UNIVERSIDAD
REY JUAN CARLOS

ESCUELA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Curso Académico 2011/2012

Proyecto de Fin de Carrera

CONSTRUCCIÓN DE GRÁFICOS ESTADÍSTICOS EN PYTHON SOBRE EL ESTÁNDAR SVG

Autor: Isabel Rodríguez Marín

Tutor: Gregorio Robles Martínez

Resumen

Hoy en día los gráficos estadísticos son el método indispensable para representar gráficamente un análisis de datos. El objetivo principal de estas representaciones gráficas es conseguir que un simple análisis visual ofrezca la mayor información posible de la muestra observada.

En este proyecto se ha realizado un programa en lenguaje Python cuyo objeto principal es una nueva implementación de ”Ploticus”, aplicación que genera gráficas estadísticas a partir de ficheros con datos.

El principal atractivo de esta herramienta es la sencillez de uso ya que permite elaborar gráficos en formato SVG *(Scalable Vector Graphics)* [4] a partir de un simple fichero de texto sin formato. Esta tecnología ofrece una notable mejora en el grado de visualización de los gráficos permitiendo al usuario realizar ampliaciones de los diagramas a altos niveles sin perder resolución.

Así mismo se introducen mejoras visuales muy significativas e inexistentes actualmente en ”Ploticus”, como son los filtros y las animaciones (realizadas en Javascript), que ofrecen un mayor dinamismo a los gráficos generados.

En esta aplicación la configuración de los gráficos por parte del usuario es muy alta, permitiendo a éste controlar todo tipo de detalles del mismo: colores, tamaños, estilos, etc.

Una ventaja adicional sobre Ploticus es la posibilidad de editar directamente el fichero SVG resultante (que no deja de ser un fichero de texto en un formato XML) si el usuario no queda satisfecho con los resultados.

Índice general

Capítulo 1

Introducción

La gran motivación de este proyecto viene dada por la necesidad de elaborar un software para la construcción de gráficos estadísticos, intuitivo, de fácil uso pero sobre todo muy flexible, que permita al usuario final representar de una manera cómoda y visual el análisis de una o varias muestras observadas. Actualmente el software más conocido que realiza esta tarea es ”Ploticus”.

Ploticus [16] es una aplicación desarrollada en un entorno Unix que realiza este tipo de gráficas a través de un fichero con datos organizados en campos. Esta basado en un *lenguaje de scripts* a través de los cuales se ejecutan las funciones básicas de estadística. El autor ofrece la posibilidad de producir gráficos sofisticados usando scripts previamente desarrollados llamados ”prefabs”. Estos scripts sin embargo son rápidos pero tienen la desventaja de ser menos flexibles y robustos.

Los nuevos usuarios pueden proporcionar una serie de parámetros de entrada con sus propios datos para generar la gráfica así como modificar o incluso crear sus propios ”prefabs” para personalizar la salida según sus necesidades.

No obstante, esta aplicación tiene grandes limitaciones como por ejemplo la imposibilidad de editar la salida obtenida, resultando inviable modificar las características del gráfico de una manera sencilla, sin necesidad de volver a realizar la ejecución del programa.

Todo esto hace que esta herramienta resulte poco atractiva. El objetivo de este proyecto ha sido precisamente eso, reimplementar esta aplicación creando un software capaz de aportar un amplio abanico de posibilidades para la implementación de cada gráfico estadístico que no limite al usuario final en ningún aspecto.

En todo momento se ha optado por elaborar un software orientado a objetos, pudiendo así ser la base para futuros desarrollos mediante la adición de un mayor número de funcionalidades a los gráficos existentes o incluso mediante la creación de nuevos componentes a partir de los componentes básicos SVG incluidos en el mismo.

En este apartado voy a realizar una breve introducción a la estadística como ciencia centrándonos sobre todo en el análisis descriptivo de los datos y los distintos métodos de visualización existentes según el tipo de muestra observada. Así mismo hablaré de los estándares y herra-

mientas utilizadas tanto para la realización del proyecto como para la publicación del mismo dirigiéndome en este último apartado a la forma en la cual se ha dispuesto el recurso para descargar e instalar nuestro programa.

Finalmente matizaré las principales características del lenguaje de programación en el que ha sido escrito el código fuente así como las nociones básicas del modelo que identifica a un software libre.

1.1. Estadística como ciencia

La Estadística actual es el resultado de la unión de dos disciplinas que evolucionan independientemente hasta concluir en el siglo XIX: el cálculo de probabilidades y la Estadística (o ciencia del Estado, *del latín Estatus)* que estudia la descripción de datos. La integración de ambas líneas da lugar a una ciencia que estudia como obtener conclusiones de la investigación empírica mediante el uso de modelos matemáticos.

La estadística es la principal disciplina puente entre los modelos matemáticos y los fenómenos reales. Esta ciencia proporciona una metodología clave para evaluar y juzgar las discrepancias existentes entre la realidad y la teoría. Es frecuente que por razones técnicas o económicas, no sea posible estudiar todos los elementos de una población y por ello se acude a una muestra representativa.

1.1.1. Estadística descriptiva

La *estadística descriptiva* [5], se dedica a los métodos de recolección, descripción, visualización y resumen de datos originados a partir de los fenómenos de estudio. Los datos pueden ser resumidos numérica o gráficamente. El objetivo de un gráfico es describir simple y fielmente la información contenida en los datos observados *(variables)*, resultando uno de los métodos más interesantes para representar información de una manera clara y eficaz y el recurso ideal para representar los datos de nuestros ficheros de entrada.

Una variable es simplemente una característica que al ser medida en diferentes individuos es susceptible de adoptar diferentes valores. Las *variables cuantitativas* son las que se representarán a través de este software ya que son las que se expresan mediante cantidades numéricas. Estas pueden ser de dos tipos:

- ***Variable discreta:*** Es la variable que presenta separaciones o interrupciones en la escala de valores que puede tomar. Estas separaciones o interrupciones indican la ausencia de valores entre los distintos valores específicos que la variable pueda asumir. *Ejemplo*: El número de hijos (1, 2, 3, 4, 5).

- ***Variable continua:*** Es la variable que puede adquirir cualquier valor dentro de un intervalo especificado de valores. Por ejemplo la masa (2,3 kg, 2,4 kg, 2,5 kg, ...) o la altura

(1,64 m, 1,65 m, 1,66 m, ...), que solamente está limitado por la precisión del aparato medidor, en teoría permiten que siempre exista un valor entre dos variables.

En consecuencia, es la naturaleza de la variable estudiada la que nos sugiere la utilización de una representación gráfica u otra.

1.1.2. Diagrama de barras

Para datos de variables discretas se utiliza el *diagrama de barras*. Este diagrama representa los valores de la variable en el eje de abscisas levantando en cada punto una barra de longitud igual a la frecuencia relativa o cantidad de veces que se repite un determinado valor de la variable.

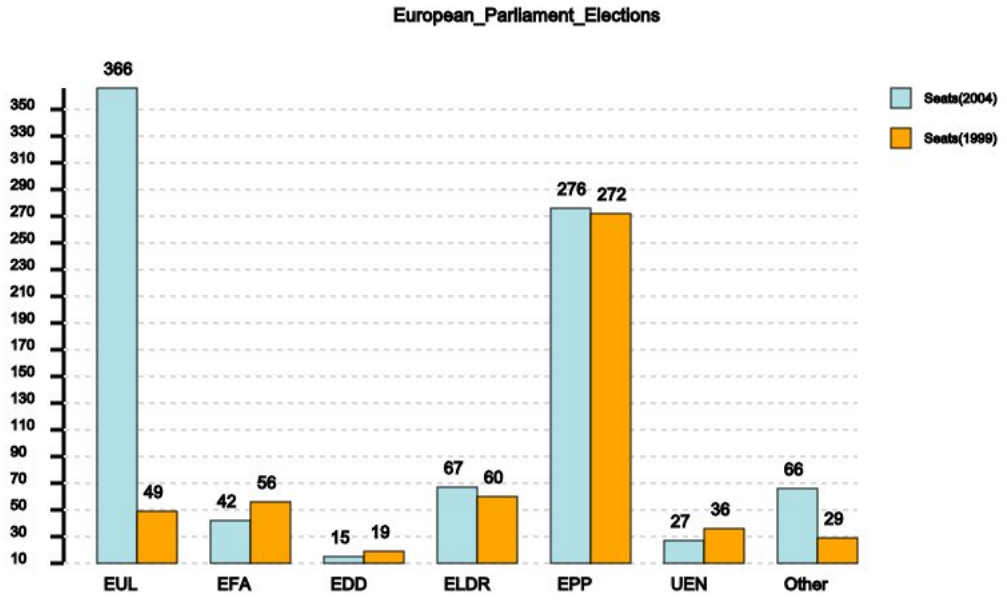

Figura 1.1: Diagrama de barras

1.1.3. Diagrama de sectores

Como alternativa al diagrama de barras podemos representar datos de variables discretas mediante el *diagrama de sectores* también conocido como *gráfico de tarta o pictograma*. Este tipo de diagramas se construye dividiendo el círculo de manera que el área de cada porción sea proporcional a la frecuencia relativa (número de veces que se repite un determinado valor de la variable). Es conveniente que el numero de categorías no sea excesivamente grande para que la imagen proporcionada sea lo suficientemente clara.

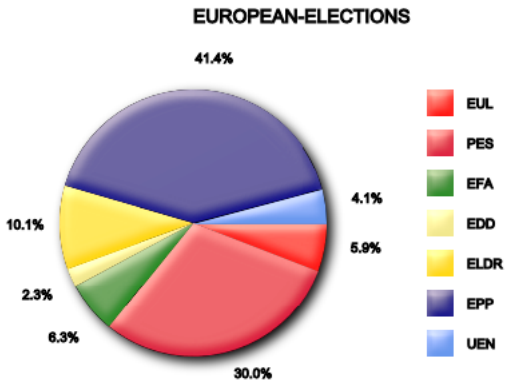

Figura 1.2: Diagrama de sectores

1.1.4. Diagrama de dispersión

La representación gráfica más útil de dos variables continuas sin agrupar es el *diagrama de dispersión,* que se obtiene representando cada observación bidimensional (x_i,y_i) como un punto en el plano cartesiano. Este diagrama es especialmente útil para indicar si existe o no relación entre las variables.

La *correlación* indica la fuerza y la dirección de una relación lineal entre dos variables aleatorias. Se considera que dos variables cuantitativas están correlacionadas cuando los valores de una de ellas varían sistemáticamente con respecto a los valores homónimos de la otra: si tenemos dos variables (A y B) existe correlación si al aumentar los valores de A lo hacen también los de B y viceversa.

Cuando dos variables están relacionadas de forma lineal, los puntos tienden a agruparse en el diagrama de dispersión alrededor de una recta que describe su evolución conjunta. Esta recta puede construirse con el criterio de minimizar su distancia a los puntos observados. Si decidimos medir las distancias en sentido vertical, la recta resultante se denomina *recta de regresión* y tiene la siguiente forma: $h(x) = a + bx$

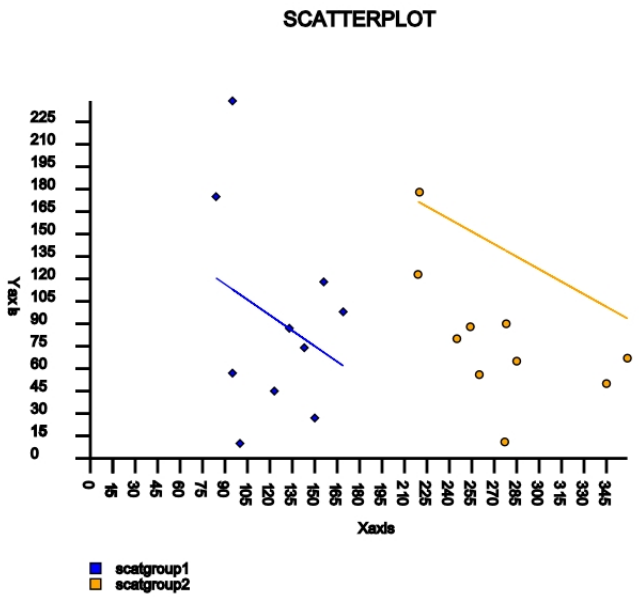

Figura 1.3: Diagrama de dispersión

1.1.5. Diagrama de líneas

Cuando lo que se desea es comparar las observaciones tomadas en dos o más conjuntos de datos a intervalos regulares de tiempo el *gráfico de líneas* es una representación sumamente útil. Es un tipo de gráfico que muestra información mediante una serie de puntos en el plano cartesiano conectados por segmentos de líneas rectas. En este tipo de gráficos temporales el orden de los datos es importante y debe tenerse en cuenta en el análisis.

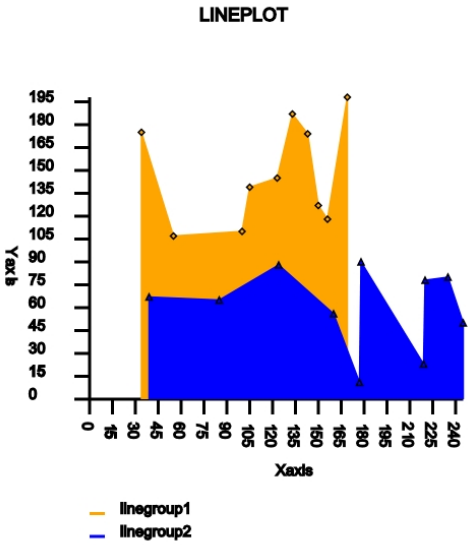

Figura 1.4: Diagrama de líneas

1.2. Gráficos vectoriales

Para entender como están construidos estos gráficos debemos comprender en primer lugar que es y de que se compone un gráfico vectorial [7].

Una *imagen vectorial* es una imagen digital formada por objetos geométricos independientes (líneas, polígonos, curvas, etc.), cada uno de ellos definido por distintos atributos de forma, posición, color, etc.

Una de las grandes ventajas que ofrecen los gráficos vectoriales es la posibilidad de ampliar el tamaño de una imagen a voluntad sin sufrir el efecto de escalado que sufren los *gráficos rasterizados* (también llamados *imágenes matriciales* o *bitmap*), los cuales son un simple fichero de datos que representa una rejilla rectangular de píxeles o puntos de color.

Asimismo, permiten mover, rotar, estirar y retorcer imágenes de manera relativamente sencilla ofreciendo al usuario un modo útil para realizar animaciones sin gran acopio de datos ya que lo que se hace es reubicar las coordenadas de los vectores en nuevos puntos dentro de los ejes cartesianos.

Otra gran ventaja que ofrece esta tipología de gráficos es que los objetos definidos por vectores pueden ser guardados y modificados en el futuro requiriendo menor espacio en disco que un mapa de bits.

Las principales aplicaciones en las que se utiliza esta tecnología son:

- GENERACIÓN DE GRÁFICOS: Se utilizan para crear logos ampliables así como en programas de diseño técnico.
- LENGUAJES DE DESCRIPCIÓN DE DOCUMENTOS: Los gráficos vectoriales permiten describir el aspecto de un documento independientemente de la resolución del dispositivo de salida *(PostScript y PDF)* lo cual permite visualizar e imprimir documentos sin pérdida de resolución.
- TIPOGRAFÍAS: La mayoría de aplicaciones actuales utilizan texto formado por imágenes vectoriales *(p.e. PostScript).*
- VIDEOJUEGOS
- INTERNET: La mayoría de los gráficos vectoriales que se encuentran en el *World Wide Web* suelen ser *formatos abiertos* como SVG.

1.2.1. SVG

El patrón por excelencia para la creación de estos gráficos vectoriales es el estándar abierto conocido como SVG *(Scalable Vector Graphics)* [4]. Este estándar no es más que una especificación para describir gráficos vectoriales bidimensionales, tanto estáticos como animados en formato XML.

El SVG permite tres tipos de objetos gráficos:

- Formas gráficas de vectores *(p.e. caminos construidos a partir de rectas o curvas y áreas limitadas por ellos)*
- Bitmaps o imágenes digitales
- Texto

Los objetos gráficos pueden ser agrupados, transformados y pueden recibir un estilo común. El texto puede estar en cualquier espacio de nombres XML admitido por la aplicación, lo que mejora la posibilidad de búsqueda y la accesibilidad de los gráficos SVG. El juego de características incluye las transformaciones anidadas así como los filtros de efectos.

El dibujado de los SVG puede ser dinámico e interactivo. El Document Object Model (DOM) para SVG, que incluye el DOM XML completo, permite animaciones de gráficos vectoriales sencillas y eficientes mediante un lenguaje de *scripts*. Un juego amplio de manejadores de eventos, como "onMouseOver" y "onClick", pueden ser asignados a cualquier objeto SVG.

1.2.2. XML

Vamos seguidamente a conocer más detalladamente qué características básicas están englobadas en el formato XML.

XML (eXtensible Markup Language) [3] es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). No es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos lenguajes que usan XML para su definición son XHTML y SVG.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande con unas posibilidades mucho mayores como son por ejemplo las *DTD (Definición de Tipo de Documento)* que definen la estructura del documento XML o el *DOM (Modelo de Objetos de Documento)* que sirve para manipular los documentos XML permitiendo acceder de una manera sencilla a la información contenida en los mismos.

Su objetivo prioritario es dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Las principales ventajas que ofrece este lenguaje son una fácil ampliación mediante la adición de nuevas etiquetas y un analizador sintáctico estándar que puede ser utilizado para cada versión de lenguaje XML lo que posibilita el empleo de cualquiera de los analizadores disponibles acelerando así el desarrollo de aplicaciones.

Un aspecto a tener en cuenta es que si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla mejorando así la compatibilidad entre aplicaciones ya que permite interrelacionar aplicaciones de distintas plataformas, sin que importe el origen de los datos. Esto aporta al lenguaje una mayor flexibilidad para estructurar los documentos.

1.2.3. Estándar abierto

Como se ha indicado previamente SVG es un estándar abierto. Un estándar de estas características es una especificación disponible públicamente para lograr un objetivo determinado. Los estándares abiertos tienden a generar un mercado libre y muy dinámico, porque al no haber restricciones en su uso hace que sobre unos estándares abiertos se edifiquen otros y así sucesivamente como actualmente ocurre con los estándares más comunes de Internet. Ser un estándar abierto implica que las licencias de las posibles patentes estén disponibles de forma libre y gratuita.

Un claro ejemplo de estándar abierto es el lenguaje de marcas predominante en la elaboración de páginas web, *HTML/XHTML*, especificación de W3C para el formato de documentos estructurados. W3C *(World Wide Web Consortium)* es una comunidad internacional que desarrolla estándares Web para asegurar el crecimiento largo plazo. Fue creada el 1 de octubre de 1994 por Tim Berners-Lee en el MIT *(Massachusetts Institute of Technology),* actual sede central del consorcio*,* uniéndose posteriormente el ERCIM *(European Research Consortium for Informatics and Mathematics)* y la *Universidad de Kelo,* la mayor y más antigua institución de educación superior en Japón.

En la actualidad la comunidad se encuentra integrada por:

- Miembros del W3C: 330 miembros en abril de 2010.
- Equipo W3C (W3C Team): 65 investigadores y expertos del mundo entero.
- Oficinas W3C (W3C Offices): Repartidas por los cinco continentes.

1.3. Paquetes Debian

Uno de los objetivos principales para integrar un proyecto dentro de la comunidad de software libre es distribuirlo del modo más sencillo y útil posible teniendo en cuenta las necesidades de cualquier tipo de usuario ya sean usuarios inexpertos o desarrolladores profesionales. El sistema de paquetes Debian [12] es una de las mejores aplicaciones de instalación, actualización y desinstalación del software. Gracias a las muchas utilidades que ofrece esta tecnología ha sido posible generar el recurso para la descarga e instalación de nuestro software.

Una de las herramientas más potentes de los paquetes de Debian es la *resolución de dependencias*. Cuando se instala un paquete, automáticamente se verifica en el sistema si se tiene todo lo necesario para poder usar el paquete pedido y se ofrece la posibilidad de resolver cualquier conflicto que pueda surgir o incluso de instalar cualquier paquete que sea necesario adicionalmente. Esta capacidad se encuentra sometida a que las dependencias del paquete estén apropiadamente establecidas.

Un paquete es una colección de archivos con instrucciones acerca de qué hacer con ellos. Normalmente contiene uno o varios programas aunque también puede contener documentación

relacionada. El paquete contiene instrucciones acerca de donde deberían ir los archivos dentro del árbol de directorios, también sobre las librerías o programas que necesita nuestro software para funcionar correctamente así como instrucciones de instalación y scripts de configuración básica.

Los paquetes usualmente contienen software precompilado *(paquetes binarios)* aunque también pueden contener código fuente.

1.4. Epydoc

Epydoc [9] es la herramienta de generación de documentación más potente para los módulos y paquetes de Python con la cual se ha generado la documentación de cada módulo del presente proyecto. Esta basada el análisis de los ”docstrings” de Python. Además de texto plano y de su propio formato, llamado epytext, soporta *reStructuredText* y sintaxis *Javadoc*.

Epydoc proporciona dos formatos de salida HTML y PDF y ofrece dos interfaces de usuario:

- La interfaz de línea de comandos, a la cual se accede a través de un script llamado *epydoc*. El uso a través de línea de comandos se hará de la siguiente manera

```
epydoc [--html|--pdf] [-o DIR] [--parse-only|--introspect-only] [-v|-q]
[--name NAME] [--url URL] [--docformat NAME] [--graph GRAPHTYPE]
[--inheritance STYLE] [--config FILE] OBJECTS...
```

  donde las opciones serán especificadas por el usuario mediante el carácter ”-” o la secuencia de caracteres ”- -” al inicio de la misma (p.e. ”- -html”) y ”OBJECTS” indicará el módulo o módulos que serán analizados.

- La interfaz gráfica, a la cual se accede mediante un script llamado *epydocgui*. En la actualidad, la interfaz gráfica sólo puede generar la salida HTML.

1.4.1. Epytext

Epytext es un lenguaje de marcado ligero que analiza los ”docstrings” de Python y genera una documentación estructurada. Se encuentra dividido en varias categorías:

- ***Estructuras de bloque***: Se codifican con sangría, líneas en blanco o una secuencia de caracteres especiales. Las líneas en blanco se utilizan para separar bloques y la secuencia de caracteres especiales para indicar el comienzo de algunos bloques (*p.e. ”-” se utiliza como un marcador que indica el comienzo de una lista desordenada y ”>>>” se utiliza para marcar bloques de documentación*).

  - ***Bloques básicos***:

- *Párrafos*: Compuestos por una o varias líneas de texto. Deben estar justificados a la izquierda
- *Listas:* Epytext soporta listas ordenadas y desordenadas. Las primeras serán iniciadas con el marcador ”-” y las segundas con un numero seguido de un punto. Los elementos de la lista pueden contener más de un párrafo, y a su vez pueden contener sublistas, bloques literales, y/o bloques de documentación. Las sublistas estarán separadas de su lista predecesora por un sangrado.

- ***Bloques jerárquicos:*** Representan la estructura de animación de la cadena de documentación.
  - *Secciones*: Una sección consta de un encabezado seguido por uno o más bloques secundarios. El encabezado se representa mediante una línea de texto subrayado cuya longitud debe coincidir exactamente con el texto indicado. Los bloques secundarios pueden ser: *subsecciones, párrafos, listas, bloques literales o bloques de documentación*.
- ***Bloques literales*:** Se utilizan para representar bloques con formato ”texto”. Todo se muestra tal y como aparece en el texto de entrada.
- ***Bloques de código****:* Contienen código literal de lenguaje Python. Comienzan con la secuencia especial " >>> ".
- ***Campos***: Se usan para describir propiedades especificas de un objeto*: parámetros (@param), valor de retorno de una función (@return), variables de instancia de una clase (@ivar), autor de un módulo (@author) etc,*. Deben ser situados al final del ”docstring” tras la descripción del objeto

- ***Línea de marcado:*** Son regiones de texto situadas dentro de un bloque básico con distintas propiedades: *propiedades de estilo (B o I o C) hipervinculos (U), expresiones matemáticas (M), términos indexados (X), símbolos (S) y gráficos (G)*. Tienen la forma *”x{...}*” donde x es una letra mayúscula que determina como sera mostrado el texto escrito dentro de las llaves.

1.5. Python

Python [6] es un lenguaje interpretado basado en *scripting* cuyo principal objetivo es proporcionar una sintaxis limpia que aporte legibilidad al código. Es un *lenguaje orientado a objetos* que ofrece apoyo e infraestructura para otros programas más grandes.

Al ser un lenguaje de alto nivel, tiene incluidas las estructuras de datos necesarias para la resolución de un rango de problemas más amplio.

Debido a su sencillez y fácil sintaxis es el lenguaje ideal para el desarrollo rápido de aplicaciones en diversas áreas sobre la mayoría de las plataformas, permitiendo la portabilidad entre ellas con el uso del interprete adecuado.

Pasemos a nombrar algunas de sus características técnicas más relevantes:

- Se trata de un lenguaje multiparadigma que permite múltiples estilos de programación.
- Usa un tipado dinámico que hace que no resulte necesario declarar previamente el tipo de las variables permitiendo que puedan tomar distintos tipos de valores durante la ejecución del programa.
- Controla de forma automática la gestión de memoria reservándola cuando un objeto es creado y liberándola cuando éste no vuelve a ser referenciado.
- Gestiona de forma dinámica la resolución de nombres, enlazando un método y un nombre de variable durante la ejecución del programa.
- Ofrece una modularidad completa ofreciendo la posibilidad de dividir un programa en módulos reutilizables desde otros programas en Python.
- Posee una extensa colección de módulos estándar (librerías) que son la base para realizar la mayoría de las tareas (E/S de ficheros, llamadas al sistema, sockets, etc). Estos módulos pueden ser fácilmente ampliados y/o modificados por otros módulos personalizados implementados en Python o en lenguajes de más bajo nivel como C o C++.
- Permite la gestión de errores mediante el tratamiento de excepciones.

El intérprete de Python estándar incluye un modo interactivo en el cual se escriben las instrucciones en una especie de intérprete de comandos que hace posible que las expresiones puedan ser introducidas una a una, pudiendo verse el resultado de su evaluación inmediatamente, lo que da la posibilidad de probar porciones de código en el dicho modo antes de integrarlo como parte de un programa.

```
>>> 7/-3
3
>>> a = range(10)
>>> print a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

A diferencia de otros lenguajes Python permite escribir programas muy compactos y legibles gracias a estructuras de datos integradas en el propio lenguaje como son las listas, tuplas, diccionarios y conjuntos.

```
>>> list = [66.25, 333, 333, 1, 1234.5]
>>> tuple = 12345, 54321, 'hola!'
>>> set = set(list)
>>> dict = {'mes': 'octubre', 'anyo': 2011}
```

Python está calificado como software libre y fue creado a finales de los ochenta por *Guido van Rossum*. Posee una licencia de código abierto compatible con la Licencia pública general de GNU a partir de la versión 2.1.1.

1.6. Software libre

Hoy día muchos tienden a confundir la definición de ”software libre” con la de ”software gratuito”. Vamos a explicar cada uno de ellos para entender a lo que se refiere cada tipología.

La denominación de *software libre* o *”free software”* [8] se aplica a aquél que ofrece libertad a los usuarios que lo adquieren para usarlo, copiarlo, estudiarlo, modificarlo e incluso redistribuirlo libremente pero esto no quiere decir que sea totalmente gratuito. El software libre puede estar disponible de forma gratuita o al precio que cuesta la distribución en los distintos medios. Un software puede conservar su carácter de ”libre” pero puede ser distribuido comercialmente.

De la misma forma un *software gratuito* o *”freeware”* puede incluso contener código fuente no siendo libre por no estar garantizados los derechos de modificación y distribución de dicho código.

Por otro lado tampoco se debe llegar a confusión con el ”software de dominio publico” ya que a diferencia del software libre éste no requiere estar bajo ningún tipo de licencia puesto que cualquiera puede hacer uso de el siempre con fines legales y manteniendo su autoría original.

El software libre por tanto es el que garantiza las siguientes cuatro libertades:

- *(Libertad 0)*: Libertad para usar el programa con cualquier propósito
- *(Libertad 1)*: Libertad para estudiar como funciona el programa y para modificarlo adaptándolo a nuestras necesidades.
- *(Libertad 2)*: Libertad para distribuir copias y así ayudar al resto de nuestros compañeros.
- *(Libertad 3)*: Libertad para realizar y hacer publicas mejoras en el programa de manera que se beneficie toda la comunidad. Una precondición para esto es la disponibilidad del código fuente.

Una de las licencias más utilizadas en el software libre es la *Licencia Pública General de GNU (GNU GPL)*. En ella el autor conserva los derechos de autor (*Copyright*), permitiendo la redistribución y modificación bajo términos diseñados asegurándose así que todas las versiones posteriores del software permanecen bajo los términos más restrictivos de la misma.

Entre las muchas ventajas que ofrece este tipo de software se encuentran las siguientes:

- Coste bajo de adquisición
- Innovación tecnológica, pues cada usuario aporta sus conocimientos a la vez que su experiencia colaborando así en la evolución y el desarrollo del software.
- Resulta independiente del proveedor al tener disponibilidad de acceso al código fuente.
- Mejora del producto así como una gestión de errores rápida y eficaz al ser de escrutinio publico.
- Adaptación del software a las necesidades de cada usuario.
- Multilenguaje.

Capítulo 2

Objetivos

2.1. Descripción del problema

En este proyecto se pretende la creación de una nueva y mejorada implementación de ”Ploticus” [16], aplicación que genera gráficas estadísticas a partir de un fichero con datos. El objetivo principal es solventar las limitaciones existentes en dicha herramienta mediante la utilización de un modelo mucho más flexible, como es el caso del estándar SVG. Estas limitaciones son básicamente las siguientes:

- Fichero de entrada con un formato preestablecido.
- Salida estática, no permitiendo elementos animados.
- Formato de salida no editable, lo que imposibilita al usuario la modificación manual de los archivos para realizar cambios en las características del diagrama.
- Software cerrado

Este software ha sido desarrollado en lenguaje Python. Está enfocado a ser una exitosa y robusta herramienta para qué tanto usuarios inexpertos como desarrolladores avanzados dispongan de una solución útil que les permita realizar gráficos estadísticos de una manera sencilla a la vez que efectiva.

Una posible representación de la arquitectura del software podría ser la siguiente:

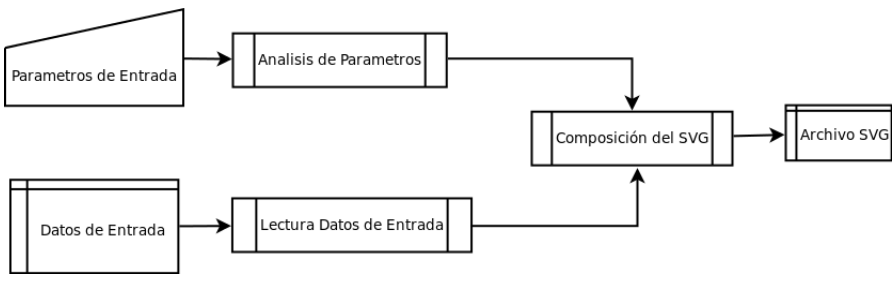

Figura 2.1: Diagrama de arquitectura

Dándose así la siguiente secuencia de uso:

1. El usuario introducirá los datos de entrada y los parámetros necesarios a través de la línea de comandos para indicar todas las especificaciones de cada tipología de gráfico.
2. De forma paralela se realizará la lectura del fichero y el análisis de las opciones elegidas para determinar los elementos SVG necesarios para realizar la composición del diagrama seleccionado.
3. Finalmente se realizará la escritura del fichero SVG resultante.

2.2. Requisitos

De forma general el proyecto estará orientado a cumplir ciertos requisitos básicos:

- Será un software de uso sencillo, abierto y fácilmente ampliable para futuros desarrollos.
- Contará con una buena distribución a través de los distintos medios.
- Tendrá fácil acceso a los recursos disponibles en las distintas distribuciones, de una manera totalmente libre y gratuita

Así mismo y de forma obligatoria deberán cumplirse los siguientes requisitos específicos:

- Deberá ser un software fuertemente orientado a objetos.
- La inclusión de información en el gráfico se realizará a través de un fichero de texto sin formato permitiendo al usuario introducir y/o modificar los datos de forma manual mediante un simple editor de textos, independientemente de la plataforma en la que se encuentre trabajando.
- El diseño será elaborado mediante elementos básicos del estándar SVG *(círculos, rectángulos, polígonos, caminos, etc)* [1] que permitirán al usuario generar cualquier tipo de gráfica a través de la composición de todos ellos e incluso posibilitará la creación futuros desarrollos mediante la adición de nuevas funcionalidades de soporte para un mayor numero de diagramas.
- Proporcionará un estilo dinámico a los diagramas resultantes mediante la adición de elementos estilísticos como filtros y animaciones.
- El formato de salida será implementado en un estándar totalmente editable *(Scalable Vector Graphics)* permitiendo al usuario efectuar de una manera sumamente sencilla cualquier modificación, en caso de que la salida obtenida no resulte satisfactoria.

- Deberá encontrarse bajo una licencia robusta [15] que garantice la libre distribución, modificación y uso de software.
- Deberá contar con un portal de acceso a los recursos disponibles.

2.3. Modelo de desarrollo

Todo proceso de creación de un software incluye una metodología de desarrollo especifica que indica el orden en el cual se han de realizar las distintas tareas para llegar a la consecución del objetivo final. Así mismo esta metodología proporciona los requisitos de entrada y salida para cada una de las etapas del proceso. Estos requisitos son los que describen el comportamiento esperado en el software una vez este desarrollado.

Para este proyecto en particular se ha aplicado el modelo de desarrollo incremental ya que es el que más se adecua a las necesidades del software. Como su propio nombre indica este modelo desarrolla las etapas de manera creciente, por incrementos, permitiendo ampliar el desarrollo de la etapa anterior para hacer evolucionar el producto hasta llegar a la versión definitiva. En cada iteración se analizan los nuevos requisitos y se realizan los cambios en el diseño agregándose si es necesario nuevas funcionalidades a las ya existentes. El objetivo de cada iteración es es ser simple, directa y modular. Una vez la iteración es evaluada y se comprueba el correcto funcionamiento de la misma *(alcanza objetivos)* se procede a la siguiente de forma sucesiva hasta llegar al producto final.

Se ha optado por este modelo de desarrollo debido a que es este software sigue un diseño muy estructurado de manera que cada gráfico es independiente del resto y puede ser considerado como una iteración, con sus propios requisitos y su propio diseño independiente al resto, a la que se aplican sucesivas mejoras hasta conseguir las funcionalidades requeridas por el usuario.

La aplicación de este modelo establece las siguientes etapas:

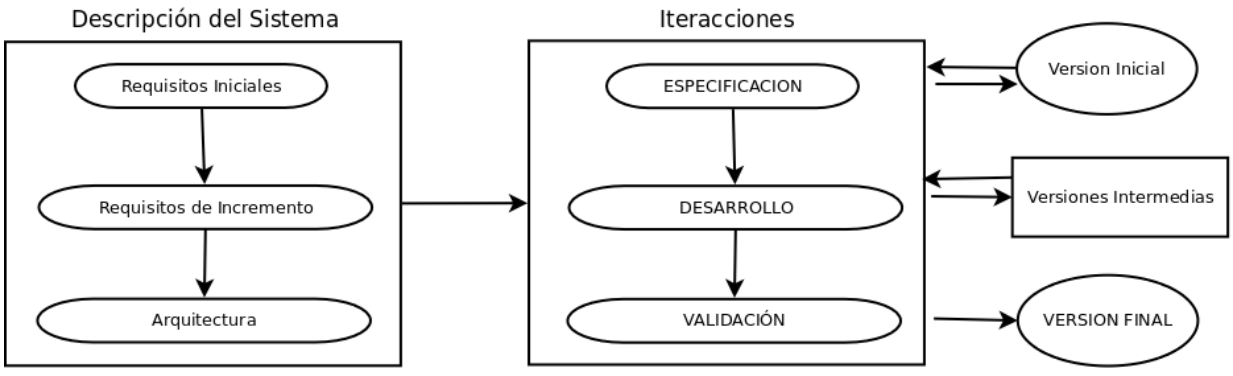

Figura 2.2: Diagrama de desarrollo evolutivo incremental

- ***Descripción del sistema***

- *Requisitos iniciales:* Definidos en la sección 2.2, son los requisitos establecidos por el cliente en la toma de contacto inicial, en nuestro caso se asemeja a la primera reunión mantenida con el tutor del proyecto en la que se definen las funcionalidades con las que debe contar este software.
- *Asignación de requisitos de incremento:* En esta fase se determinarán en base a los requisitos establecidos cuantas iteraciones han de realizarse para llegar al producto final y que funcionalidades tendrá cada una de ellas.
- *Arquitectura:* En esta fase definiremos el diseño interna del programa, elementos (módulos y clases) de los que consta y como interactúan entre ellos.

- ***Iteraciones (versiones intermedias):*** Cada iteración tendrá tres fases:
  - *Especificación* de requisitos
  - *Desarrollo:* Esta será la fase en la que se realizara el diseño y la implementación de cada incremento.
  - *Validación:* Una vez implementado el código se probará de forma independiente cada iteración siendo integrada posteriormente con el resto del sistema.
- ***Versión final:*** Una vez han concluido satisfactoriamente las iteraciones obtenemos el producto final.

El enfoque incremental resulta muy útil cuando nuestro proyecto posee áreas bien definidas capaces de ser desarrolladas con suficiente independencia como para ser diseñadas en etapas sucesivas permitiendo así la entrega de versiones parciales al cliente sin tener que esperar hasta la finalización del proyecto.

Capítulo 3

Diseño e implementación

3.1. Descripción del sistema

Teniendo en cuenta el análisis de los requisitos iniciales es hora de diseñar la implementación del software en base al modelo de desarrollo incremental.

3.1.1. Asignación de requisitos a cada iteración

Lo primero que debemos distinguir son las distintas funcionalidades que deben establecerse en este programa.

Una primera funcionalidad, que puede ser considerada la iteración base del proyecto puesto que estará presente en todas las demás, debe encargarse de la lectura y escritura de los datos en los formatos establecidos en los requisitos iniciales. En esta iteración se crearán las funciones necesarias tanto para la lectura de datos desde el fichero origen como para la escritura de la salida resultante a un archivo con formato SVG que representará la figura del diagrama y proporcionara una interfaz de pruebas muy útil para el desarrollador. Esta primera iteración puede darse en un módulo independiente que se encargue de realizar todas las operaciones de tratamiento de ficheros independientemente del formato del mismo, permitiendo así incluir en desarrollos futuros más funciones de manejo de ficheros en caso de ser necesaria alguna más a petición expresa del cliente.

La siguiente funcionalidad que debemos establecer es la creación de los diagramas que serán las versiones intermedias del proyecto y por tanto iteraciones independientes entre si. Asignaremos de este modo cada nuevo diagrama creado a una iteración diferente. Así, siguiendo el modelo de desarrollo, cada gráfica tendrá su fase de requisitos, diseño y pruebas de forma totalmente ajena al resto. Se establecerán por tanto cinco iteraciones correspondientes a los cinco gráficos que se pretenden generar: *diagrama de barras en dos dimensiones, diagrama de barras en tres dimensiones, diagrama de sectores, diagrama de dispersión y diagrama de líneas.* Cada una de estas iteraciones usará la iteración inicial para obtener los datos de entrada y presentar la gráfica resultante en el archivo de salida lo que resulta útil ya que nos va a

permitir establecer prototipos semejantes a la funcionalidad final. Además nos permitirá realizar incrementos de los prototipos iniciales en base a mejoras establecidas por el cliente.

Las funcionalidades finales servirán para añadir mejoras estilísticas a los diagramas: efectos visuales (de luz, profundidad, etc.) o incluso animaciones para resaltar determinadas características de cada gráfica.

Por último pero no menos importante la ultima fase del proyecto será la encargada de dar a conocer nuestro software mediante un sitio web donde el usuario dispondrá de múltiples recursos para aprender a utilizar y probar esta innovadora herramienta.

En el siguiente esquema se muestra la arquitectura de las iteraciones que van a ser desarrolladas en este proyecto:

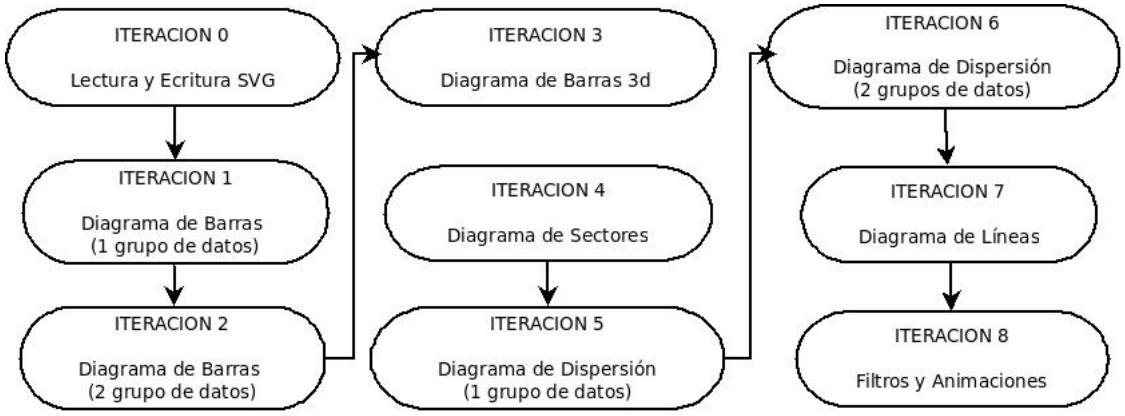

Figura 3.1: Diagrama de iteraciones

3.1.2. Arquitectura

El principal objetivo que se plantea a la hora de diseñar la arquitectura de este proyecto es sencillez así como un código jerárquicamente estructurado de manera que los objetos básicos del estándar SVG sean nuestras clases base y aquellos formados por la composición de varios de estos elementos sean las clases derivadas. Así se podrán crear diagramas de forma fácil mediante la descomposición de cada una de sus partes.

Se establecerán tres módulos principales:

- ***Filetext:*** Un módulo responsable del tratamiento de los ficheros de entrada y salida.
- ***Svgelements***: Un módulo donde quedarán englobados todos los elementos necesarios para la composición de cada tipología de gráfico.
- ***PySVG:*** Finalmente, el módulo principal que contendrá los programas necesarios para el acceso a los parámetros de entrada siendo también el encargado de componer el gráfico SVG mediante la conjunción de los valores y los parámetros de entrada.

Comencemos por especificar como se hará el diseño del módulo encargado del tratamiento de los datos de entrada y salida *"filetext"*. Este módulo tendrá dos partes claramente diferenciadas, la escritura y la lectura de los datos:

- La etapa de lectura de datos se hará mediante un simple método que ira leyendo los datos del fichero de entrada estructurados en dos o más columnas. El valor representado por cada columna será indicado a través de los parámetros de entrada. Por tanto será el propio usuario el encargado de asignar el orden en el que se muestran los datos incluidos en el fichero.

- En cuanto a la escritura de datos se realizará a través de una función que recogerá el documento SVG completo que representa el gráfico y lo transportará a un fichero para su almacenamiento definitivo. Este código vendrá dado por una cadena de texto que proporcionará el método constructor del gráfico. Cada tipología de gráfico tendrá su propio constructor del SVG denominado *"printsvg*". Este constructor se encontrará ubicado en el módulo *"svgelements"*.

El siguiente paso es realizar el diseño de cada gráfico. Como ya se ha indicado anteriormente la idea básica es realizar cada diagrama por descomposición de cada una de sus partes. Por ejemplo un diagrama de barras estará compuesto por un elemento línea vertical, que será el eje de ordenadas, un elemento columna, que será cada barra del diagrama junto con su texto en el eje de abscisas más un numero en su parte superior con el valor correspondiente a su altura si el usuario desea que se muestre, una serie de líneas horizontales junto con un texto que serán los indicadores de los distintos valores en el eje de ordenadas y finalmente uno o varios elementos exteriores al gráfico como son el titulo, (un texto) o la leyenda, (un cuadrado más un texto).

Una buena forma de construir este diseño es realizar una librería de elementos SVG. El módulo encargado de integrar y actualizar dicha librería a medida que se vayan generando gráficos en las distintas iteraciones será el módulo *"svgelements"*. Este módulo contendrá tanto las clases con las formas básicas en SVG (cuadrado, círculo, polígono, camino, etc) como elementos derivados de ellas (columnas, líneas con texto, etc) asimismo contendrá los elementos estilísticos que serán utilizados en posteriores iteraciones para realizar las mejoras decorativas. Veamos un ejemplo de la arquitectura que va a tener el diagrama de barras:

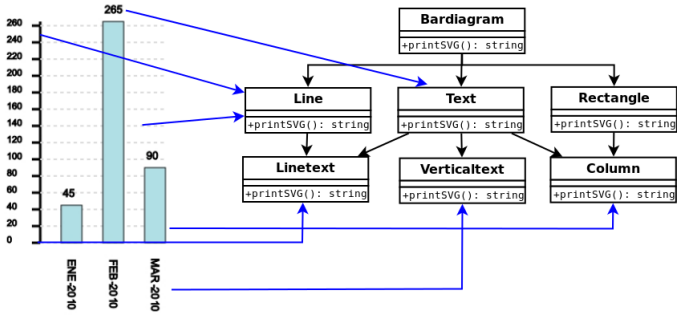

Figura 3.2: Arquitectura básica del diagrama de barras

Finalmente hay que matizar la forma en la que van a ser introducidas las opciones de entrada que serán las encargadas de seleccionar el tipo de gráfico y personalizar el diseño del mismo cambiando estilos, tamaños, colores, etc. También se encargarán de añadir elementos descriptivos externos en caso de ser necesarios. Mediante la adición de elementos estilísticos como animaciones o filtros y componentes exteriores de ampliación de información como títulos, etiquetas o leyendas el usuario final podrá diseñar un gráfico totalmente adaptado a sus necesidades.

El módulo principal *"pysvg"* sera el encargado de leer estos argumentos y llamar al constructor del gráfico con las opciones de diseño especificadas. A continuación se muestra un diagrama en el que se puede apreciar como va a resultar la arquitectura final del diseño atendiendo a estos requisitos.

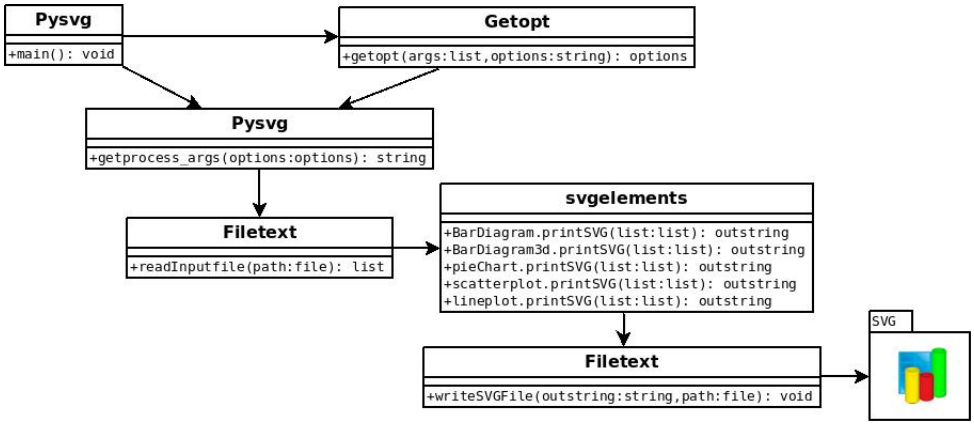

Figura 3.3: Arquitectura final del diseño

Lo único que nos queda por comentar brevemente es el formato del fichero de entrada. Como ya se ha indicado previamente va a ser un simple fichero de texto cuyos datos estarán estructurados en columnas. Cada columna corresponderá a un grupo de datos que será identificado a través de línea de comandos. Por ejemplo para el diagrama de dispersión necesitaremos dos o cuatro columnas dependiendo de cuantos grupos de puntos se deseen mostrar. En caso de

mostrar dos grupos de puntos se deberán indicar las correspondientes coordenadas cartesianas (x_1,y_1) y (x_2,y_2) cada una en una columna diferente.

3.2. Iteración 0

3.2.1. Requisitos

Esta sección no tendrá grandes requisitos de funcionamiento. Únicamente debe ser capaz de reconocer datos desde un fichero de texto y generar un fichero de salida con el código SVG final.

3.2.2. Desarrollo

Para desarrollar esta fase invocaremos al módulo *"filetext.py"*. Este módulo como ya se ha indicado anteriormente contendrá todas las operaciones de tratamiento de ficheros para no limitar la adición de nuevas funcionalidades en posteriores desarrollos de este software. Las funcionalidades base que se darán en este proyecto serán exclusivamente la lectura y escritura de datos en fichero.

La función de lectura tendrá la siguiente cabecera:

```
def readInputfile (path):
```

Esta función será la encargada de abrir y recorrer el fichero de entrada asignando cada dato a una posición determinada dentro de una lista llamada "finalcad" de manera que una vez se hayan leído todas las columnas del fichero:

```
valorA      numero1
valorB      numero2
valorC      numero3
```

quede algo similar a la instrucción siguiente:

```
finalcad=[['valor1','valor2','valor3'],['numero1','numero2',numero3']]
```

Ésta será la lista que utilizaremos en los métodos constructores de cada uno de los diagramas en las posteriores iteraciones. Se ha decidido optar por esta estructura de datos debido a que resulta una estructura muy flexible de acceso aleatorio que ofrece múltiples métodos de tratamiento y permite contener elementos de diferentes tipos. Comúnmente están consideradas como *"el caballo de tiro de Python"*.

Uno de estos métodos es la función "split".

```
str.split([separador])
```

Esta función permite dividir cualquier cadena de texto en base a una determinada separación, un símbolo o cualquier espacio en blanco entre palabras (valor por defecto). Resulta

muy útil para dividir y reubicar cada elemento columna en la lista definitiva que será usada posteriormente.

Sin embargo este método no puede ser utilizado de forma aislada, necesita un método previo que almacene los datos del fichero en una cadena de texto, ese método sera conocido como ”readline”.

```
file.readline([size])
```

La función readline pertenece al módulo ”os”, el cual será importado al inicio, en la fase de importación de módulos.

```
import os
```

Este método lee una línea completa del archivo de entrada y la almacena en una cadena de texto, separaciones incluidas, devolviendo en un parámetro interno el tamaño de la misma. Finaliza cuando lee una cadena vacía retornando el valor interno *EOF (end of file)* que será la señal de fin de fichero.

Otro de los métodos que emplearemos para construir la lista definitiva será ”append”. Este método añade un único elemento al final de la lista y será utilizado para construir la lista final insertando los elementos una vez reorganizados en el orden correcto a medida que van siendo divididos.

Pasemos ahora a hablar acerca de la funcionalidad de escritura. En esta, intervendrán dos factores sumamente relevantes. Por un lado el método encargado de representar el código SVG final en una cadena de texto y el método de encargado de traspasar dicha cadena a un fichero.

Para construir la cadena de texto resultante *(outstring)* se creará un nuevo módulo llamado *”svgelements.py”* que como ya se ha indicado previamente contendrá todas las operaciones necesarias para construir el SVG de cada tipología de gráfico.

En este módulo se incluirán los constructores de cada diagrama, cada uno de los cuales definirá los parámetros necesarios para la fabricación del mismo. Cada constructor estará diseñado en una clase diferente, cada una de las cuales tendrá presente dos métodos básicos: el método ”__init__” o inicializador de clase, y el método ”printsvg” que será el responsable de devolver el string resultante con el código SVG final de cada diagrama.

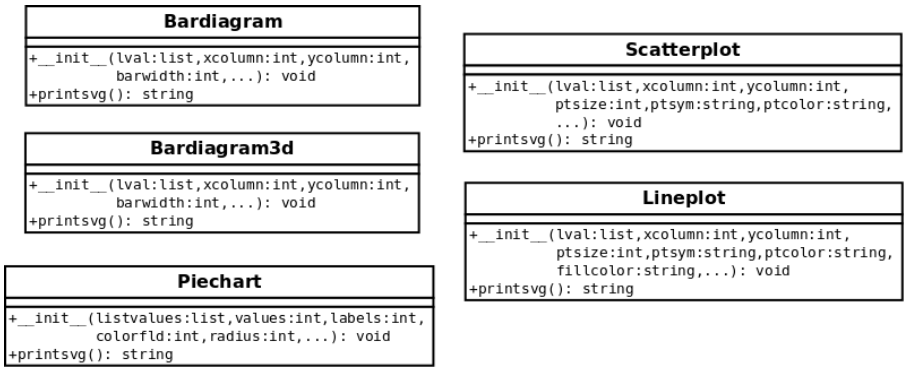

Figura 3.4: Clases constructoras

Sin embargo esto no es suficiente, el código debe tener una estructura concreta para que pueda ser interpretado correctamente por un navegador. Por ello se necesitará construir una escena que devuelva el SVG completo con la composición adecuada. Este método quedará incluido en la clase base *Svgelements* y definirá la cabecera y el cierre del documento mostrando la siguiente salida:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
<g id="body" style = "fill-opacity:1.0; stroke:black; stroke-width:1;">
.................. < código SVG del gráfico > ......................
</g> </svg>
```

Finalmente para escribir el SVG resultante en un fichero, basta simplemente con utilizar un método que se encargue generar o reemplazar el archivo en la ruta indicada y abrirlo en modo escritura para incluir dicho código. Esta función constará en el módulo ”filetext.py” y tendrá la siguiente cabecera:

```
def writesvgfile(path, outstring):
```

3.2.3. Prueba del incremento

Para probar esta iteración se ha a generado un documento SVG en el que se ha incluido un código simple para generar un objeto básico en SVG. Para esta fase hemos seleccionado el objeto rectángulo que se usará en posteriores iteraciones para componer ciertos diagramas.

Para ello se ha creado una nueva clase base en el módulo ”svgelements.py” llamada ”Rectangle” que como su propio nombre indica devolverá el código SVG de un elemento rectángulo.

Esta clase tendrá la misma arquitectura que las clases constructoras, un método inicializador ”__init__” así como el método de composición ”printsvg”:

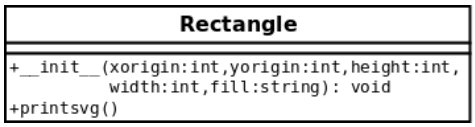

Figura 3.5: Clase Rectangle

Una vez invocado el método constructor de dicho elemento desde el módulo principal mediante el siguiente código:

```
import svgelements
import filetext
svgdoc = svgelements.Svgelements()
rectangle = svgelements.Rectangle(100, 100, 50, 50, ”red”)
begin, end = svgdoc.printsvg()
chartsvg = rectangle.printsvg()
filetext.writesvgfile(path, begin + chartsvg + end)
```

Obtendremos el siguiente resultado:

Figura 3.6: Rectángulo

Cuyo código SVG completo será el siguiente:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
<g id="body" style = "fill-opacity:1.0; stroke:black; stroke-width:1;">
<rect x="100" y="100" height="50" width="50" fill="red" />
</g> </svg>
```

3.3. Iteración 1

3.3.1. Requisitos

En esta iteración vamos a construir un diagrama de barras en dos dimensiones para un sólo grupo de datos de entrada. El requisito básico que deberá cumplir esta gráfica será simplemente la admisión de elementos externos a la misma para complementar la información. Dichos elementos serán: el titulo y la leyenda.

3.3.2. Desarrollo

En primer lugar vamos a realizar un análisis exhaustivo de la composición de un diagrama de barras. Si desglosamos la información que contiene cada uno de los elementos que conforman un gráfico de estas características podemos apreciar que:

- El eje de ordenadas esta formado por una serie de elementos línea con texto dispuestos a intervalos regulares de forma incremental.

- El eje de abscisas únicamente incluye la información de entrada mediante un elemento texto por lo que podrá ser incluido dentro del elemento columna que representará cada barra del diagrama el cual estará formado por un rectángulo y un texto superior variable.
- Opcionalmente también podrá incluir una malla en la parte trasera del panel de barras, que aportará una mayor información para la interpretación de los datos. Su diseño será realizado mediante una simple línea discontinua que se iniciará en el intervalo correspondiente del eje de ordenadas y finalizará en punto máximo del eje de abscisas.
- La leyenda esta únicamente compuesta por una columna horizontal construida mediante un rectángulo más un texto en su parte derecha
- Finalmente el titulo puede elaborarse mediante un simple elemento texto situado en la parte superior del diagrama

Para realizar cada parte será necesario la creación de una serie de clases individuales que corresponderán a los objetos básicos que conformarán el gráfico. Cada clase contendrá un método "__init__" responsable de la inicialización de los parámetros de entrada y un método "printsvg" que devolverá el string correspondiente al código SVG de cada objeto.

Para empezar vamos a centrarnos en el "eje de ordenadas". Mediante la clase línea *(Line)* construiremos la línea vertical del eje de ordenadas así como las líneas horizontales discontinuas que formarán la malla trasera de esta gráfica. Asimismo a través de la clase línea con texto *(Linetext)* se representarán los intervalos que indicarán los distintos valores que pueden tomar los datos de entrada. Vamos a analizar y a explicar detalladamente los parámetros necesarios para la creación de estos elementos.

Dos parámetros que juegan un papel muy importante para todos los diagramas incluidos en este proyecto, son *"xorigin"* e *"yorigin"*. Ambos parámetros constituirán la coordenada de inicio correspondiente a la esquina superior izquierda. Esta coordenada será el punto de partida para todos los objetos SVG que incluirá cada gráfica. Se tratará simplemente de dos *int.*

Centrándonos ya en el inicializador de la clase *Line* hay que destacar los siguientes parámetros:

- Los valores *endx* y *endy* establecerán la coordenada final del elemento línea y serán, al igual que los parámetros descritos anteriormente, dos *int*.
- Los valores decorativos *strokecolor* y *strokewidth* indicarán simplemente el color y anchura del elemento línea en cuestión y serán un *string* y un *int* respectivamente.
- Por último tendremos el valor *ygrid* el cual indicará si la línea debe o no ser discontinua.

Por otro lado, respecto a la clase texto *Text* cabe nombrar los siguientes parámetros:

- El valor denominado *text* indicará el texto que será mostrado al crear un objeto de esta clase.

- El valor *idtext* simplemente será un identificador dentro del documento SVG para posibles referencias posteriores a dicho elemento.
- Finalmente, *stroke* y *fontsize* serán valores decorativos que indicarán tanto el color como el tamaño del objeto texto.

La clase línea con texto *Linetext* será una clase derivada de las descritas anteriormente que incluirá un elemento línea junto a un elemento texto, ambos en una posición determinada respecto a la coordenada de partida.

```
Line (xorigin - offsetxlt, yorigin, xorigin, yorigin)
Text (xorigin - 4 * offsetxlt, yorigin)
```

El valor *offsetxlt* será simplemente un desplazamiento negativo en el eje de abscisas para situar el elemento correctamente en la escena del gráfico.

Vamos seguidamente a diseminar el ”panel de barras” para ver su composición y detallar los distintos parámetros que intervienen en la construcción de los distintos elementos que lo conforman.

Como ya se ha indicado anteriormente cada barra será un componente ”columna” que estará estructurado mediante un texto inferior correspondiente a los datos de entrada del eje de abscisas, un rectángulo y un elemento texto situado en la parte superior del mismo que será visualizado o no en base al valor de una determinada variable. Dicha variable será incluida en la línea de comandos por parte del usuario.

En este apartado nos centraremos en el análisis de la clase derivada denominada *Column* ya que es la hace confluir todos los parámetros de las clases básicas texto *(Text)*, rectángulo *(Rectangle)* y texto vertical *(Verticaltext)* para representar el objeto deseado.

En primer lugar hay que destacar los parámetros *xtext* e *ytext*, cuyos valores representarán los textos que serán visualizados en la parte inferior y superior del rectángulo respectivamente. Ambos valores serán de tipo *string*.

Por otro lado tendremos los valores de forma, *height* y *width*, a través de los cuales se representarán tanto la altura como la anchura que delimitarán el rectángulo. Éstos serán de tipo *int*. Este último valor nos servirá también para posicionar en la parte central del rectángulo, el elemento de texto vertical que será incluido en la parte inferior de la columna si el ancho de la misma es menor a 25 unidades.

El elemento rectángulo también tendrá un valor decorativo que nos permitirá elegir el color de relleno del mismo. Este sera el valor *fillcolor* y estará representado por un *string*.

Otros parámetros clave son por un lado, el valor *idrect* de tipo *string* que será el identificador del rectángulo dentro del documento SVG y cuya labor será muy importante a la hora de realizar filtros o animaciones sobre este elemento. Por otro lado el valor *vals* de tipo booleano indicará

si hay que mostrar o no el texto superior del objeto columna cambiando totalmente la presencia de la escena.

Por último, pero no menos importante hay que hablar de los elementos externos al diagrama (titulo y leyenda) que servirán para ampliar la información del mismo y serán comunes a todos los gráficos incluidos en el proyecto. El titulo como ya se ha establecido anteriormente será representado mediante un elemento texto y la leyenda por otra parte estará compuesta un objeto columna horizontal *hcolumn.*

También jugarán un papel sumamente importante los identificadores de objeto ya que, haciendo referencia a cada uno de ellos: rectángulo *(idrect)* o texto *(idtext)* o bien al grupo completo a través del identificador de grupo *(idgroup)* se podrán establecer diversos efectos estilísticos sobre los diagramas, efectos de los cuales se hablará en iteraciones posteriores.

Todos estos valores anteriores serán introducidos en el método inicializador "__init__" de la clase *Bardiagram* junto a las opciones de entrada que serán introducidas por el usuario a través de línea de comandos y que serán las encargadas de especializar el gráfico.

```
Bardiagram.__init__(self, lval, xcolumn, ycolumn, barwidth, xorigin, yorigin, delim,
vals, yinc, yrange, ygrid, fillcolor, name, title, legend)
```

La lista de valores será indicada en el parámetro *lval* e incluirá tanto los argumentos correspondientes a los valores del eje de abscisas como la altura relativa de cada barra. Para distinguir entre unos u otros utilizaremos los parámetros *xcolumn* e *ycolumn*, que serán dos *int* y representarán el indicador de la columna de datos correspondiente utilizándose como índice en la lista de valores para acceder al elemento correcto en cada caso.

Los valores *yinc* e *yrange* representarán el valor de incremento y el valor inicial respectivamente en el eje de ordenadas. Ambos serán de tipo *int*.

Por otro lado, *ygrid* sera el parámetro indicador de presencia respecto a la malla posterior al panel de barras. Su tipo también sera *int*.

En las barras intervendrán dos parámetros esenciales. En primer lugar *bardwidth*, un *int* que especificará la anchura de cada barra y en segundo lugar *delim* también de tipo *int* e indicará la separación existente entre cada una de ellas.

Finalmente *name*, *title* y *legend* serán los parámetros de especificación para los elementos externos ya descritos previamente. Ambos de tipo *string*, el primero indicará el nombre del grupo de datos en la leyenda y el segundo el texto que será incluido en la parte superior central del diagrama. El último será un indicador de presencia de tipo *booleano* que determinará si se desea incluir o no la leyenda dentro de la escena del gráfico.

Una vez identificadas todas las partes que intervienen en el diagrama vamos a describir con detalle el método "printsvg" encargado de obtener el código SVG del gráfico mediante la composición de cada una de ellas.

Antes de nada debemos identificar cual será el punto máximo en ambos ejes. Para hallar el del eje de abscisas hay que tener en cuenta el numero de barras que van a ser dibujadas así como la delimitación establecida entre cada una de ellas y la anchura de las mismas. Este valor será calculado mediante la siguiente instrucción:

```
endbars = xorigin + (delim * numbars) + (barwidth * numbars)
```

Este valor será utilizado como limite para dibujar las líneas discontinuas del fondo del diagrama en caso de que el parámetro ygrid tome el valor ”yes”.

El punto máximo del eje de ordenadas será equivalente a la altura de la barra más alta, *heightmaxbar*. Para hallar este valor se ha generado una función auxiliar llamada *setmaximumbars* que toma como parámetro la lista de valores de la columna *”y”* de entrada y devuelve el valor máximo de la misma.

```
def setmaximumbars(self, lval):
```

Para dibujar el eje de ordenadas hay que crear en primer lugar la línea vertical hasta la posición indicada por *heightmaxbar* teniendo siempre en cuenta el valor del parámetro *yrange* que nos indicará a que altura comenzará este eje, bien en el valor 0, por defecto, o bien en un incremento indicado por el usuario mediante dicho argumento de entrada.

```
vertical_line = Line(xorigin, (yorigin + heightmaxbar - yrange), xorigin, yorigin)
```

Posteriormente rellenaremos este eje con las líneas con texto iterando en un bucle while hasta que el valor del incremento sea mayor que *heightmaxbar.* El texto que mostrará este elemento corresponderá al valor del incremento perteneciente a cada iteración.

```
linetext = Linetext(xorigin, yorigin + heightmaxbar - inc, fontsize, str(inc))
inc = yrange + (yinc * counter)
```

En este mismo bucle se realizará la comprobación que determinará si hay que mostrar o no la línea discontinua correspondiente a la malla de fondo. En caso afirmativo se dibujará dicha línea junto a cada objeto línea con texto hasta el punto máximo del eje de abscisas.

```
backgroundline = Line(xorigin, yorigin + heightmaxbar - inc, endbars, yorigin +
heightmaxbar - inc, ygrid)
```

Por último se crearán las barras junto a sus nombres iterando mediante un bucle for objetos columna hasta la longitud de la lista de valores de entrada correspondiente al primer conjunto de datos indicado por el parámetro *xcolumn.*

```
column = Column(xvalue, yvalue, yvalue - yrange, barwidth, xorigin, yoriginbar,
fillcolor, "colum_" + str(cont), vals)
```

De manera opcional el usuario podrá incluir los elementos externos descritos inicialmente, la leyenda y el titulo. Para ello se harán dos comprobaciones previas, por un lado verificaremos si la cadena de texto *title* no consta vacía y por otro la presencia del argumento *legend*. En caso de que el usuario haya indicado el titulo del gráfico se situará un elemento texto en la parte central superior de la gráfica.

```
bartitle = Text(endbars / 2, yorigin / 2, 14, title)
```

Si *legend* consta en los argumentos de línea de comandos se mostrará en el lado derecho del diagrama un rectángulo junto a un texto aportado por el parámetro *name* que será incluido a través de la misma vía de entrada. Este parámetro de tipo *string* será el identificador del conjunto de datos.

```
namelegend = Hcolumn(name, endbars + offsetlegend, yorigin, fillcolor, vbarsizelegend,
vbarsizelegend, "vbarlegend", "vbarect", "vbartext", "", "", False, "none")
```

Los parámetros *offsetlegend* y *vbarsizelegend* simplemente indicarán el desplazamiento de la leyenda respecto al gráfico en el eje de abscisas así como el tamaño del rectángulo de la misma.

Finalmente cabe destacar la variable más importante en todo este proceso. Esta será la variable *string* y englobará todas las cadenas de texto resultantes de los métodos ”printsvg” de cada objeto. Dicha variable sera la que genere el valor de retorno del método *Bardiagram.printsvg()* devolviendo así el código SVG del gráfico.

3.3.3. Prueba del incremento

Para probar este incremento se ha desarrollado el siguiente código en el módulo principal *pySVG.py:*

```
import svgelements
import filetext

def main():
options, args = getopt.getopt(sys.argv[1:], "h", ["help", "prefab=", "delim=",
"x=", "y=", "vals", "yrange=", "yinc=", "ygrid=", "barwidth=", "color=",
"color2=", "legend","name=", "title="])
for option, arg in options:
     if option in ("-h", "--help"):
     print __doc__ sys.exit(2)
if option == ("--prefab"):
     prefab = arg
if option == ("--delim"):
     delim = arg
.....
```

```
for inputargs in args:
     lval = filetext.readinputfile(inputargs)
svgdoc = svgelements.Svgelements()
begin, end = svgdoc.printsvg()
if prefab == ”bardiagram”
     bardiagram = svgelements.Bardiagram(lval, xcolumn, ycolumn, barwidth,
xorigin, yorigin, delim, vals, yinc, yrange, ygrid, color,      name, title,
legend)
chartsvg = bardiagram.printsvg()
filetext.writesvgfile(”vbars2D.svg”, begin + chartsvg + end)


if __name__ == ’__main__’:
main()
```

Donde *args* es la lista de datos de entrada introducida a través de un fichero de texto con el siguiente formato:

```
EUL      366
EFA       42
EDD       15
ELDR      67
EPP      276
UEN       27
Other     66
```

Y las opciones de especificación almacenadas en la variable *options* introducidas a través de línea de comandos han sido las siguientes:

```
--prefab bardiagram --delim=25 --x=1 --y=2 --vals --yrange=0 --yinc=20 --ygrid=yes
--color=powderblue --barwidth=25 --name=alumnos_primer_ciclo vbar1g.txt
```

Nótese que la nomenclatura obligatoria para introducir las mismas sera siempre dos guiones simples (- -) seguidos del nombre de la opción y un signo de igualdad (=) posterior al cual se establece valor del argumento *sin espacios*. En caso de que sea un argumento de presencia *(boolean)* se omitirá el signo de igualdad y el valor del mismo.

A continuación se muestra el SVG resultante:

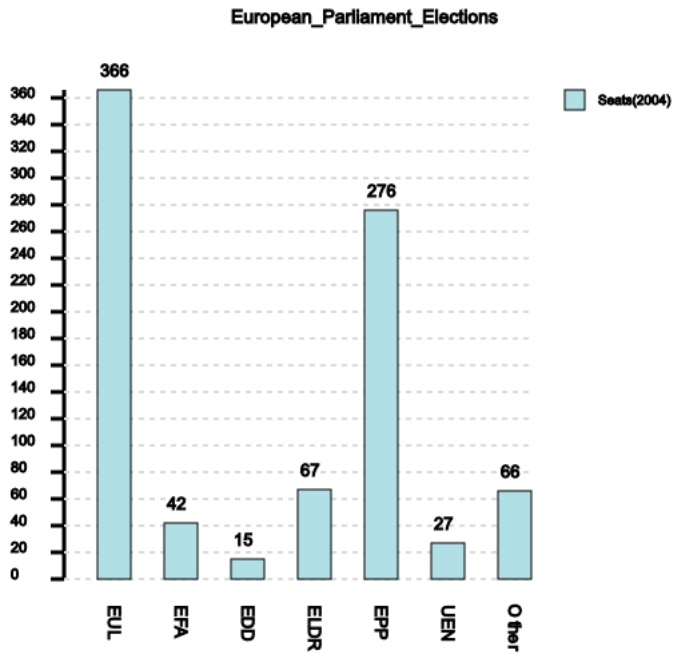

Figura 3.7: Resultado iteración 1

3.4. Iteración 2

3.4.1. Requisitos

En esta iteración vamos a usar lo aprendido hasta ahora para mejorar el diagrama de barras realizando una ampliación del mismo de manera que soporte un nuevo grupo de datos.

3.4.2. Desarrollo

La composición básica del diagrama sera exactamente igual que la gráfica de la iteración anterior con la salvedad de que habrá que incluir más elementos columnas y establecer las delimitaciones entre estas de manera diferente.

Para que esta iteración funcione correctamente deberán realizarse algunas modificaciones en determinados métodos y funciones de la clase *Bardiagram*. El primer cambio se realizará en

la función *setmaximumbars()*. Esta función deberá hallar el valor máximo de ambos grupos de datos, el grupo inicial determinado por el parámetro *ycolumn* y el grupo secundario indicado mediante el argumento *ycolumn2* de manera que *heightmaxbar* contenga la mayor altura dentro del montante total de ambos grupos de barras. Para ello se incluirá una nueva iteración *for* de modo que si la altura máxima del grupo inicial almacenada en *heightmaxbar* es menor a una variable auxiliar denominada *auxmaxbar* se realizará una sustitución asignándose a *heightmaxbar* este nuevo valor.

Los sucesivos cambios que realizaremos se harán dentro del método constructor del gráfico, *Bardiagram.printsvg()*. En este método lo primero que haremos será modificar el punto máximo del eje de abscisas almacenado en la variable *endbars*. Para explicar este nuevo valor se ha diseñado el siguiente diagrama explicativo:

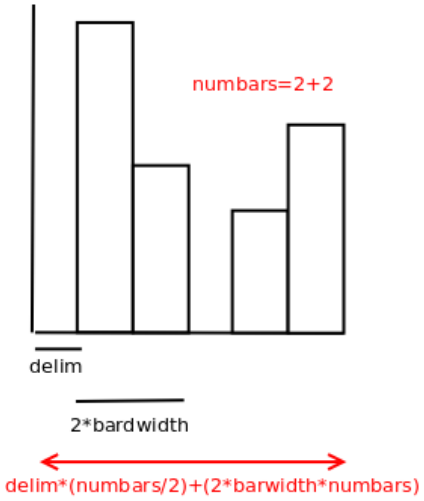

Figura 3.8: Diagrama explicativo

Cabe destacar que se parte siempre del punto inicial *(xorigin, yorigin)* con lo cual habrá que sumar el valor de esta coordenada a los datos indicados.

El siguiente cambio que realizaremos será en la iteración correspondiente a las columnas. Dentro de este bucle *for* se verificará si existe la componente *ycolum2* comprobando si la longitud de la lista de valores de entrada es igual al valor de este indicador. En caso afirmativo se creará un nuevo grupo de columnas estableciéndose su origen justo al final de cada barra perteneciente al grupo inicial.

```
column2 = Column("", yvalue2, yvalue2 - yrange, barwidth, xorigin + barwidth,
yoriginbar2, fillcolor2, "colum2_" + str(cont), vals)
```

Este nuevo grupo de barras no contendrá el texto inferior para no repetir información pero si tendrá su propio color de relleno indicado por el usuario a través del parámetro *fillcolor2* de tipo *string*.

Se establecerá también un nuevo componente en la leyenda, *name2* de tipo *string* que corresponderá al texto que da nombre al segundo conjunto de datos.

```
namelegend2 = Hcolumn(name2, endbars + offsetlegend, yorigin + 2 * vbarsizelegend,
fillcolor2, vbarsizelegend, vbarsizelegend, "vbarlegend1", "vbarrect2", "vbartext2",
"", "", False, "none")
```

3.4.3. Prueba del incremento

Para probar este incremento se ha modificado el código de la iteración anterior sustituyendo en el módulo principal *pySVG.py* la llamada a la clase *Bardiagram* por esta otra:

```
bardiagram = svgelements.Bardiagram(lval, xcolumn, ycolumn, ycolumn2,
barwidth, xorigin, yorigin, delim, vals, yinc, yrange, ygrid, color, color2,
name, name2, title, legend)
```

Y se ha añadido una nueva columna al fichero de texto:

```
EUL      366     49
EFA       42     56
EDD       15     19
ELDR      67     60
EPP      276    272
UEN       27     36
Other     66     29
```

Las opciones de especificación quedan como sigue:

```
--prefab bardiagram --delim=25 --x=1 --y=2 --y2=3 --vals --yrange=10 --yinc=20
--ygrid=yes --color=powderblue --color2=orange --barwidth=30 --name=Seats(2004)
--name2=Seats(1999) --title=European_Parliament_Elections --legend vbar2g.txt
```

A continuación se muestra el SVG resultante:

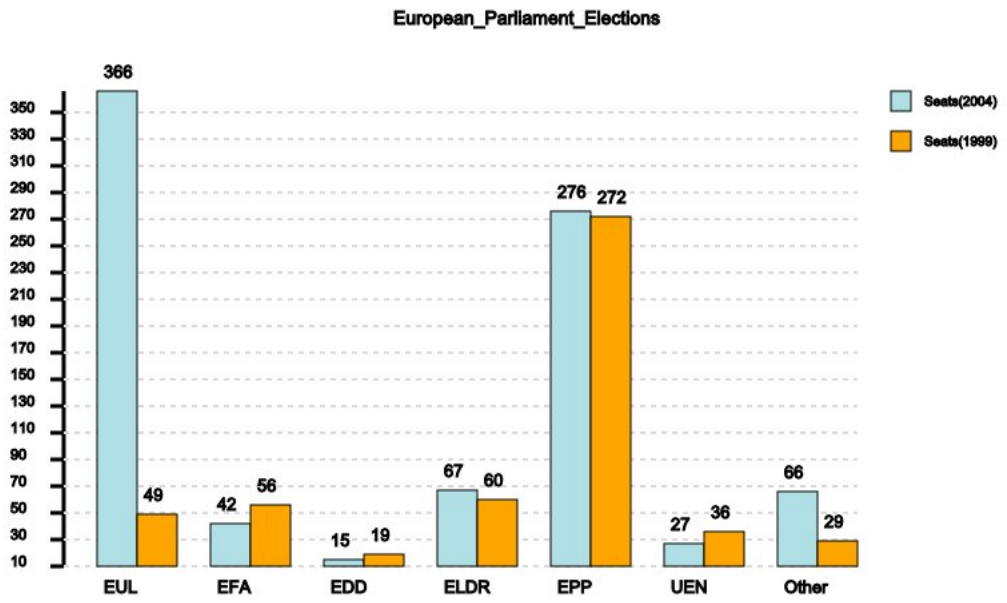

Figura 3.9: Resultado iteración 2

3.5. Iteración 3

3.5.1. Requisitos

No hay requisitos específicos para esta esta sección.

3.5.2. Desarrollo

En esta iteración se va a ejecutar una nueva modificación del diagrama de barras para crear un gráfico en tres dimensiones. Es importante destacar que las barras ya no serán simples rectángulos. Para representar cada barra en el plano tridimensional y ofrecer la sensación de profundidad deseada, se deberá crear una nueva clase adicional ”polígono” que será la responsable de generar la parte superior y lateral del ”rectangulo3d”, convirtiendo dicho elemento en un cubo volumétrico.

La creación del gráfico será muy similar al diagrama de barras respecto a ejes y elementos externos por lo que solo nos detendremos a examinar lo realmente importante, es decir, los cambios respecto al gráfico inicial. Dichos cambios serán básicamente la creación de las nuevas barras tridimensionales así como la creación del rectángulo, también en tres dimensiones, situado en la parte inferior del panel de barras y creado para aportar mayor detalle al gráfico.

Centrándonos en la construcción de las barras del diagrama podemos apreciar que surgen dos clases nuevas: *Rectangle3d* y *Colum3d,* siendo la segunda una clase derivada de la primera.

Esta nueva clase *Rectangle3d* como ya se ha indicado previamente es una composición del rectángulo bidimensional más dos objetos polígono que determinarán las paredes superior y lateral del rectángulo en el plano tridimensional. Así mismo la clase *Column3d* será la suma del objeto *Rectangle3d*, el texto correspondiente al eje de abscisas en su parte inferior y el texto superior opcional correspondiente a la altura de cada barra.

```
Rectangle3d = Rectangle.printsvg() + Polygon.printsvg() + Polygon.printsvg()
Column3d = Rectangle3d.printsvg() + Text.printsvg() + [Text.printsvg()]
```

La clase *Polygon* tendrá al igual que las demás, su método inicializador y un método ”printsvg” que devolverá el código SVG de dicho elemento. La construcción de un polígono es sumamente sencilla.

```
Polygon. __init__(pointlist, idpolygon, fillcolor)
```

El elemento principal que interviene en todo el proceso de creación del objeto polígono es la lista de puntos *pointlist* entre los cuales se generarán los diferentes segmentos que compondrán el mismo. El identificador del objeto de tipo *string, idpolygon* hará posible posteriores referencias a este objeto para incluir determinados efectos estilísticos en él. El parámetro *fillcolor* sera otro *string* que indicará el color de relleno que tendrá este elemento.

Vamos a analizar los cambios surgidos en la interfaz principal ”printsvg” de la clase *Bardiagram3d* respecto al anterior para determinar cuales son los parámetros clave que hacen posible dichas modificaciones.

El valor que juega el papel más importante en este gráfico será el desplazamiento en los ejes, *offset*, el cual se ha almacenado dentro de una constante interna ya que su valor no va a sufrir ningún tipo de alteración durante todo el proceso de construcción del diagrama. Este valor indica la altura y anchura respectivamente de los polígonos adyacentes.

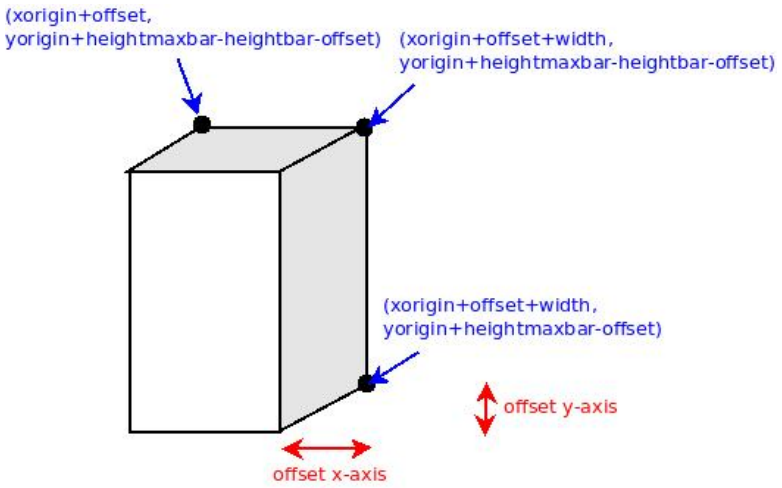

Figura 3.10: Offset

El valor de desplazamiento interferirá tanto en la construcción de las barras como en la elaboración de la malla trasera discontinua ampliando el margen derecho de la escena. A diferencia del diagrama de barras inicial en esta gráfica dicha malla estará compuesta por dos tramos de línea, el primero de ellos hasta el valor de incremento en el eje de ordenadas más un desplazamiento en ambos ejes del plano cartesiano. El segundo por otro lado, será equivalente a la línea de fondo inicial, la cual comenzará en el punto final del primer segmento.

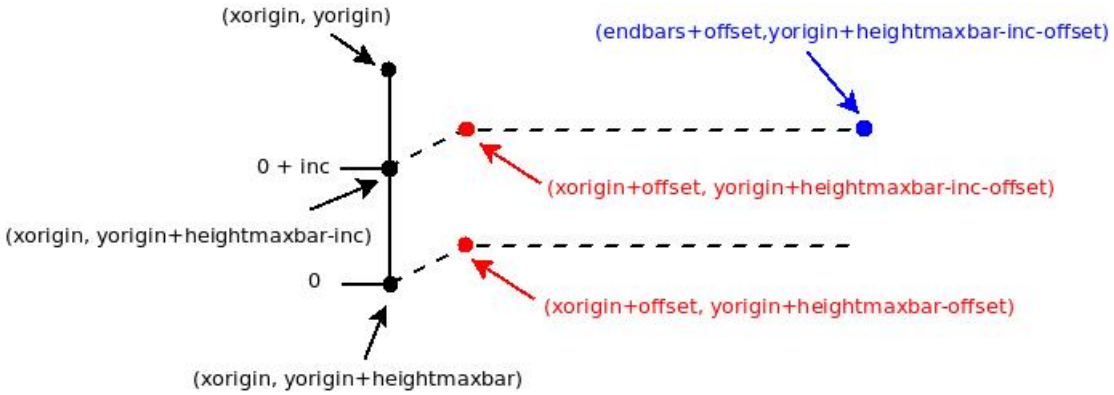

Figura 3.11: Malla de fondo

Cabe destacar que los desplazamientos en el plano cartesiano SVG respecto al eje de ordenadas suceden a la inversa, es decir, el incremento positivo en dicho eje equivaldría a un desplazamiento negativo en el plano de coordenadas normal y viceversa. Por este motivo el texto aparecerá representado mediante un incremento positivo pero a nivel interno de coordenadas hay que realizar este movimiento al contrario. Lo mismo ocurre para otros los incrementos positivos como las alturas de barras, siempre trabajaremos de la misma forma, restando el valor correspondiente para figurar un incremento positivo en el eje de ordenadas del objeto mostrado.

En caso de que el parámetro *yrange* sea distinto de 0 deberemos restar este valor en la ordenada de todo los objetos del diagrama en los que se encuentre presente.

3.5.3. Prueba del incremento

Para probar este nuevo incremento se ha añadido una nueva funcionalidad al módulo principal *pySVG.py* denominada *getprocessargs()* que será la encargada de recoger las opciones de entrada y establecerlas correctamente en cada diagrama. A medida que se vayan ampliando los diagramas generados en este proyecto crecerá el numero de argumentos en la cabecera de dicho método. Dentro de este se ha construido una nueva rama de manera que reconozca esta nueva implementación:

```
def getprocessargs(args, prefab, xcolumn, ycolumn, ycolumn2, barwidth, xorigin,
yorigin, delim, vals, yinc, yrange, ygrid, title, legend, name, name2, color, color2):
for inputargs in args:
     lval = filetext.readinputfile(inputargs)
svgdoc = svgelements.Svgelements()
begin, end = svgdoc.printsvg()
if prefab == ''bardiagram'':
     path = ''vbars2D''
     bardiagram = svgelements.Bardiagram(lval, xcolumn, ycolumn, ycolumn2, barwidth,
xorigin, yorigin, delim, vals, yinc, yrange, ygrid, color, color2, name, name2, title,
legend)
     chartsvg = bardiagram.printsvg()
elif prefab == ''bardiagram3d'':
     path = ''vbars3D''
     bardiagram3d = svgelements.Bardiagram(lval, xcolumn, ycolumn, barwidth, xorigin,
yorigin, delim, vals, yinc, yrange, ygrid, color, title, legend, name)
     chartsvg = bardiagram3d.printsvg()
filetext.writesvgfile(path, begin + chartsvg + end)
```

Donde *args* sigue siendo la lista de datos de entrada introducida a través de un fichero de texto con formato:

```
EUL      366
EFA       42
EDD       15
ELDR      67
EPP      276
UEN       27
Other     66
```

Y las opciones de especificación introducidas a través de línea de comandos son las siguientes:

```
--prefab bardiagram3d --delim=50 --x=1 --y=2 --yrange=0 --yinc=20 --ygrid=yes
--barwidth=25 --color=pink --filtered --title= European_Parliament_Elections --legend
--name=Seats(2004) --vals vbar3d.txt
```

Obteniendose como resultado el siguiente SVG:

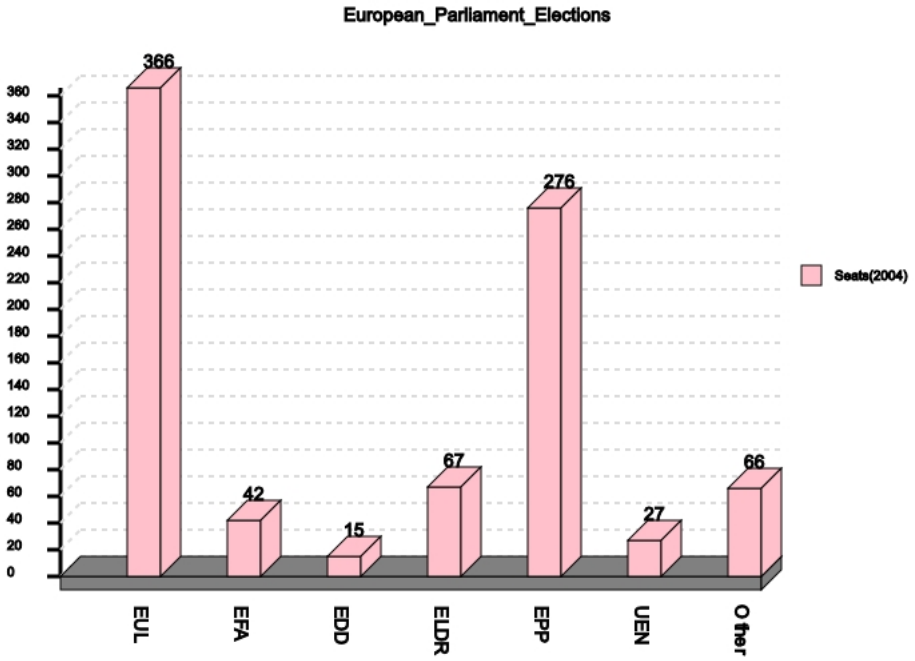

Figura 3.12: Resultado iteración 3

3.6. Iteración 4

3.6.1. Requisitos

Esta sección no será relevante en lo que a requisitos especiales se refiere por lo que omitimos este apartado.

3.6.2. Desarrollo

En esta iteración vamos a incluir una nueva implementación para representar el diagrama de sectores. En primer lugar vamos como siempre a analizar la composición del mismo para determinar que elementos serán necesarios para su construcción.

Un diagrama de sectores básicamente se puede representar con un único tipo de objeto, el camino o *path*. Mediante este elemento representaremos cada sector de la ”tarta”, la cual estará compuesta por el conjunto de todos ellos.

Los elementos externos como titulo y leyenda estarán representados al igual que en las iteraciones anteriores por un elemento *texto* así como un elemento *columna horizonta*l por cada ”porción” de la tarta.

Analizando ya la construcción cada de sector se puede ver que en su desarrollo intervienen multitud de parámetros incluidos en la clase *Path*.

El primer parámetro del que vamos a hablar es el tamaño del radio de la tarta, *radius*. Este valor de tipo *int* jugará un papel muy importante en este diagrama ya que proporcionará un método de acceso a la coordenada central del mismo siendo así el punto de partida de todos los elementos camino que formarán el gráfico:

```
xradius = xorigin + radius
yradius = yorigin + radius
```

Otro valor sumamente importante es *colorfld*. Este, determinará la columna del fichero de texto que contiene los colores de relleno de los diferentes sectores. Será también de tipo *int*. Así mismo, el valor *pos* determinará el sector de la tarta en el cual nos encontramos trabajando permitiendo junto al anterior acceder al dato correspondiente dentro de la lista de valores.

Los valores *initianradian* y *radian* serán dos *float* que especificarán el origen y tamaño del sector respectivamente. Por último *idpath* será una cadena de texto que identificará cada sector.

Centrándonos ya en el método *printsvg()* de la clase *Path* vemos que los caminos se construyen dibujando una figura en base a una serie de instrucciones indicadas en su atributo *d*. Para nuestro caso en particular tendrá una cadena de texto con la siguiente disposición:

- M más dos valores separados por una coma lo que permitirá moverse a un punto determinado sin dibujar ninguna línea. Aquí introduciremos el valor de la coordenada central del gráfico *(xradius, yradius)*.

- L más dos valores separados por una coma. Esto creará una línea hasta la coordenada indicada tomando como referencia la posición anterior permitiéndonos dibujar así los radios del diagrama. Para todos los sectores aplicaremos la misma regla:

```
Lx = xradius + cos(initianradian) * radius
Ly = yradius + sen(initianradian) * radius
```

- Para crear el arco de cada sector utilizaremos A. Esto creará una línea elíptica con una serie de especificaciones. Los dos primeros valores separados por coma indicarán el tamaño del arco en ambos ejes, estos valores serán el radio de la circunferencia. Seguidamente separado por un espacio se indicará un 0. Este valor será constante e indicará la dirección de giro del arco respecto al sistema de coordenadas. Tras él vendrá un 0 o un 1 dependiendo del tamaño del sector. Si éste es menor o igual a 180 grados *(la mitad de la circunferencia)* se introducirá un 0 y en caso contrario un 1. Separado del anterior por una coma tendremos un 1 que indicará que el giro se realiza en sentido de las agujas del reloj. El último par de valores separados por coma indicarán la coordenada final del trazo, esta se calculará atendiendo a la siguiente regla:

```
x = xradius + cos(finalradian) * radius
y = yradius + sen(finalradian) * radius
```

- Finalmente para cerrar el camino y dibujar así la *"porción"* se utilizará el elemento Z.

Dentro de la clase *Piechart* constarán todos los métodos y funciones encargados de realizar los cálculos pertinentes para mostrar cada sector de forma adecuada.

La primera función de la que hablaremos es *totalsum()*. Esta función devolverá en la variable de instancia *sumvalues* la suma total de todos los valores especificados en la columna del fichero de datos de entrada indicada por el parámetro, de tipo *int, values* resultando muy útil para transformar el valor de cada sector a radianes. Esta gestión se realizará en otra función auxiliar denominada *valuestoradian()* que aplicará la siguiente regla a cada sector:

```
sectorvalue = (2 * pi) / sumvalues
```

Devolviendo así en la variable de tipo *list*, *radianvalues*, una lista con el conjunto de valores convertidos en radianes.

Hagamos ahora una breve descripción del funcionamiento del método *printsvg()* que nos devolverá el SVG resultante. Este método transformará en primer lugar la lista de valores iniciales a radianes usando las funciones anteriormente comentadas. Posteriormente se dibujarán los sectores así como un elemento texto que indicará el porcentaje equivalente del mismo respecto al total y su leyenda correspondiente (si el usuario así lo indica en la línea de comandos) iterando mediante un bucle *for*.

Para situar los elementos en esta gráfica se van a aplicar siempre las reglas estándar de trigonometría:

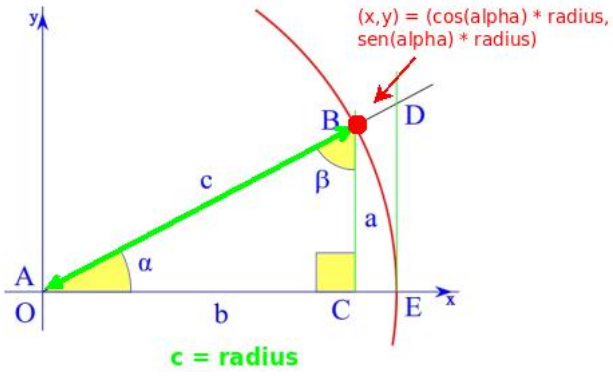

Figura 3.13: Reglas trigonométricas

Donde `(alpha)` será el tamaño en radianes del sector correspondiente determinado por la variable *pos*. Este valor será almacenado en la variable de tipo *float*, *initianradian* y actualizado continuamente comenzando por un valor equivalente a 0 el cual será incrementado en cada iteración del bucle mediante la instrucción:

```
values = radianvalues[pos]
initianradian = initianradian + value
```

Por tanto para dibujar cualquier elemento en la mitad del arco del radian su valor tendrá que ser:

```
initianradian + value / 2
```

Cabe destacar que para mostrar correctamente el texto con el valor del porcentaje en la parte exterior central del arco de cada sector hay que tener en cuenta que si el montante de los radianes hasta ese momento esta entre: `135 grados (3*pi/ 4) y 225 grados (5*pi/4)` debe desplazarse el texto un porcentaje mayor a la izquierda en el eje de abscisas. Este valor será equivalente a `(radius*1.4)` en lugar de `(radius*1.2)`.

Una vez dibujados todos los sectores se comprobará si la variable *title* de tipo *string* no esta vacía para incluir mediante un objeto texto el titulo del diagrama en la parte central superior de la escena finalizando así la composición del gráfico.

3.6.3. Prueba del incremento

Para probar este nuevo incremento se ha añadido una nueva rama en el método *getprocesargs()* del módulo principal con el siguiente código:

```
elif prefab == "pie":
   path = ”piechart.svg”
   piechart = svgelements.Piechart(xorigin, yorigin, radius, 0, lval, values, labels,
colorfld, legend, animate, filtered, title)
   chartsvg = piechart.printsvg()
```

Se ha creado un nuevo fichero de datos de entrada con el siguiente formato:

```
EUL       39      red
PES      200      crimson
EFA       42      forestgreen
EDD       15      khaki
ELDR      67      gold
EPP      276      darkblue
UEN       27      cornflowerblue
```

Y por último se han incluido las siguientes especificaciones a través de línea de comandos:

```
--prefab=pie --delim=50 --values=2 --labels=1 --colorfld=3 --legend
--title=European_Parliament_Elections pie.txt
```

Obteniéndose como resultado el siguiente diagrama:

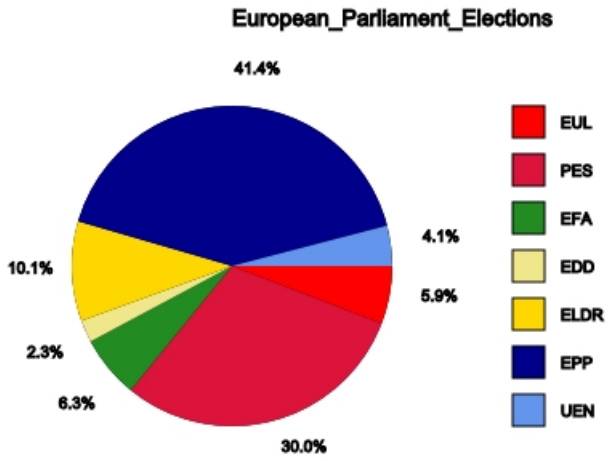

Figura 3.14: Resultado iteración 4

3.7. Iteración 5

En este incremento añadiremos al proyecto un nuevo gráfico, el diagrama de puntos también conocido como ”diagrama de dispersión”.

3.7.1. Requisitos

Los requisitos que deberá cumplir esta iteración serán los siguientes:

- Deberá admitir cinco tipo de figuras para la representación de cada coordenada: rectángulos, círculos, diamantes, triángulos y triángulos invertidos.
- Permitirá incluir la recta de regresión del conjunto de puntos.
- Adicionalmente permitirá aparte de los elementos externos estándar como titulo y leyenda, etiquetas identificativas en ambos ejes.

3.7.2. Desarrollo

Para diseñar el diagrama de dispersión lo primero es analizar su composición como en iteraciones previas. Un diagrama de dispersión se compone principalmente de:

- Una línea vertical junto a varias líneas con texto que representarán el eje de ordenadas.
- Una línea horizontal junto a varias líneas con texto vertical que representarán el eje de abscisas.

- Una nube de puntos que representará las coordenadas introducidas a través del fichero de entrada.
- Una serie de elementos externos para ampliar la información: titulo, leyenda y etiquetas.

En esta gráfica surge una nueva clase *Circle* que se encargará de representar las coordenadas de entrada mediante dicha figura. Los parámetros principales de este objeto serán: el origen de coordenadas *(xorigin ,yorigin)* de tipo *int* que será situado en el centro del objeto y corresponderá con la coordenada de cada punto del fichero de entrada, el radio del círculo *(radius)* que identificará el tamaño del objeto también de tipo *int*, la anchura de la línea que determina la circunferencia *(strokewidth)* nuevamente de tipo *int* y finalmente el color de relleno del objeto *(fillcolor)* de tipo *string*.

Centrándonos ya en la clase *Scatterplot* podemos apreciar que cada punto del gráfico va a tener sus propios atributos estilísticos: color *(ptcolor:string)*, tamaño *(ptsize:int)* y forma *(ptsym:string)* siendo este último el que determinará la figura elegida para representar cada punto del conjunto de coordenadas *(rectangle, circle, diamond, triangle ó invertedtriangle)*.

Habrá cinco tipos de figuras como ya se ha indicado en los requisitos previos. Las figuras básicas como rectángulo o círculo serán representadas por las clases con el mismo nombre. En cuanto a las demás, (triangulo, diamante y triangulo invertido), se representarán con uno de los elementos más versátiles del estándar SVG, el polígono, el cual permitirá dibujar cada una de ellas mediante un conjunto de puntos unidos por segmentos.

Otro valor muy importante en esta gráfica sera *corr*. Este parámetro de tipo booleano indicará la presencia o ausencia de la recta de regresión que medirá el grado de dependencia de las coordenadas. Esta recta será representada mediante un objeto de la clase *Linepath* que será un tipo especial de camino compuesto únicamente por segmentos en línea recta. Para construirlo será por tanto necesario la lista de puntos que atravesará dicho objeto. Este valor sera almacenado en el parámetro *lpoints* dando al elemento el siguiente aspecto:

```
<path id="regline" d=" M115.0,223.827623126 L115.0,223.827623126 L290.0,289.593147752
L134.0,230.967880086 L223.0,264.414346895 L156.0,239.235546039 L169.0,244.120985011
L143.0,234.350107066 L233.0,268.172376874 L155.0,238.859743041 L130.0,229.464668094 "
style="stroke:blue; stroke-width:1; fill:none"/>
```

El valor de M corresponderá a la primera coordenada de la lista a partir de la cual se generarán los trazos consecutivos que mostrarán la recta.

Todos los parámetros indicados anteriormente serán introducidos en el inicializador de la clase Scatterplot.

Describamos ahora el funcionamiento del método ”printsvg” que nos devolverá el código SVG del gráfico resultante. Este método creará en primer lugar las líneas horizontal y vertical mediante dos elementos de la clase *Line* que representarán los ejes de coordenadas. Estas rectas serán la base para la representación de las diferentes líneas con texto (horizontal o vertical) que

proporcionarán los distintos valores que podrán alcanzar las coordenadas respecto a un valor de incremento seleccionado por el usuario a través del parámetro *yinc*. La construcción de estas líneas será similar a las secciones anteriores iterándose en un bucle *while* hasta que el valor del incremento supere el valor máximo de cada eje.

Posteriormente se realizará la creación de la nube de puntos. Estos estarán englobados en una lista de valores que sera recorrida mediante un bucle *for*. Dentro de este bucle se chequeará el valor del parámetro *ptsym* para representar la figura adecuada mediante la instancia de clase correspondiente: círculo, rectángulo o polígono. Cabe indicar que los puntos para representar estos objetos tendrán cuenta que el valor de la coordenada actual será el centro de la figura dibujándose la misma alrededor de dicho punto de la siguiente manera:

```
if ptsym == "diamond":
    diamondpoints = [[xpoint, ypoint - ptsize / 2], [xpoint + ptsize / 2, ypoint],
[xpoint, ypoint + ptsize / 2], [xpoint - ptsize / 2, ypoint]]
    shapepoint = Polygon(diamondpoints, ptsym, self.ptcolor)
    string += shapepoint.printsvg()
```

donde *xpoint* e *ypoint* son las coordenadas del punto actual.

En caso de que el parámetro *corr* este presente en la línea de comandos representaremos la recta de regresión del conjunto de puntos. Para ello hay que hablar de una serie de funciones auxiliares que serán las encargadas de ejecutar los cálculos matemáticos para obtener los diferentes puntos contenidos en la misma.

La recta mostrada será la *recta de regresión de y sobre x* donde ”x” sera la coordenada independiente e ”y” la coordenada dependiente. Esta recta será construida mediante la función *getregressionline(lval)* y tendrá la siguiente ecuación para cada coordenada del plano en el eje de ordenadas:

$$y = \frac{Cov(x,y)}{\nabla x^2}(x-\bar{x})+\bar{y}$$

donde:

$$b = \frac{Cov(x,y)}{\nabla x^2} \qquad \nabla x^2 = \frac{\Sigma(x-\bar{x})^2}{n} \qquad Cov(x,y) = \frac{\Sigma(x-\bar{x})(y-\bar{y})}{n} \qquad \bar{x} = \frac{\Sigma x_i}{n} \qquad \bar{y} = \frac{\Sigma y_k}{n}$$

Siendo *”n”* es el numero total de elementos de la lista. Cada función auxiliar se encargará de realizar uno de estos cálculos:

$\bar{x},\bar{y} =$ `getaverage(lval)`
$\nabla x^2 =$ `getvariance(lval)`
$Cov(x,y) =$ `getcovariance(lval)`

Las funciones *getmaxpoint(lval)* y *getmaxaxis (lval)* proporcionarán los valores máximos de ambos ejes y serán determinantes para establecer las coordenadas de la recta de regresión ya que antes de plasmar la misma en el gráfico debemos transformar con el método *regtocoordenates(lval, ymaxaxis)* cada coordenada a su equivalente en el plano cartesiano teniendo en cuenta que: $\hat{y_i} = ymaxaxis - y_i$. Así, la lista final de valores que contendrá la recta de regresión sera la siguiente:

```
lpoints = regtocoordenates(getregressionline(lval), getmaxpoint(lval[ycolumn]))
```

Finalmente se añadirán los elementos externos que haya seleccionado el usuario cuyos valores vienen determinados por los parámetros: *xlabel* e *ylabel* para las etiquetas de los ejes, *name* para el texto de la leyenda y *title* para el del titulo del diagrama. Hay que destacar que en este caso la leyenda estará situada en la esquina inferior izquierda para una mejor representación de los datos. Asimismo las etiquetas de los ejes serán representadas por simples objetos de la clase *Text* que serán mostrados en la parte central de cada eje.

3.7.3. Prueba del incremento

Para probar este nuevo incremento se ha incluido una nueva ramificación en el método *getprocessargs()* del módulo pySVG.py:

```
elif prefab == "scat":
   path = scatterplot.svg'
   scatterplot = svgelements.Scatterplot(lval, xorigin, yorigin, xcolumn, ycolumn,
yinc, ptsize, ptsym,, ptcolor, corr, xlabel, ylabel, name, legend, title)
   chartsvg = scatterplot.printsvg()
```

Creándose para esta ocasión el siguiente fichero de pruebas:

```
15      57
190     27
34     175
123     45
56     118
69      98
43      74
133     87
55     239
30      10
```

Y estableciéndose las siguientes opciones de entrada:

```
--prefab scat --x=1 --y=2 --ptsize=4 --ptsym=diamond --ptcolor=blue --corr
--xlbl=Xaxis --ylbl=Yaxis --name=scatgroup1 --legend --title=SCATTERPLOT scat1g.txt
```

Dando como resultado el siguiente diagrama en SVG.

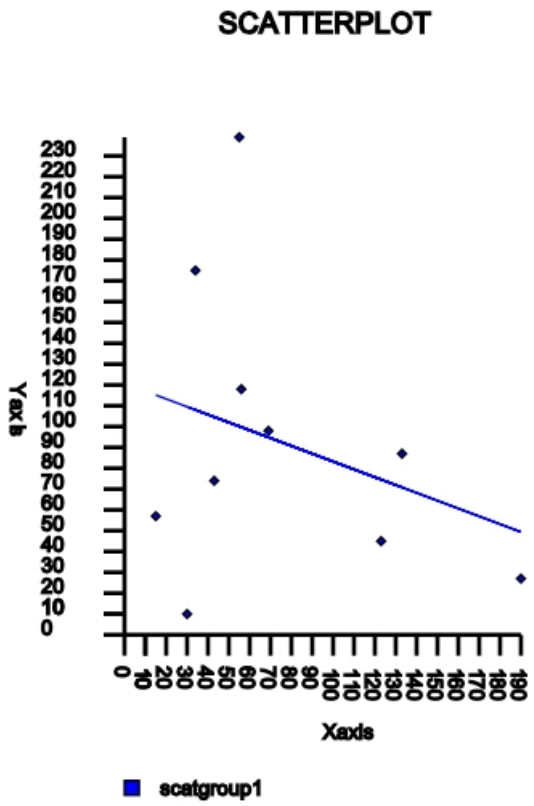

Figura 3.15: Resultado iteración 5

3.8. Iteración 6

En esta iteración vamos a mejorar el diagrama de dispersión generado en el incremento anterior.

3.8.1. Requisitos

El único requisito que se establece para esta iteración será la posibilidad de inclusión de un nuevo conjunto de datos con las mismas características que el inicial, incluyendo la recta de regresión.

3.8.2. Desarrollo

Para realizar el diseño de este nuevo incremento se ha de modificar algunos de los procedimientos de la clase *Scatterplot* descritos hasta ahora. El primero de ellos será el método *getmaxaxis(lval).* En este método se incluirá una nueva verificación que permitirá hallar el valor máximo en el eje de ordenadas para ambos grupos de datos estableciéndose así un valor resultante que será el punto máximo del montante total de coordenadas *(xmaxaxis, ymaxaxis).*

Dentro del inicializador de la clase se incluirán los atributos estilísticos para el nuevo grupo de datos: color *(pt2color:string)*, tamaño *(pt2size:int)* y forma *(pt2sym:string)* pudiéndose representar los puntos con los mismos objetos destinados inicialmente a tal fin.

Por otro lado en el método ”printsvg” se realizará una nueva verificación dentro del bucle *for* que comprobará si existen las columnas x2 e y2 en el fichero de entrada cuyos indicadores vendrán determinados por los parámetros *xcolum2* e *ycolum2* respectivamente. En caso afirmativo se crearán de forma paralela los puntos de ambos grupos de datos dentro del mismo plano cartesiano.

Finalmente si el atributo *corr* consta presente se creará la recta de regresión para ambos grupos de forma similar a la iteración anterior.

3.8.3. Prueba del incremento

Para probar este incremento se han añadido los parámetros correspondientes en la llamada a la clase *Scatterplot* desde el módulo principal quedando del siguiente modo:

```
scatterplot = svgelements.Scatterplot(lval, xorigin, yorigin, xcolumn, ycolumn,
xcolumn2, ycolumn2, yinc, ptsize, ptsym, pt2sym, ptcolor, pt2color, corr, xlabel,
ylabel, name, name2, legend, title)
```

La información del fichero de datos de entrada estará diseminada en cuatro columnas. Las dos primeras corresponderán al primer conjunto de datos y las dos últimas al segundo.

95	57	285	65
150	27	359	67
84	175	278	90
123	45	345	50
156	118	220	178
169	98	245	80
143	74	260	56
133	87	254	88
95	239	277	11
100	10	219	123

Asimismo las nuevas especificaciones de entrada serán las siguientes:

```
--prefab scat --x=1 --y=2 --x2=3 --y2=4 --yinc=15 --ptsize=5 --ptsym=diamond
--ptcolor=blue --pt2sym=circle --pt2color=orange --corr --xlbl=Xaxis --ylbl=Yaxis
--name=scatgroup1 --name2=scatgroup2 --legend --title SCATTERPLOT scat2g.txt
```

Obteniendo como resultado:

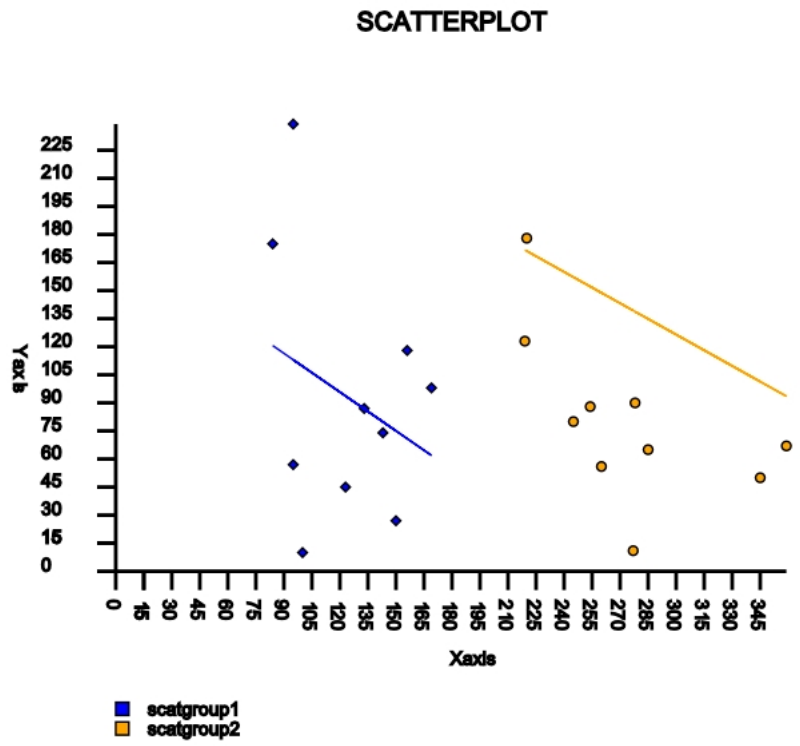

Figura 3.16: Resultado iteración 6

3.9. Iteración 7

En esta nueva iteración vamos a tomar lo aprendido hasta ahora en el diagrama de puntos para generar un diagrama de líneas que no es más que una ampliación del diagrama de dispersión con sus puntos ordenados y unidos mediante segmentos en línea recta.

3.9.1. Requisitos

El único requisito que va a establecerse en esta iteración será la admisión de dos grupos de datos y el relleno del área inferior opcional de cada línea de puntos mostrada en el gráfico.

3.9.2. Desarrollo

La clase *Lineplot* va a ser desarrollada como una clase derivada de *Scatterplot* ya que compartirá la mayoría de los métodos existentes en la misma.

El inicializador de esta clase contendrá una llamada al inicializador de la clase *Scatterplot*. Este método sera el encargado de almacenar todos los parámetros de la clase hija para poder redefinir y utilizar los métodos de la superclase.

Los únicos parámetros nuevos que surgirán en esta iteración serán *fillcolor* y *fillcolor2*, ambos de tipo *string* que indicarán el color que será utilizado para rellenar el área inferior de las líneas correspondientes a cada grupo de puntos.

Dentro ya del método que devolverá el SVG resultante ”printsvg” debemos tener en cuenta algunas diferencias respecto al diagrama de dispersión. La principal será la línea que enlazará los trazos correspondientes a la unión de las coordenadas de cada conjunto de puntos. Dicha línea seguirá un orden ascendente comenzando por la coordenada del conjunto de datos correspondiente cuyo valor en el eje de abscisas sea el menor e irá avanzando hasta el punto máximo en dicho eje.

Para gestionar la ordenación de cada conjunto de datos y representar así esta línea de forma adecuada se ha elegido uno de los métodos de ordenación más eficientes que existen actualmente, el algoritmo de ordenación rápida o *quicksort*, siendo el método que menor numero de comparaciones ejecuta.

La ordenación de los elementos de la lista será realizada en base a las coordenadas del eje de abscisas de todos los puntos. Se hará en un procedimiento auxiliar denominado *ordenatewithquicksort(lval, first, last)* donde se recibirán como parámetros la lista desordenada así como los valores de la primera y ultima coordenada *x* de la misma.

La filosofía de ordenación de este procedimiento es la técnica *”divide y vencerás”*. El objetivo es dividir la lista inicial en dos *sub-listas* más pequeñas divididas por un ”pivote” que será el elemento central de la misma, de manera que los elementos que queden a la izquierda del pivote sean los valores superiores y los que queden a la derecha los superiores:

```
pivot = (lval[first][xcolumn] + lval[last][xcolumn]) / 2
```

Una vez hecho esto se ordena recursivamente cada ”sublista” dividiéndola nuevamente a la mitad.

```
if first < jvar:
lval = ordenatewithquicksort(lval, first, jvar)
if last > ivar:
lval = ordenatewithquicksort(lval, ivar, last)
```

Donde *ivar* será el primer y *jvar* el último elemento de cada nueva *sub-lista* respectivamente. Cuando el tamaño de la *sub-lista* quede reducido a 1 ya no deberá existir ordenación

y por tanto la gestión de ordenación habrá finalizado devolviendo en la misma lista de entrada *lval* una nueva lista ordenada ascendentemente. Tras la aplicación de este algoritmo las listas resultantes serán almacenadas en *lpointsordenate* y *lpointsordenate2* respectivamente. En estas serán insertados posteriormente dos puntos de forma manual para cerrar correctamente el polígono que formará este elemento camino. Estos puntos serán *(Xordenate,0)* en la posición inicial de la lista y *(Xmax(lval),0)*.

Una vez representadas las líneas de los conjuntos de datos indicados en el fichero de entrada se generarán los puntos atendiendo al método *getsvgpoint()* de la superclase como se realizo en la iteración anterior y se establecerán los elementos identificativos como etiquetas, titulo ó leyenda. Esta ultima estará ilustrada de una forma un tanto peculiar.

Ya no será necesario el uso de la clase *Hcolumn* puesto que cada grupo de datos será representado mediante un segmento de igual color al del conjunto de datos correspondiente más el texto introducido en el parámetro de entrada ”name” o ”name2” dependiendo si es el primer o el segundo conjunto de puntos. La leyenda quedará situada al igual que en el diagrama de dispersión en la esquina inferior izquierda para una mejor representación de la información.

```
if legend:
   linelegend = Line(xorigin, 1.5*yorigin + ymaxaxis, xorigin + linelegendsize,
1.5*yorigin + ymaxaxis, "no", ptcolor)
   textlegend = Text(xorigin + 2*linelegendsize, 1.5*yorigin + ymaxaxis, fontsize,
name)
```

3.9.3. Prueba del incremento

Para este incremento se ha añadido una nueva bifurcación al procedimiento principal *getprocessargs()* realizándose también una modificación respecto al tratamiento de la ruta de los ficheros permitiendo que el archivo resultante con el código SVG de la gráfica quede almacenado en el mismo directorio en el cual consta el fichero con los datos de entrada.

Eso se ha gestionado utilizando dos métodos del módulo estándar de tratamiento de rutas *os.path*. Estos métodos serán *”split*” que se encargará de separar la ruta en dos partes de manera que en *”dirname*” tendremos el directorio de trabajo y en *”filename*” tendremos el nombre de nuestro fichero de entrada y el método *”join*” mediante el cual compondremos una nueva ruta con el directorio almacenado en *dirname* más el nombre de nuestro fichero de salida con extensión *.svg*.

```
import os
...
(dirname, filename) = os.path.split(inputargs)
...
elif prefab == "lines":
   path = os.path.join(str(dirname), 'lineplot.svg')
   lineplot = svgelements.Lineplot(lval, xorigin, yorigin, xcolumn, ycolumn, yinc,
fillcolor, ptsize, ptsym, ptcolor, xlabel, ylabel, xcolumn2, ycolumn2, pt2sym,
pt2color, name, name2, legend, fillcolor2, title)
   chartsvg = lineplot.printsvg()
```

El fichero de datos de entrada será el utilizado en la iteración anterior por lo que se omite este punto. En cuanto a los parámetros de entrada serán los siguientes:

```
--prefab=lines --x=1 --y=2 --x2=3 --y2=4 --yinc=15 --ptsize=4 --ptsym=diamond
--ptcolor=orange --fill=orange --pt2color=blue --fill2=blue --pt2sym=triangle --legend
--name=linegroup1 --name2=linegroup2 --title=LINEPLOT --xlbl=Xaxis --ylbl=Yaxis
```

Pudiendo omitirse el relleno del área inferior simplemente eliminando las opciones ”- -fill” y/o ”- -fill2” de línea de comandos. Realizándose así multitud de combinaciones posibles y obteniéndose los siguientes gráficos SVG respectivamente:

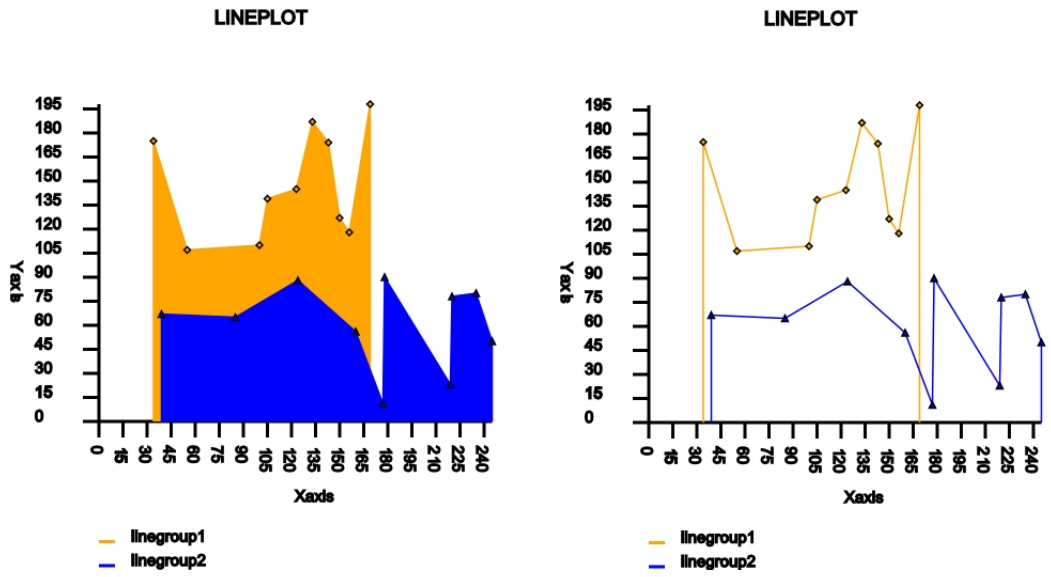

Figura 3.17: Resultados iteración 7

3.10. Iteración 8

En esta sección detallaremos ciertas mejoras estilísticas añadidas a algunos de los diagramas generados anteriormente para mejorar la calidad de su formato de salida. Precisando, estos

efectos constarán de: filtros en el diagrama de barras en tres dimensiones así como la aplicación de filtros, gradientes y animaciones en el diagrama de sectores.

3.10.1. Requisitos

Esta iteración no tendrá requisitos específicos por lo que omitimos esta sección.

3.10.2. Desarrollo

Comenzaremos describiendo como han sido desarrollados los filtros en los diagramas indicados. Cabe destacar que en muchos componentes básicos de SVG creados anteriormente como *rectangle* o *circle* aparecerán dos parámetros clave para este proceso: *filtered* y *filterid*, el primero de ellos de tipo *booleano* determinará la existencia o ausencia del elemento filtro y el segundo de tipo string describirá el tipo de filtro a aplicar sobre el elemento en cuestión.

Para entender el funcionamiento del elemento filtro vamos a explicar una serie de conceptos previos. Cuando se desea aplicar un filtro a un elemento se debe crear dentro del mismo objeto un código atributo *style* con un código similar a este:

```
style="filter: url(#filterid)”
```

Asimismo habrá que crear el código del filtro que se va a aplicar con:

```
<filter id=”filterid” filterUnits="userSpaceOnUse" x="0"
y="0" width="120%" height="120%">>
<!-- filter operations go here -->
</filter>
```

El elemento *filter* tendrá una serie de atributos que describirán la región de recorte. Se especifica un *x, y, ancho y alto* para generar el cuadro delimitador donde será aplicado el filtro. El atributo *filterUnits* tiene un valor de *objectBoundingBox* por defecto. Si se desea especificar límites en unidades de usuario, debe modificarse este valor a *userSpaceOnUse*. Los filtros funcionan por ”*primitivas”* que no son más que ordenes que indican las operaciones que se desean hacer sobre el objeto indicado. Las primitivas tendrán unas o varias entradas que podrán ser: el gráfico original *”SourceGraphic”* el canal alpha de la gráfica *”SourceAlpha”* o la salida de una primitiva anterior y exactamente una salida. Este código deberá ser incluido en la sección <*defs*> para poder ser referenciado posteriormente desde cualquier parte del documento.

Para generar todos los filtros de esta sección se ha creado una nueva clase *Filter* con la siguiente disposición:

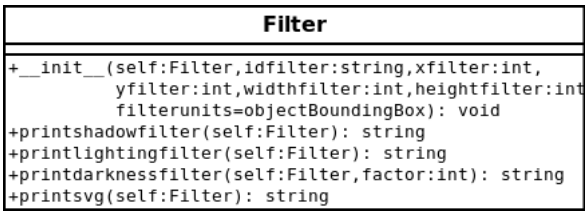

Figura 3.18: Clase Filter

Centrándonos ya en el diagrama de barras en tres dimensiones vamos a describir los cambios realizados respecto al gráfico inicial. En este diagrama va a generarse un filtro de oscuridad o ”Darkness” para aumentar la sensación de profundidad existente en el gráfico. Este filtro será aplicado sobre el objeto polígono mediante el cual se han construido el ”techo” y el lateral derecho del rectángulo en tres dimensiones.

Para incluir el filtro se ha modificado la cabecera del inicializador de la clase *Polygon* incluyendo los dos elementos ya descritos previamente: *filtered* y *filterid* y se ha modificado la estructura de salida del SVG resultante añadiendo en el atributo *style* el código indicado anteriormente en caso de que el parámetro *filtered* este presente en la línea de comandos *(filtered = True)*.

Para generar el filtro de oscuridad se ha creado un nuevo método en la clase Filter denominado *printdarknessfilter()*. Este método devolverá el código del filtro seleccionado. Para ello utilizará la primitiva ”feColorMatrix” que permitirá modificar la matriz de color del elemento disminuyendo sus valores RGB para oscurecer la zona deseada. El elemento ”factor” determinará el nivel de oscuridad aplicado.

Vamos seguidamente a describir los cambios realizados en el ”diagrama de sectores” respecto al gráfico inicial para incluir las mejoras indicadas. Describamos en primer lugar los filtros utilizados. Para esta gráfica se han seleccionado dos tipos de filtros: un filtro de sombra paralela que será aplicado en el círculo posterior de la ”tarta” a la vez que un filtro de profundidad que será aplicado en los sectores y los rectángulos de la leyenda.

Al igual que en el diagrama de barras en tres dimensiones se han modificado algunas de las clases base que lo componen para la admisión de estos elementos. Una de esas clases será la clase *Path*. En el inicializador de la misma incluiremos los parámetros: *filtered* y *filterid*. A su vez modificaremos el método *printsvg()* de manera que si el parámetro *filtered = True* se incluirá el código previamente indicado en el atributo *style*. En la clase *Hcolumn* y por definición en la clase *Rectangle* se ha efectuado la misma operación al igual que en la clase *Circle*. En todas ellas se han inicializado estos parámetros a su valor por defecto *(filtered=False, filterid=”none”)* para que la inclusión del elemento filtro no afecte a la utilización de estas clases en diagramas que no contienen dicho efecto.

Para generar el filtro de sombra paralela del círculo de la tarta se ha creado un nuevo método en la clase Filter denominado *printshadowfilter()*. Este método utilizará básicamente la primitiva ”feGaussianBlur” con una desviación lateral derecha para simular la sombra.

Finalmente para generar el filtro de profundidad en los sectores de la tarta así como en los rectángulos de la leyenda se creado otro nuevo método dentro de esta clase Filter, *printlightingfilter()*. Este método utiliza parte del filtro anterior para generar tanto la parte exterior más oscura de los sectores como de los rectángulos y aplica posteriormente la primitiva ”feSpecularLighting” en el resultado para generar una iluminación en la parte central del elemento que ofrezca así el efecto de profundidad buscado.

Una vez aplicados los filtros vamos a incluir animación al diagrama de sectores para generar una salida dinámica. La animación comenzará cuando el usuario mueva el ratón por encima de los sectores o las leyendas. Cada vez que pase por un sector determinado, éste y el rectángulo de la leyenda correspondiente a ese grupo de datos aumentará de tamaño. Paralelamente a medida que aumenta el tamaño del sector se aplicará un efecto gradiente al mismo y cambiará la forma del rectángulo de la leyenda suavizando sus bordes. Si el ratón abandona el área sobre la cual estaba situado, el elemento volverá a su estado inicial.

En este proceso intervienen dos atributos clave: *mouseover* y *mouseout*. Estos se incluirán en el elemento correspondiente para permitir la inclusión de animaciones en el mismo. Por otro lado, para controlar la aparición de la animación se establecerá un nuevo parámetro en línea de comandos denominado ”animate”. Este será un parámetro de tipo booleano que determinará la existencia o ausencia de animación en el gráfico.

```
<path id="pie" .... onmouseover="animationOn('pie');"
onmouseout="animationOff('pie');"...>
<g id="pielegend" onmouseover="animationOn('pie');"
onmouseout="animationOff('pie');">
```

Para recrear la animación se añadirá un nuevo método *getanimationscript()* dentro de la clase *Piechart* que devolverá el código del script de tipo Javascript [14] encargado de realizar la misma.

```
<script type="text/javascript">
<![CDATA[-- código de la sección-- ]]>
```

La instrucción *CDATA* se incluirá, para evitar la aparición de elementos reservados de XML. Este código *script* tendrá cuatro partes bien diferenciadas:

- La función *animationOn(id)* será la responsable de lanzar la animación cuando se pase el ratón por las zonas indicadas. Esta función se encargará de inicializar el contador de tiempo *”timevalue”* así como de aplicar el efecto gradiente al sector correspondiente a la vez que lanza las funciones que realizarán los cambios en el sector y la leyenda correspondiente al *id* indicado: *ScaleIn() y LegendScaleIn()*.

- La función *ScaleIn()* actualizará el contador de tiempo *”timevalue”* sumando el incremento determinado en ”timer_increment” y comprobando posteriormente si la animación

ha llegado a su fin *(max_time)* retornándose de la función en caso afirmativo. En caso contrario establecerá el factor de escala ”scalefactor” de 1 a 1.5 y realizará la traslación correspondiente en ambos ejes para que el elemento permanezca estático a la vez que aumente de tamaño. Por último se lanzará nuevamente la función con *setTimeout()* una vez se haya esperado el tiempo indicado por ”timer_increment”.

- La función *LegendScaleIn()* modificará los atributos del rectángulo de la leyenda ampliando su escala a 1.5 en el eje de abscisas y suavizando las esquinas mediante la modificación de las componentes ”rx” y ”ry” de este elemento. Además realizará los desplazamientos necesarios para que dicho objeto permanezca estático dando así una sensación de movimiento a izquierda y derecha respecto a los laterales del objeto. Por último modificará el atributo *”font-weigth*” del texto de la leyenda para poner este objeto en ”negrita” y el atributo *”stroke*” para rellenar el texto con el mismo color del rectángulo.

- Finalmente la función *animationOff(id)* restablecerá los valores iniciales de todos los elementos que intervienen en la animación mediante la llamada a la función *MouseOut()*. Para parar la animación se actualizará el contador de tiempo a su máximo valor: *timevalue = maxvalue.*

Para incluir dentro de la animación el elemento gradiente, que no es más que una transición de color a lo largo de una línea recta, se deberá en primer lugar crear una nueva clase denominada *”Gradient*”. Esta clase tendrá la misma estructura que la clase *Filter* y devolverá el código de dicho elemento en su método *printsvg()*. Al igual que la clase *Filter* deberá ser incluido dentro de la sección <*defs*>. La transición comenzará por el color perteneciente al sector sobre el cual se este realizando la animación y terminará en un color blanco de la siguiente manera:

```
<defs>
<línearGradient id="pieXgradient">
<stop offset="0%" style="stop-color: crimson;"/>
<stop offset="100%" style="stop-color: white;"/>
</línearGradient>"
</defs>
```

Donde X será el numero correspondiente al sector sobre el cual se esta trabajando.

3.10.3. Prueba del incremento

Para probar este incremento utilizaremos los ficheros de entrada establecidos en las iteraciones anteriores y añadiremos dos nuevos parámetros a las especificaciones en línea de comandos ”- -filtered” y ”- -animate”. El primero controlará el filtrado del gráfico y el segundo la animación.

Vamos a realizar tres pruebas para comprobar los diferentes resultados:

1. Diagrama de barras en tres dimensiones con filtro de oscuridad:

   - Fichero de entrada:

```
EUL       366
EFA        42
EDD        15
ELDR       67
EPP       276
UEN        27
Other      66
```

   - Especificaciones de entrada:

```
--prefab bardiagram3d --delim=50 --x=1 --y=2 --yrange=0 --yinc=20 --ygrid=yes
--barwidth=25 --color=pink --filtered --title=European_Parliament_Elections
--legend --name=Seats(2004) --vals vbar1g.txt
```

   - Resultado de la prueba:

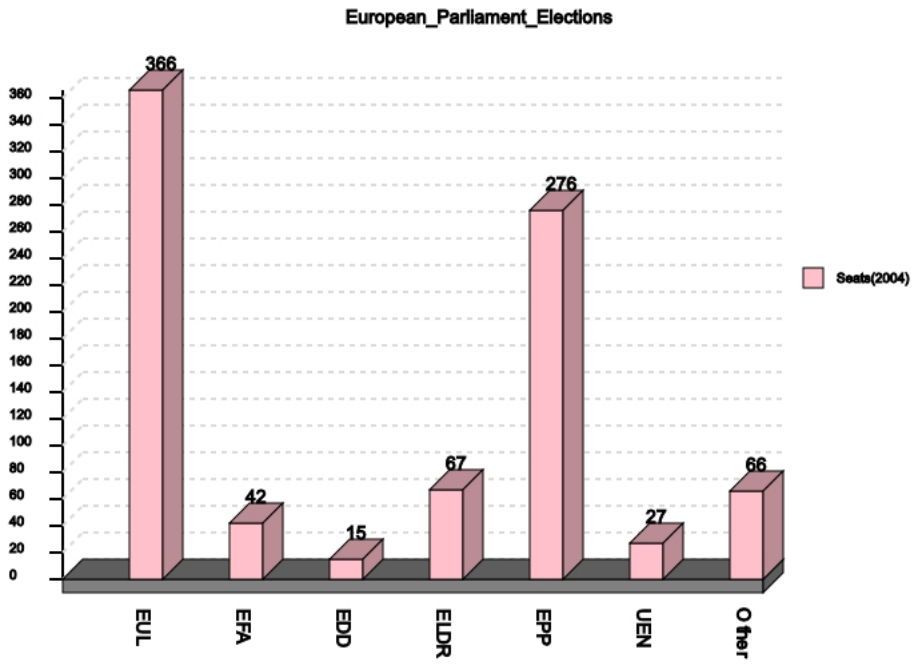

Figura 3.19: Resultado prueba 1 iteración 8

2. Diagrama de sectores:

   - Fichero de entrada:

```
EUL        39      red
PES       200      crimson
EFA        42      forestgreen
EDD        15      khaki
ELDR       67      gold
EPP       276      darkblue
UEN        27      cornflowerblue
```

- Especificaciones de entrada 1: Piechart animado no filtrado.

```
--prefab=pie --delim=50 --values=2 --labels=1 --colorfld=3 --legend --animate
--title= European_Parliament_Elections pie.txt
```

- Especificaciones de entrada 2: Piechart animado y filtrado.

```
--prefab=pie --delim=50 --values=2 --labels=1 --colorfld=3 --legend --filtered
--animate --title= European_Parliament_Elections pie.txt
```

Resultados de la pruebas con Piechart:

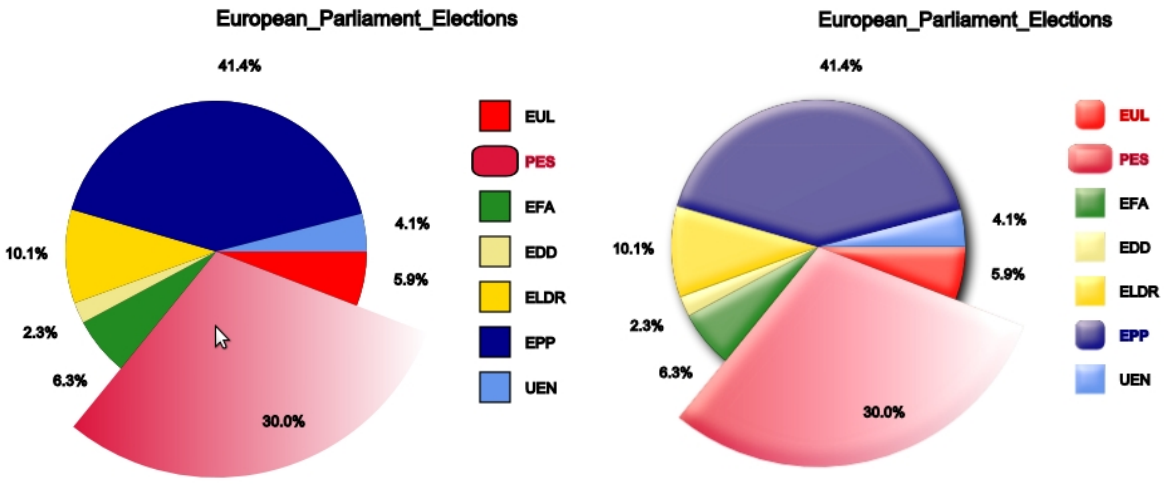

Figura 3.20: Resultados pruebas Piechart iteración 8

Capítulo 4

Fase de Pruebas

Para comprobar el éxito de nuestro proyecto se ha realizado una fase de pruebas externa con usuarios ajenos al autor dentro del módulo de docencia para proyectos de fin de carrera del departamento ”GSyC” de la Universidad Rey Juan Carlos: ”http://docencia.etsit.urjc.es/moodle”.

Esta prueba ha consistido en una fase introductoria con una serie de ejercicios destinados a que el usuario realice todo tipo de pruebas sobre los diagramas generados en nuestro programa más una encuesta donde el usuario podrá calificar la calidad del software y hacer cualquier tipo de aportación de mejora.

Dentro de la introducción a la prueba se han establecido varios apartados:

- Un apartado introductorio en el que se ha descrito brevemente las principales características del software así como el objetivo de implementación del mismo.
- Un apartado de objetivos en el que se han establecido los objetivos principales de la prueba los cuales serán descargar e instalar los recursos disponibles para la adquisición del software y realizar los ejercicios incluidos en la misma. Estos recursos se encuentran disponibles en una web creada para dicho fin ”http://www.pysvg.orgfree.com” dentro de la sección de descargas y estará disponible en varios formatos entre los cuales destacan un *Paquete Debian* que el usuario podrá instalar fácilmente mediante la orden ”dpkg” en línea de comandos. No obstante para cualquier duda se ofrece al usuario dentro de esta web una guía de instalación rápida.
- El tercer apartado contendrá la guía de usuario para el uso del software que contendrá todas las indicaciones necesarias para utilizar el programa de forma correcta.
- Finalmente en el cuarto y último apartado se expondrán una serie de ejercicios propuestos para la realización de la prueba.

Una vez el usuario haya realizado los distintos ejercicios que se plantean en esta primera fase deberá realizar la encuesta anexa para determinar la calidad final del software.

Esta fase de pruebas ha sido realizada por un usuario del grupo de investigación GSyC/LibreSoft de la Universidad Rey Juan Carlos. Hay que tener en cuenta que ha sido un usuario inexperto en

Ploticus. Del análisis del resultado de esta encuesta pueden establecerse por tanto las siguientes conclusiones:

- Previa visita a nuestra web para la descarga, instalación y utilización de nuestro software, el usuario encuestado opina que los recursos disponibles para la descarga del programa son excelentes. Por otro lado considera que el grado de dificultad respecto a la instalación es mínimo siendo nada necesarias las instrucciones de instalación existentes en la web.

- Respecto a la documentación online dicho usuario considera que los recursos disponibles representan son de fácil acceso y un buen soporte para el uso del software siendo totalmente necesarios para una buena utilización del mismo. Sin embargo considera necesario un mayor numero de ejemplos que ayuden a entender como funcionan las diferentes opciones de los gráficos.

- Centrándonos ya en la utilización del software este usuario considera que es difícil de usar debido al gran numero de opciones existentes en el mismo. Opciones que por otro lado son necesarias para establecer un software idéntico a Ploticus y mejorar sus capacidades. No obstante opina que el formato de entrada mediante un fichero de texto resulta bastante útil y que la nomenclatura de las opciones de entrada resulta bastante intuitiva sin necesidad de añadir ninguna opción más ya que las existentes engloban todas las tipologías que pueden surgir en un gráfico.

- Referente a los elementos estilísticos incluidos en las gráficas de este proyecto el usuario final considera que la calidad de los filtros y animaciones creadas ha sido buena generándose gráficos sumamente creativos.

- Finalmente respecto al formato de salida de gráficos considera que a pesar de ser algo difícil de modificar es un formato excelente. Asimismo opina que la modificación mediante la alteración de las opciones de entrada no ejerce ningún tipo de dificultad permitiendo cambiar las características del gráfico de forma sencilla y eficaz.

Cabe indicar que ha habido otro usuario en esta fase de pruebas que ha realizado la fase previa a la encuesta realizando los distintos ejercicios y cuyo analisis de conclusiones se omite debido a la no realización de la encuesta final.

Por último destacar que gracias a los usuarios implicados en esta fase se ha podido certificar que nuestra aplicación tiene una valoración satisfactoria para los miembros de la comunidad respecto a instalación, utilización y recursos disponibles.

Un matiz a tener en cuenta indicado por los usuarios que han realizado esta fase y que ha resultado clave para la versión final del código ha sido la utilización de los elementos depurativos ”Pep8” y ”Pylint” [13] gracias a los cuales se ha implementado un código más legible y con mayor consistencia.

Mediante estas dos herramientas se ha podido suplir las carencias existentes en la versión 0.0.2 del código, generando una nueva implementación atendiendo a los patrones de error determinados por estas dos aplicaciones. Al ser mecanismos muy restrictivos se han podido corregir multitud de aspectos no detectados por el intérprete de Python.

Al realizar la depuración con Pylint se ha detectado que muchos de los errores existentes provienen de los *docstrings* utilizados para la documentación del código con Epydoc por ello se ha deshabilitado la detección de dichos errores mediante la orden:

```
pylint --disable W0105 filename.py
```

Donde el código *W0105* corresponde al error *"String statement has no effect"*.

Atendiendo a todo lo anterior los cambios básicos generados en la última versión respecto a la anterior han sido principalmente:

- Modificación en los nombres de variables y métodos ajustándolos al estándar marcado por Pylint "[a-z_][a-z0-9_]{2,30}"
- Eliminación de variables no usadas como "strokewidth" o "strokecolor" en los métodos constructores del módulo *svgelements.py*.
- Eliminación de *imports* no usados como por ejemplo *"from math import *"* sustituyéndolo por las funciones esenciales usadas en el módulo svgelements.py *"from math import cos, sin, pi"*.
- Asignación de espacios entre los operadores (+, -, =, etc).
- Eliminación de líneas en blanco innecesarias *"E303 too many blank lines (2)"*.
- Modificación de nombres correspondientes a nomenclatura reservada en Python como *e, list, input* etc.
- Reducción de numero de caracteres en las líneas para no superar el margen de 80 establecido *"Line too long (125/80)"*.
- Corrección del sangrado ajustándolo al numero de espacios indicado *"Bad indentation. Found 15 spaces, expected 16* ".

Es importante resaltar que al final del reporte extraído por Pylint se puede observar una puntuación general de nuestro código. Gracias a dicha información se ha conseguido que los módulos de este proyecto alcancen las siguientes calidades respectivamente:

- Svgelements.py:

- `Versión 0.0.3, nota de calidad = 9.30`
- `Versión 0.0.2, nota de calidad = -0,63`

- Pysvg.py:
  - `Versión 0.0.3, nota de calidad = 9.50`
  - `Versión 0.0.2, nota de calidad = -2.78`

- Filetext.py
  - `Versión 0.0.3, nota de calidad = 10.00`
  - `Versión 0.0.2, nota de calidad = 2.89`

Capítulo 5

Conclusiones

Gracias a las todas pruebas realizadas, internas y externas, se ha verificado el correcto funcionamiento de esta herramienta cumpliéndose así los objetivos propuestos inicialmente en el proyecto.

Cabe destacar en este apartado las notables discrepancias ocasionadas en la visualización de los SVG resultantes respecto al navegador utilizado, siendo *Opera* el que mejor se ha adaptado a nuestras necesidades ofreciendo un resultado mucho más claro y completo. Las pruebas realizadas con el navegador *Firefox* sin embargo han resultado un tanto negativas puesto que no se admiten la mayoría de los efectos estilísticos generados mediante los filtros. Este hecho se observa claramente en el diagrama de sectores donde una vez establecidos los correspondientes parámetros de entrada se realiza la aplicación únicamente del filtro de sombra sobre el círculo posterior mientras que las animaciones sin embargo se dan correctamente. Otra diferencia considerable respecto a la salida de los gráficos en *Firefox* son los textos. *Opera* nos permite la opción de texto vertical mientras que *Firefox* no admite esta característica visualizando siempre el texto de izquierda a derecha.

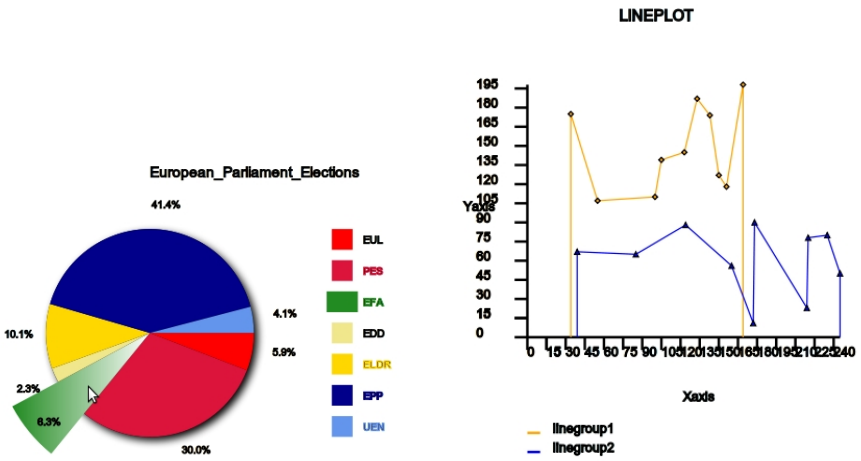

Figura 5.1: Diagramas ejecutados en Firefox

Por último matizaremos que, aunque las funciones de los módulos en este proyecto tengan un numero de parámetros algo elevado, uno de los mayores logros surgidos de la ejecución de una herramienta tan genérica, ha sido la amplia gama de posibilidades que ofrece al usuario para modelar a su gusto cualquiera de las características de un gráfico siendo así un proyecto muy útil para interpretar cualquier conjunto genérico de datos con una simple estructura en columnas dentro de un fichero de texto. Así mismo ofrece una salida mucho más definida que no depende del zoom aplicado al área visualizada.

5.1. Lecciones aprendidas

Gracias a la realización de este proyecto se han aprendido a utilizar multitud de herramientas que se desconocían anteriormente lo que ha supuesto una experiencia muy provechosa. Herramientas como ”Pylint” y ”Pep8” utilizadas para la depuración del código han sido clave para una correcta implementación del mismo. Por otro lado el estándar SVG tambien ha jugado un papel crucial para nuestro proyecto permitiéndonos generar objetos en dos y tres dimensiones compatibles con HTML.

Otra lección clave para este proceso ha sido el aprendizaje de un nuevo lenguaje de scripts como es Python mediante el cual se ha ampliado el conocimiento acerca de la programación orientada a objetos con el uso de clases dependientes para la composición de los diagramas.

Asimismo este proyecto ha permitido aprender a usar una de las herramientas más sofisticadas para la generación de documentación de los módulos de Python a través de sus *docstrings*. Esta herramienta ha sido Epydoc y su lenguaje de marcado ligero Epytext.

Respecto a la fase de publicación. La manera que se ha elegido para dar a conocer nuestro trabajo ha sido a través de un sitio web. Esta web ha sido totalmente programada en HTML permitiendo así recordar la esencia de este lenguaje y contiene todos los recursos necesarios para trabajar con el software. Esta fase se describirá con más detalle en la sección de anexos de este proyecto.

Finalmente gracias a los recursos de descarga expuestos en dicha web se ha aprendido también a construir paquetes Debian lo que ha resultado sumamente útil para la fase de pruebas y ha dado al proyecto el broche final para conseguir el éxito deseado.

5.2. Objetivos futuros

Tras la finalización de este proyecto se proponen varias vías para extender su campo de actuación haciendo de esta, una herramienta mucho más practica dentro del campo de la estadística.

Una posible innovación supondría la extensión de la librería de objetos SVG creada en el módulo *svgobjects.py* para aumentar el número de gráficas resultantes contribuyendo así a am-

pliar su funcionalidad ya que proporcionaría nuevos métodos de interpretación de variables, independientemente del carácter de las mismas. Dos posibles gráficos podrían ser el diagrama de caja y el histograma, ambos muy útiles para la representación de variables continuas, aunque también resultaría sumamente atrayente generar un mayor número de gráficos en el plano tridimensional para dar al usuario una nueva visión empírica respecto al posicionamiento espacial.

Asimismo la extensión de clases en dicho módulo podría suponer la creación de elementos descriptivos adicionales en los gráficos existentes como por ejemplo líneas de tendencia que permitan evaluar la relevancia de la información proporcionada. Por otro lado, podrían generarse nuevos efectos estilísticos mediante la adición de nuevos componentes como por ejemplo degradados radiales, o nuevas primitivas de filtro que permitan resaltar determinadas áreas de cualquiera de las gráficas.

Por último también resultaría interesante expandir las competencias de los diagramas existentes mediante la inclusión de animaciones o filtros en los diagramas que actualmente carecen de dichos efectos estilísticos para mejorar la calidad en la representación de los datos. Para ello podrán modificarse algunas de las clases existentes permitiendo así la admisión de estos elementos usando el parámetro *filtered*.

Bibliografía

[1] J. David Eisenberg. *SVG Essentials.* O'Reilly & Associates 2002
http://commons.oreilly.com/wiki/index.php/SVG_Essentials

[2] Especificación del *SVG 1.1 (Second Edition)*
http://www.w3.org/TR/SVG/

[3] Entrada de "XML" de Wikipedia en ingles.
http://en.wikipedia.org/wiki/XML

[4] Entrada de "SVG" de Wikipedia en castellano.
http://es.wikipedia.org/wiki/Scalable_Vector_Graphics

[5] Daniel Peña. *Estadística Modelos y métodos 1. Fundamentos.* Alianza Editorial, 2000

[6] Mark Pilgrim. *Dive into Python.* APRESS (SPRINGER), 2000

[7] Entrada de "Grafico Vectorial" de Wikipedia en castellano.
http://es.wikipedia.org/wiki/Grafico_vectorial

[8] Entrada de "Software libre" de Wikipedia en castellano.
http://es.wikipedia.org/wiki/Software_libre

[9] Web de documentación en la página oficial de Epydoc.
http://epydoc.sourceforge.net/epydoc.html

[10] Juan José Amor Iglesias, Israel Herraiz Tabernero, Gregorio Robles Martínez. *Desarrollo de proyectos de software libre*. UOC 2007

[11] Articulo web "casidiablo.net". C*reacción de paquetes DEB para programas en Python.*
http://casidiablo.net/debianizar-aplicacion-python/

[12] Web del proyecto Debian. *Guía del nuevo desarrollador de Debian.*
http://www.debian.org/doc/manuals/maint-guide/index.es.htm

[13] Tutorial Pylint página oficial de Logilab.
http://www.logilab.org/card/pylint_tutorial

[14] Tutorial JavaScript página oficial "w3schools".
http://www.w3schools.com/js/default.asp

[15] Sección de licencias de la página oficial de GNU.org.
http://www.gnu.org/licenses/licenses.es.html

[16] Página oficial de Ploticus.
http://ploticus.sourceforge.net/doc/welcome.html

A. Anexos

A.1 Diagramas de jerarquía de clases

En esta parte ampliaremos el detalle de la jerarquía de clases utilizada para la construcción cada diagrama. Así se entenderá fácilmente como se ha desarrollado la implementación del código para este proyecto.

Comenzando por el diagrama de barras presentamos seguidamente su jerarquía general de clases:

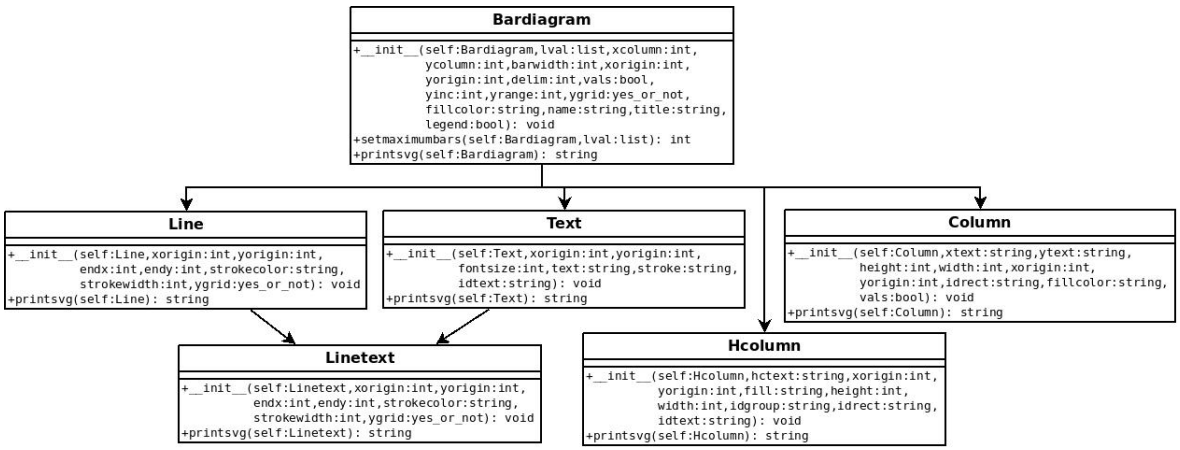

Figura 5.2: Clases del diagrama de barras

Cada elemento columna se regirá a su vez por el siguiente escalafón:

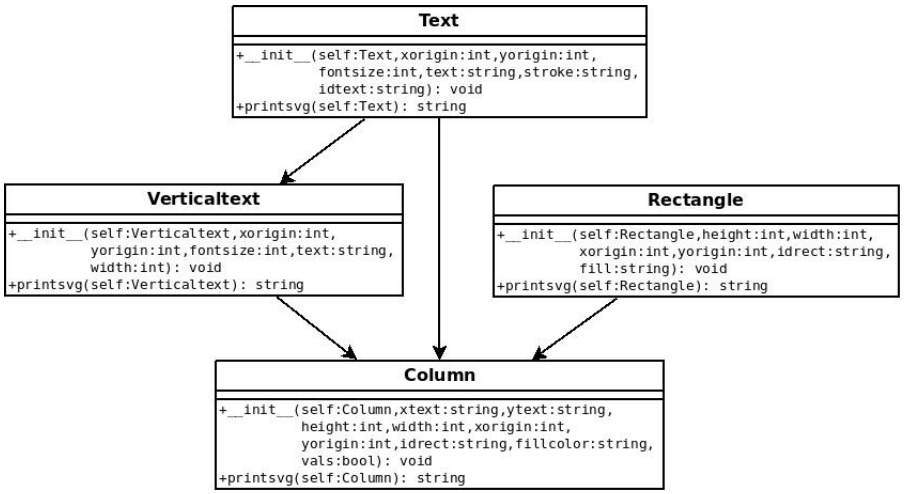

Figura 5.3: Clases del panel de barras

Hay dos elementos que coincidirán en la mayoría de los diagramas expuestos en este proyecto. En primer lugar tenemos el elemento leyenda cuyo organigrama de clases básico es el siguiente:

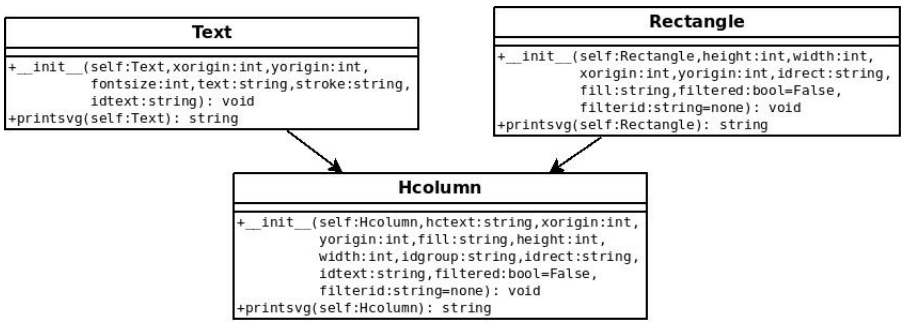

Figura 5.4: Clases de la leyenda

Pudiendo esta verse alterada mediante la variación de alguna de sus partes para dar más énfasis a la información aportada como es el caso del diagrama de líneas, en el cual la leyenda estará formada por un elemento *Line* más un elemento *Text*.

El eje de ordenadas por otro lado estará construido en todos los gráficos seleccionados en este proyecto a partir dos clases base (línea y texto) junto con una clase derivada (línea con texto).

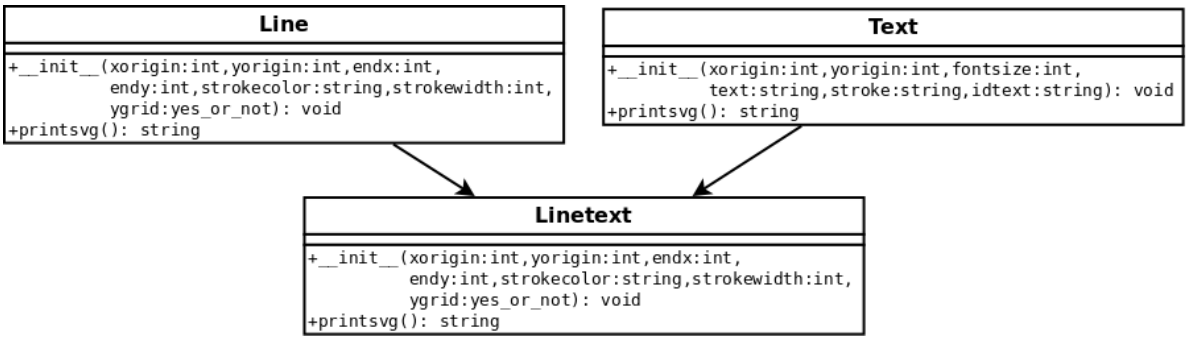

Figura 5.5: Clases del eje de ordenadas

Hablemos ahora del gráfico de barras en tres dimensiones para establecer las diferencias respecto al anterior. Si desglosamos esta gráfica se puede distinguir la siguiente jerarquía:

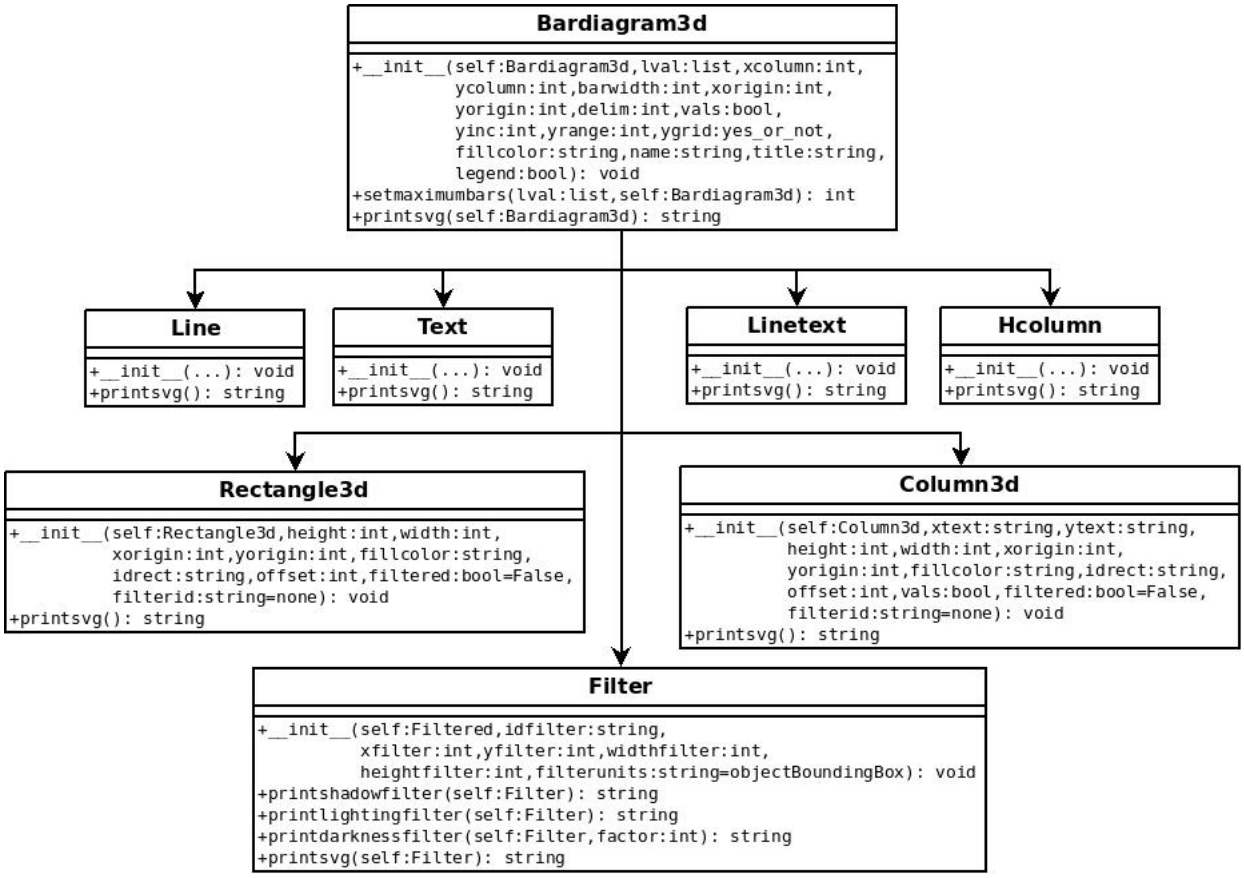

Figura 5.6: Clases Bardiagram3d

Donde al igual que en el diagrama de barras bidimensional cada columna en tres dimensiones *Column3d* tendrá su propio organigrama de clases:

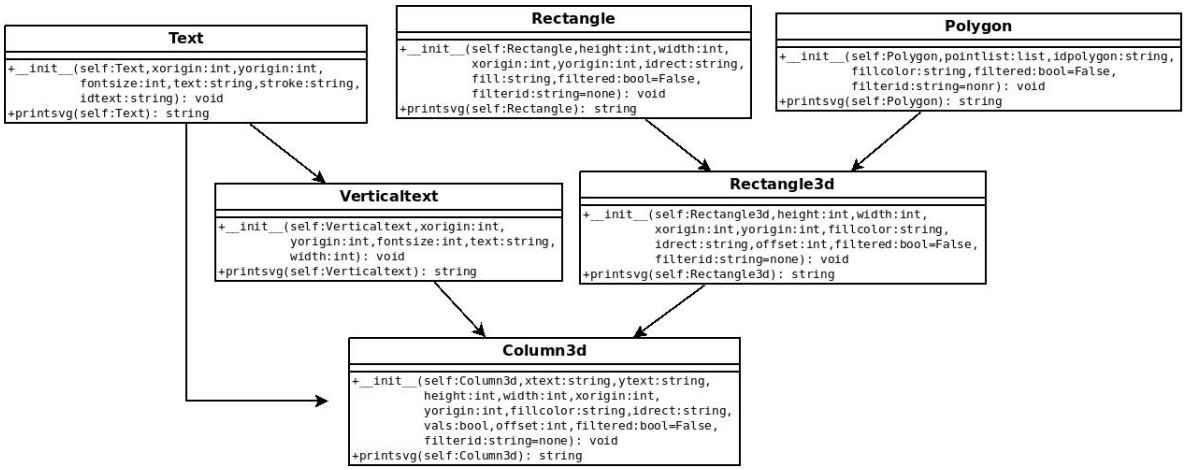

Figura 5.7: Clases Piechart

Respecto al gráfico de sectores podemos establecer la siguiente jerarquía general:

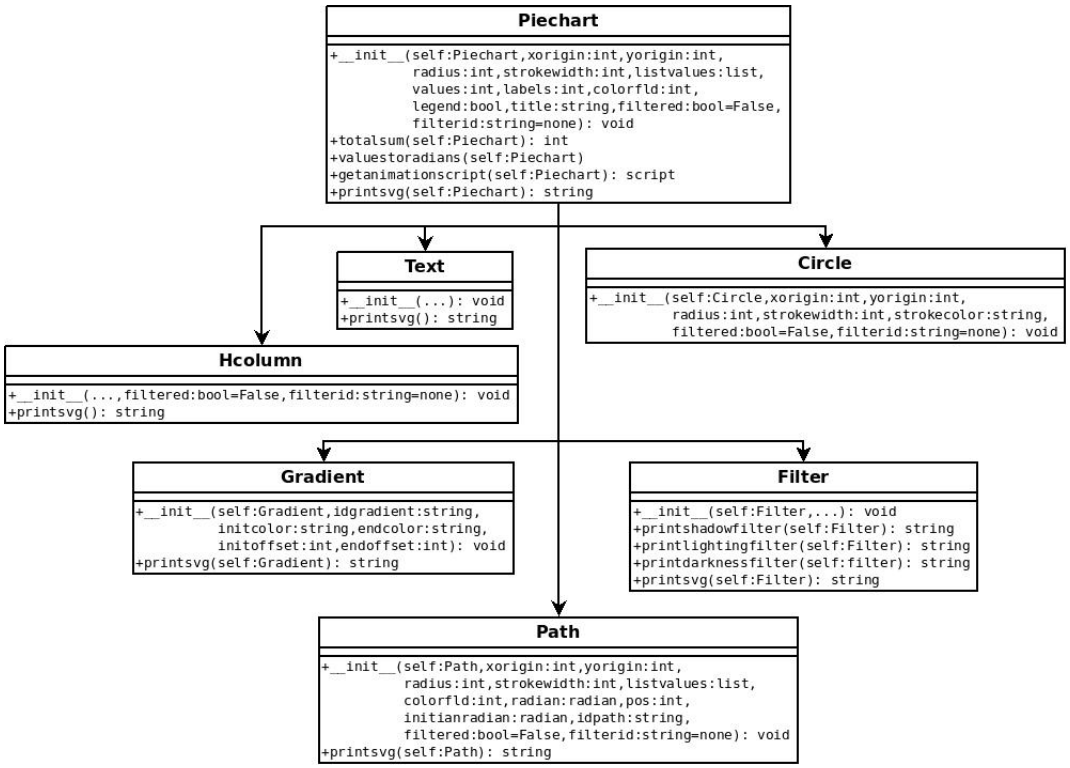

Figura 5.8: Clases Piechart

El diagrama de dispersión junto con el de líneas, constituyen dos de los diagramas que contendrán la jerarquía de clases más completa agrupando la mayor parte de los elementos básicos de SVG quedando establecido de la siguiente manera:

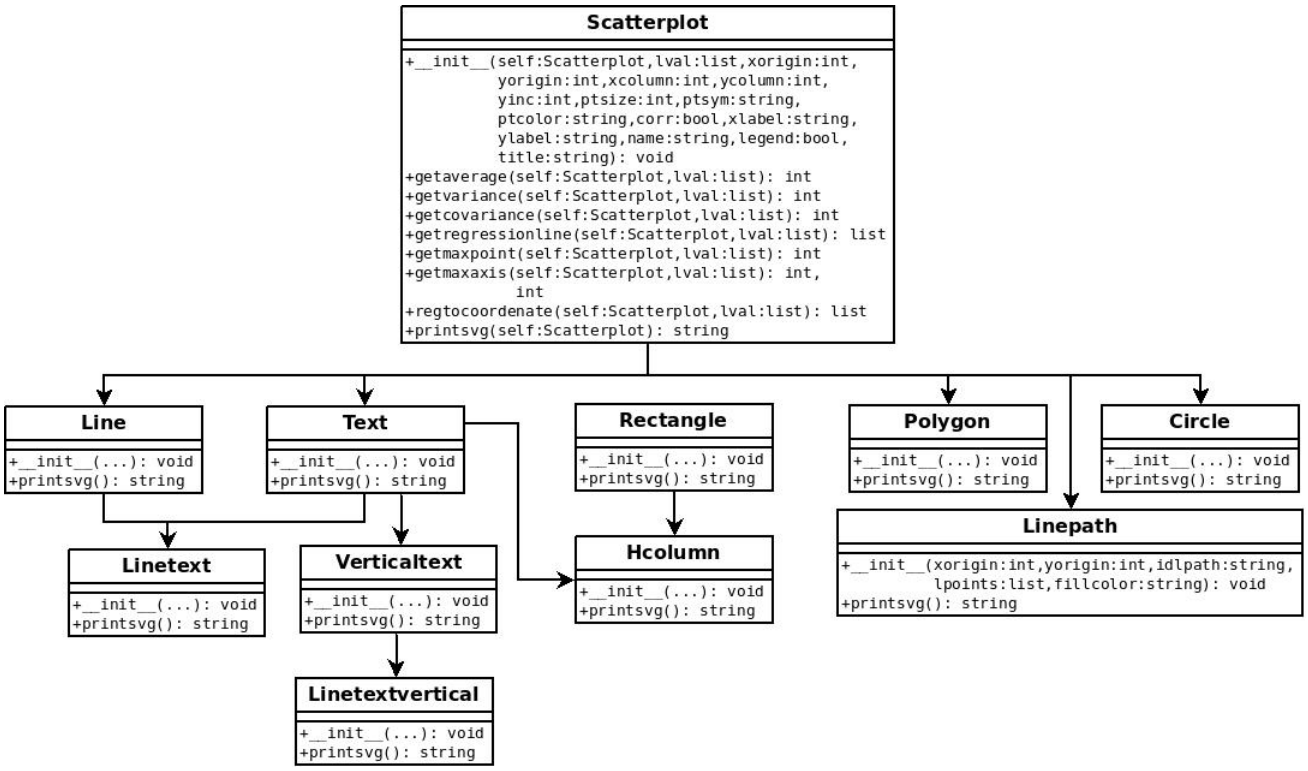

Figura 5.9: Clases Scatterplot

Se omite la estructura del diagrama de líneas ya que es una clase derivada del diagrama de dispersión y contiene los mismos componentes de cara a la jerarquía de clases.

A.2 Publicación del proyecto

Una de las fases más importantes de un proyecto para incentivar el uso del software es su etapa de publicación [10]. Esta fase permite dar a conocer nuestro producto y resaltar sus prestaciones para garantizar su éxito.

En este apartado se describirán brevemente los pasos seguidos para componer todas las herramientas necesarias que nos garanticen ese ansiado éxito. En primer lugar se ha seleccionado una licencia GPL (General Public Licence) cuyo texto reducido ha sido incluido en todos los archivos fuentes del proyecto. Su texto completo por otra parte ha sido añadido en un fichero externo denominado ”COPYNG”. Se ha seleccionado este tipo de licencia ya que resulta más robusta y nos asegura que el software generado se puede catalogar dentro de las directrices de un ”software libre”.

Para realizar la infraestructura del proyecto se ha elaborado una interfaz de usuario a través de una pagina web dentro del servidor gratuito ”orgfree.com”. La pagina con la siguiente URL: http://pysvg.orgfree.com se ha organizado en varias secciones claves cada una de las cuales aporta los recursos necesarios para la sostenibilidad del proyecto.

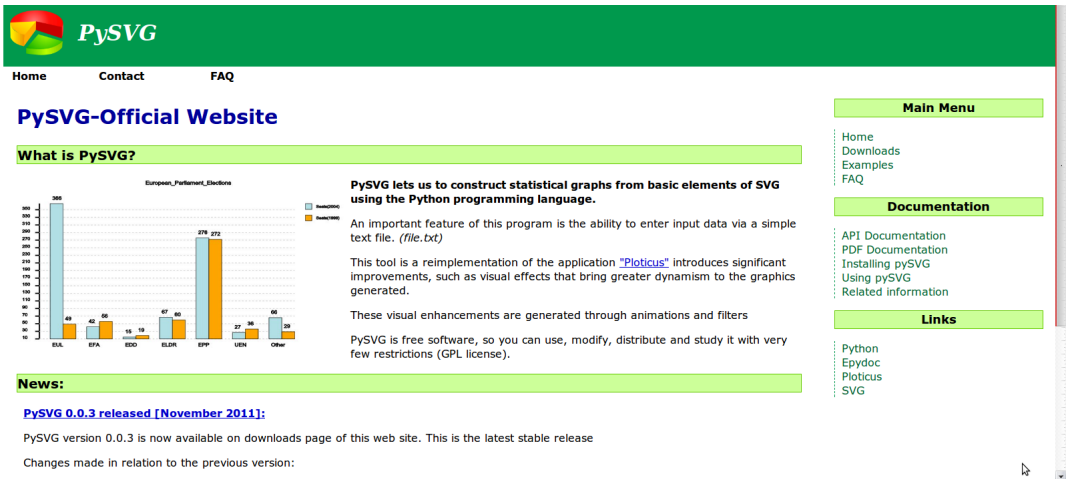

Figura 5.10: Index.html

La pagina de bienvenida ofrece al usuario una breve descripción del software así como los links necesarios para acceder a cada recurso mediante un menú lateral con múltiples opciones. También contiene una sección de novedades en la que se incluyen todas las actualizaciones generadas en el software desde el comienzo del proyecto.

Centrándonos ya en el contenido del menú de opciones podemos ver que se ha incluido una sección de ”Downloads”. En ésta el usuario podrá descargarse nuestro programa en tres formatos:

- Paquete debian mediante un archivo con extensión *.deb*
- Paquete comprimido con archivos fuente con extensión *.rar*
- Paquete precompilado con extensión *.orig.tar*

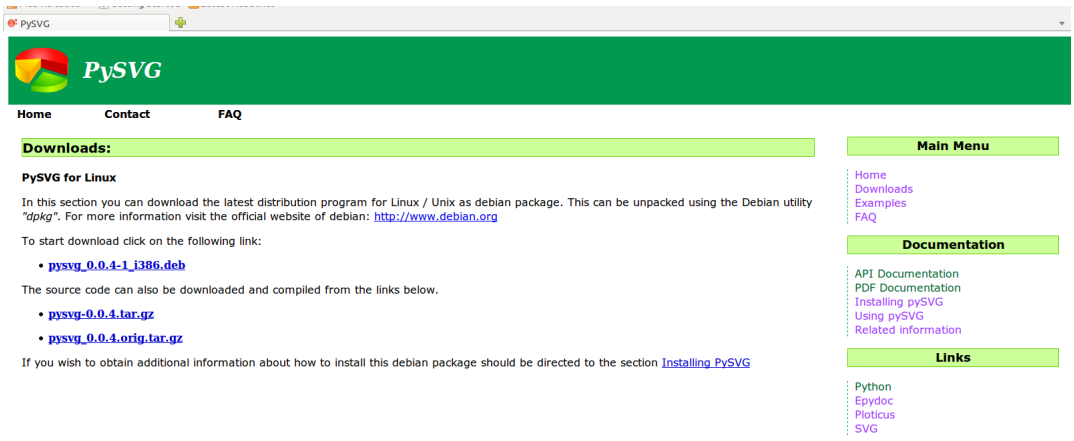

Figura 5.11: Downloads.html

La siguiente sección que detallaremos sera el apartado ”Examples”. En él se han establecido una serie de ejemplos de uso del programa para ayudar al usuario a comprender el funcionamiento del mismo así como el comportamiento de los distintos parámetros de entrada.

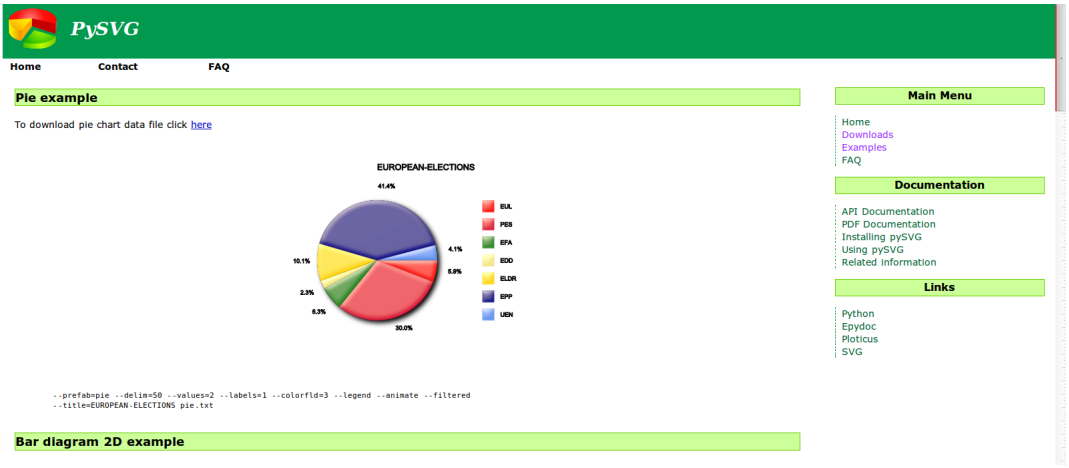

Figura 5.12: Examples.html

Una sección fundamental es la documentación. En esta web se ofrecen dos tipos de documentación: una orientada a desarrolladores profesionales que deseen colaborar en el proyecto, en formato HTML *(API Documentacion) o PDF (PDF Documentation)*

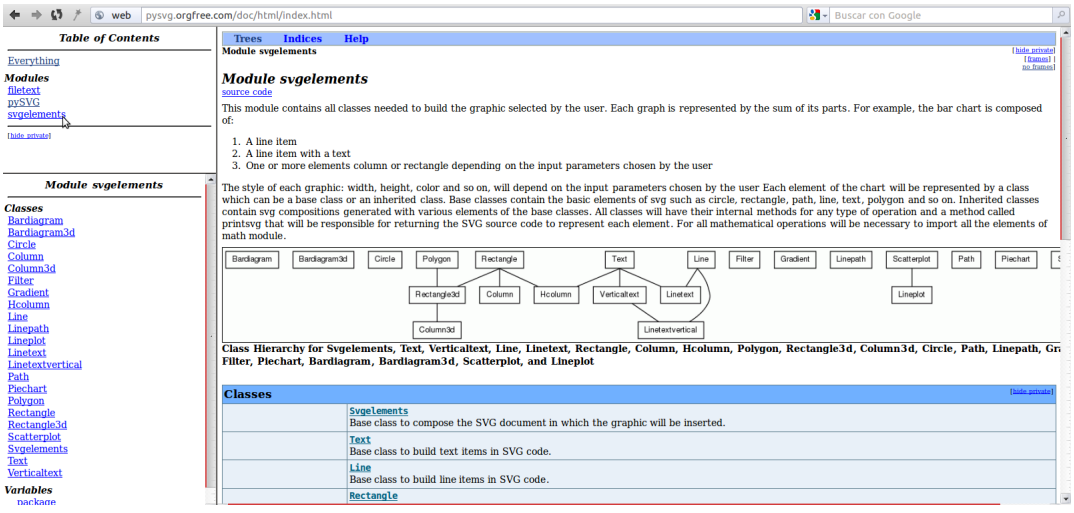

Figura 5.13: API Documentation

Y otra destinada a ser el tutorial base para cualquier usuario *(Installing pySVG y Using pySVG)*. Esta ultima proporcionará información acerca de la instalación del *Paquete Debian* así como del manejo del software detallando la utilización de cada parámetro de entrada dentro del gráfico. En el siguiente anexo de describirá más profundamente los pasos que se han seguido para la construcción de este recurso.

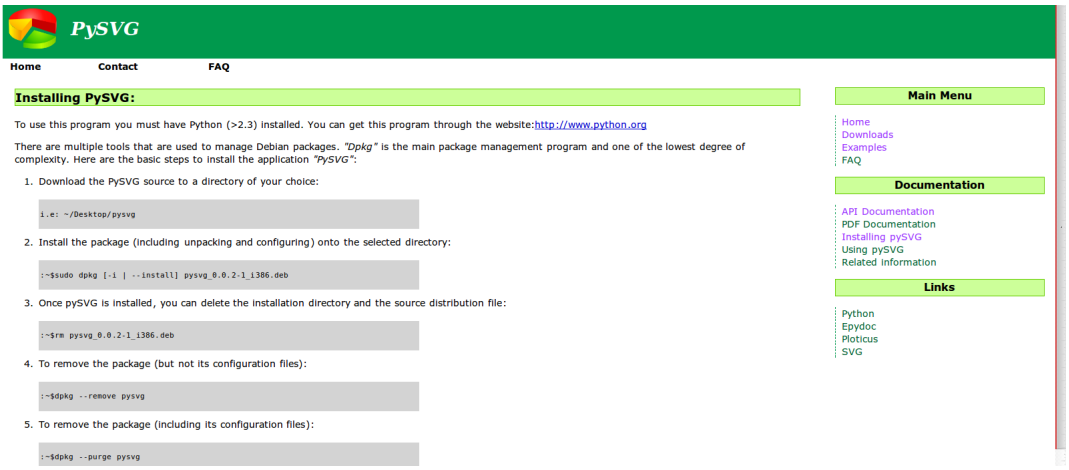

Figura 5.14: Installing pySVG.html

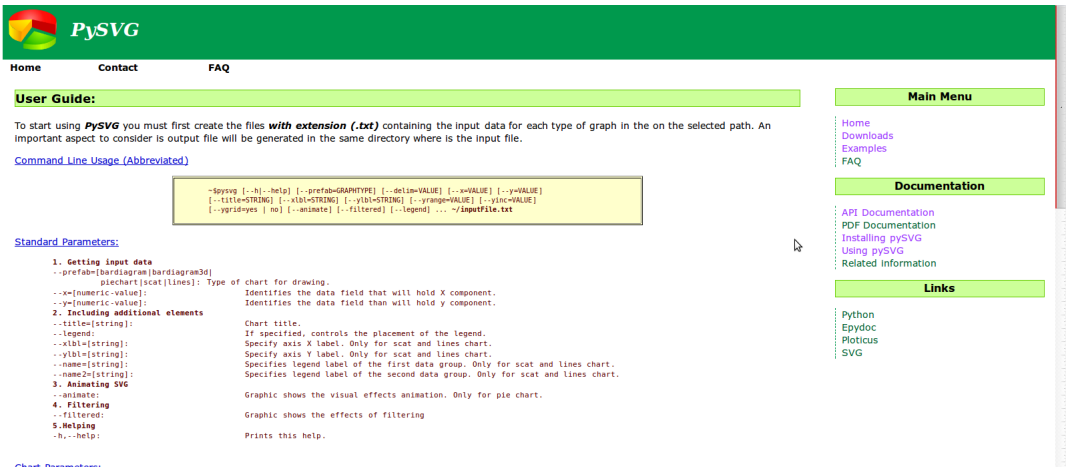

Figura 5.15: Using pySVG.html

En esta sección también se ha incluido un apartado denominado *Related Information*. Esta subsección engloba algunos componentes relacionados con nuestro proyecto como son la licencia sobre la cual se ha elaborado el software así como los cambios realizados desde la versión inicial y los futuros trabajos de ampliación que se podrían realizar.

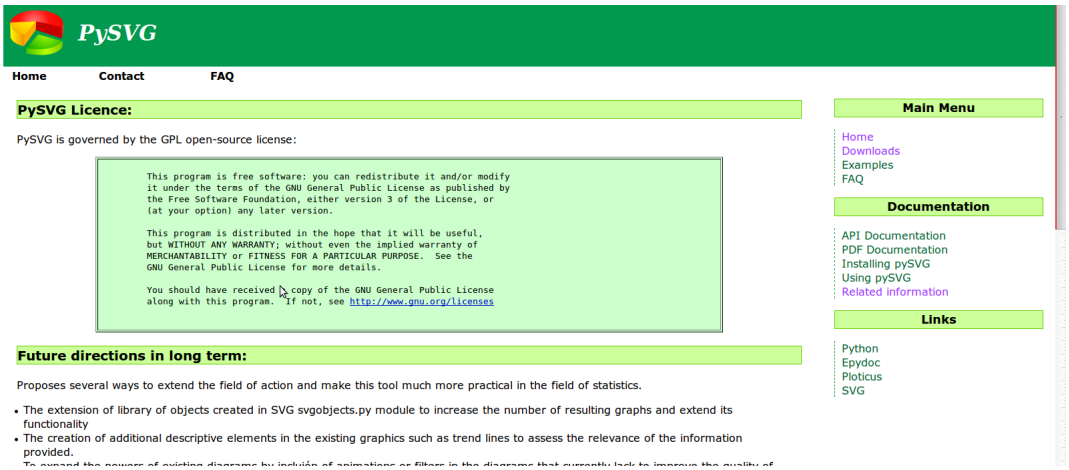

Figura 5.16: Related info.html

Para finalizar se ha establecido una pagina de preguntas frecuentes con todo tipo de preguntas que pueden surgir a la hora de ejecutar esta herramienta.

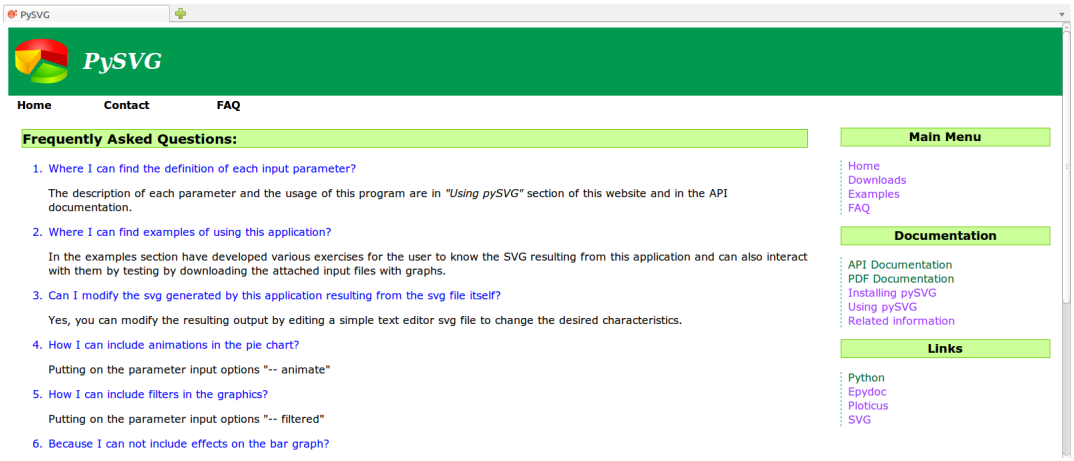

Figura 5.17: FAQ.html

A.3 Construcción del Paquete Debian

Las principales herramientas para la construcción del paquete son:

- *dh-make*: Permite tomar un paquete de código fuente estándar y convertirlo en un formato que le permitirá crear paquetes de Debian.
- *dpkg-dev*: Proporciona las herramientas de desarrollo necesarias para desempaquetar, generar y subir paquetes fuente de Debian.

Tras instalar ambos paquetes podemos comenzar con el empaquetado de nuestro programa. Para ello creamos un nuevo directorio de trabajo en el que incluimos una carpeta cuyo nombre

sea del tipo: ***nombreprograma-versión*** *(p.e. pysvg-0.0.X).* Dentro de esta, almacenamos una versión comprimida del programa en un archivo *.tar.gz.*

```
$ mkdir directorio
$ cd directorio
$ mkdir pysvg-0.0.X
$ cd ~/ruta_del_programa
$ tar cfzv pysvg-0.0.X.tar.gz
$ cp miprograma-0.1.tar.gz ~/directorio/pysvg-0.0.X

$ cd ~/directorio/pysvg-0.0.X
```

El siguiente paso es lo que llamamos debianizar el paquete. Para ello se utiliza la herramienta *dh-make,* de la cual hemos hablado previamente, con las siguientes opciones:

```
$ dh_make -e email@proveedor -f miprograma-0.1.tar.gz -c GPL
```

donde cada opción significa lo siguiente:

-e email@proveedor el correo electrónico del autor del programa

-f miprograma-0.1.tar.gz, el paquete comprimido que contiene nuestro programa

-c GPL, el tipo de licencia dentro de la cual se encuentra nuestro software.

Nos preguntará que tipo de paquete deseamos crear, seleccionamos la opción *"s" (simple)* y pulsamos "ENTER".

```
Type of package: single binary, multiple binary, library, kernel module or
cdbs? [s/m/l/k/b] s
Maintainer name : Autor_del_programa
Email-Address : xxx@mail.com
Date : Fecha_actual
Package Name : miprograma
Version : 0.X
License : gpl
Using dpatch : no
Type of Package : Single
Hit <enter> to confirm: Currently there is no top level Makefile.
This may require additional tuning. Done. Please edit the files
in the debian/ subdirectory now. You should also check that the

nombreprograma Makefiles install into $DESTDIR and not in /.
```

De esta manera, una vez haya terminado su ejecución se habrá creado una carpeta llamada "DEBIAN" y un archivo denominado *miprograma_0.1.orig.tar.gz* dentro del directorio inicial que contendrá los archivos fuentes.

Lo siguiente que debemos hacer es editar y realizar la configuración los archivos del paquete para que nuestro programa se instale de manera correcta.

- **Editando el archivo "Control":**

La sintaxis de los campos de control es muy simple: el nombre del campo, seguido por dos puntos, y luego el valor del campo: 'nombredecampo: valor' . Algunos campos requieren un valor de una palabra, otros sin embargo permiten conjuntos de valores y texto sin formato. Muchos de los campos son opcionales por lo que puede que nuestro paquete no necesite todos. Por ejemplo, un paquete no requerirá el campo 'Source' si el código fuente de ese paquete no está disponible. Si el paquete no tiene conflictos con otros, entonces no es necesario incluir el campo 'Conflict'.

Para nuestro software este archivo ha sido configurado de la siguiente manera:

```
 1 Source: pysvg
 2 Section: utility
 3 Priority: optional
 4 Maintainer: Isabel <isabel_rguez@hotmail.com>
 5 Build-Depends: debhelper (>=7)
 6 Standards-Version: 3.8.4
 7 Homepage: <http://pysvg.orgfree.com/>
 8
 9 Package: pysvg
10 Architecture: any
11 Depends: python (>=2.5.2-1ubuntu1)
12 Description: pySVG is a python program lets you convert data entered
through a text file in SVG elements to build statistical graphs.
13 It's very easy to use and can point out possible problems with your
code execution.
```

Vamos a hablar de los diferentes campos que se han incluido en este archivo. Las siete primeras líneas describen la información de control para el paquete fuente mientras que las restantes describen la del paquete binario.

- El campo '*Source*', es el nombre del paquete fuente
- El campo '*Section*' determina la sección de la distribución dentro de la cual se encuentra el paquete que en nuestro caso sera *utility.*
- El campo *'Priority'* describe el nivel de importancia que tiene la instalación del paquete. La prioridad opcional se utiliza para paquetes nuevos como el nuestro que no entran en conflicto con otros de prioridad más alta.
- El campo *'Maintainer'* establece el nombre del autor del programa (o el de su compañía), y una dirección de mail dentro de <>.
- El campo *'Buid-Depends'* incluye la lista de paquetes requeridos para construir nuestro paquete. Para todos los paquetes construidos con ''dh'' se debe incluir `debhelper (>=7)` en este campo para ajustarse a las normas de Debian respecto al objetivo `clean`.
- El campo 'Standars-Version', contiene la versión de los estándares definidos en las normas de Debian.

- El campo 'Homepage' simplemente indica la dirección URL de la pagina web del programa.

Entrando ya en la descripción del paquete binario podemos distinguir los siguientes campos:

- El campo *'Package'*, especifica el nombre del paquete binario.
- El campo *'Arquitecture'* indica que el paquete binario generado es dependiente de la arquitectura del paquete fuente.
- El campo *Depends* debería contener el nombre de cualquier paquete que sea indispensable para que nuestro paquete funcione correctamente. Hay varios campos opcionales para trabajar con dependencias. Debemos asegurarnos de listar y ordenar exactamente de que paquetes puede depender nuestro paquete antes de la instalación. Las dependencias pueden abarcar hasta 5 campos en archivo de control: *Depends, Enhances, Pre-Depends, Recommends, y Suggests.* Si existe un paquete *'recomendable*' relacionado con nuestro paquete, pero que no es necesario para que funcione, entonces deberías ser listado en el campo 'Recommends' o 'Suggests' en todo caso. Cuando los paquetes están separados por un (|) significa que cada paquete cubrirá la dependencia y por tanto no hace falta instalar ambos. El campo 'Enhances' es básicamente la declaración de que nuestro paquete puede hacer otro paquete más útil. Cuando un paquete es listado como un *'Pre-Depends'*, fuerza al sistema a asegurarse de que los paquetes nombrados ahí están completamente instalados antes de intentar instalar nuestro paquete. Finalmente los campos *'Conflicts'* y *'Replaces'* especifican paquetes que no son dependencias, pero es preferible que dichos paquetes no estén en el sistema junto con el nuestro.
- Para finalizar con este archivo las líneas 12 y 13 constituyen el campo *'Description'*. Ambas contienen la descripción corta y larga respectivamente de nuestro programa.
- **Editando el archivo rules:**

Este archivo será el encargado de realizar la instalación del programa. En primer lugar editaremos las secciones build-stamp, encargada de la construcción de los archivos compilados a partir de los archivos fuente y la sección clean encargada de eliminar los archivos innecesarios dejandolas asi:

```
build-stamp: configure-stamp
	dh_testdir touch
	build-stamp

clean:
	dh_testdir
	dh_testroot
	rm -f build-stamp

	configure-stamp dh_clean
```

Por último, modificaremos la sección install donde configuraremos el modo de instalación de nuestro programa. En nuestro caso se creará una nueva carpeta en /usr/share/pysvg donde se descargarán los archivos de nuestro software. Para ello utilizaremos la variable `${CURDIR}:` que indica el directorio actual de trabajo de la siguiente manera:

- `${CURDIR}` es equivalente a `~/pysvg-0.0.3/`, por tanto...
- `${CURDIR}/debian/`, es nuestra carpeta 'DEBIAN' generada en la debianización del paquete.
- `${CURDIR}/debian/pysvg/`, es una carpeta que aún no existe, pero que será creada, y que representa al `root (/)`.
- `cp *.py ${CURDIR}/debian/pysvg/usr/share/pysvg/` hará que, en el momento de la instalación, se copien todos los archivos con extensión .py al directorio /usr/share/pysvg/ del sistema operativo.

Quedando por tanto esta sección establecida como sigue:

```
install: build
	dh_testdir
	dh_testroot
	dh_clean -k
	dh_installdirs
	mkdir -p ${CURDIR}/debian/pysvg
	mkdir -p ${CURDIR}/debian/pysvg/usr/share/pysvg
	cp pysvg ${CURDIR}/debian/pysvg/usr/bin/
	chmod a+x ${CURDIR}/debian/pysvg/usr/bin/pysvg

	cp *.py ${CURDIR}/debian/pysvg/usr/share/pysvg
```

- **Editando el archivo dirs:**

El archivo dirs especifica los directorios que se necesitan para instalar el programa pero que por alguna razón no se crean en un proceso de instalación normal:

```
usr/bin
usr/share/pysvg

usr/share/applications
```

- **Editando el archivo copyright:**

Dentro del archivo *copyright* debemos editar la información del autor de la aplicación, la licencia del programa, y adjuntar información acerca de componentes que posean otra licencia (imágenes, sonidos, etc.). Quedando para nuestro caso algo similar a lo siguiente:

```
This work was packaged for Debian by:
Isabel <isabel_rguez@hotmail.com> on Fri, 28 Oct 2011 09:08:52 +0200
It was downloaded from:
<http://www.pysvg.orgfree.com>
Upstream Author(s):
Isabel Rodriguez <isabel_rguez@hotmail.com>
Copyright:
<Copyright (C) 2011 Isabel Rodriguez>
License:
This program is free software: you can redistribute it and/or modify it under the terms of the
GNU General Public License as published by the Free Software Foundation, either version 3 of the
License, or (at your option) any later version.
This package is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY;
without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with this program. If
not, see <http://www.gnu.org/licenses/>.
On Debian systems, the complete text of the GNU General Public License version 3 can be found in
"/usr/share/common-licenses/GPL-3".
The Debian packaging is:

Copyright (C) 2011 Isabel <isabel_rguez@hotmail.com>
```

- **Finalizando la creación del paquete:**

Una vez editados y modificados los archivos anteriores borraremos los archivos con extensión *.ex o .EX* ya que no son necesarios para nuestra configuración. Seguidamente incluiremos en el archivo *'README.Debian'* cualquier discrepancia entre el programa original y su versión debianizada, para el caso que nos compete se incluirá la información referente al contenido de nuestro paquete. Tambien incluiremos en el archivo *'changelog'* todos los cambios realizados en el codigo fuente desde la versión original del paquete hasta la versión actual.

Por último estando ya en la carpeta de nuestro programa `~/pysvg-0.0.3` ejecutaremos el siguiente comando para generar nuestro paquete finalizando asi la creación del paquete Debian *(.deb)*.

```
$ sudo dpkg-buildpackage
```