

Hemos visto tecnologías que nos han sido muy útiles para el desarrollo de software en general. En concreto, Heroku y Docker, dos herramientas muy presentes en el día a día del desarrollo software, ya sea desarrolladores independientes como empresas, y se puede afirmar que han sido tareas bastante beneficiosas en la formación académica de la persona a cargo del desarrollo del proyecto.

Para acabar, hemos podido tratar WordPress, un CMS con una gran presencia en la actualidad, y durante el desarrollo hemos podido tratar desde la creación e instalación del mismo hasta el desarrollo de plugins para añadir más funcionalidad a la misma. Además, para algunas tareas de la misma, hemos reutilizado Docker, consiguiendo ampliar los conocimientos del mismo y ver su funcionamiento no solo para desarrollo de software sino como el despliegue del mismo.

En definitiva, podemos concluir que hemos tratado un tema que al lector de este trabajo le puede ser muy útil en un futuro, y además hemos podido tratar herramientas que generan un gran impacto en la ingeniería informática a día de hoy.

11.1 Ideas futuras

Debido a la duración del trabajo, existen ciertos campos que no hemos podido tratar pero que serían muy provechosos para ampliar la información de este documento. Entre estos trabajos se encuentra ampliar la información al respecto de la visualización de datos, ya que existen muchos más tipos de gráficas y otras formas de representar la información que no son gráficas, como los mapas de calor.

Otro trabajo bastante interesante puede ser la adaptación de las gráficas a dispositivos móviles, puesto que debido a su reducido tamaño, sería útil buscar nuevas formas de representar los datos de una manera efectiva.

Para finalizar, otro trabajo bastante interesante podría ser buscar la manera de saber qué gráfica usar en cada momento. En este trabajo hemos tratado cada gráfica de manera independiente, y no nos hemos parado a pensar qué gráfica usar en cada momento. Sería ideal buscar la manera que poder obtener qué gráfica representaría los datos de una manera mucho más precisa en cada momento.

Bibliography

[1] Stephen Few *Show Me the Numbers: Designing Tables and Graphs to Enlighten*

[2] Fernando José Santana Pineda *Visualización en ingeniería del software. Aplicación a SonarQube*

[3] Tricia Aanderud, Zencos Consulting *How to Become the MacGyver of Data Visualizations*

[4] https://developer.github.com/v3/

[5] https://www.eurocanariasoftalmologica.com/daltonismo-afecta-mas-hombres-mujeres/

[6] https://w3techs.com/technologies/details/cm-wordpress/all/all

[7] https://coolors.co/