

Facultad de Matemática y Ciencia de la Computación

Universidad de La Habana



# Herramienta de graficación de datos no estructurados

Trabajo de Diploma presentado en opción al título de  
Licenciado en Ciencia de la Computación

Autor: Elias Bestard Lorigados

e.bestard@estudiantes.matcom.uh.cu

Tutor: Lic. Pedro Quinteras Rojas

La Habana, 2019

# Agradecimientos

Agradecer al colectivo de profesores de la Facultad de Matemáticas y Ciencias de la Computación, Universidad de La Habana, por recibirme en esta carrera y hacer el transito ameno. En especial al profesor Pedro Quinteras Rojas, **muchas gracias** por guiarme pacientemente durante estos meses de desarrollo del trabajo de curso y toda la ayuda brindada. A mis compañeros por el apoyo diario, las noches interminables de estudios compartiendo estos 5 años. Agradecer a mi familia por su ayuda, **paciencia** y ejemplo de constancia y superación.

**MUCHAS GRACIAS A TODOS** los que contribuyeron de una forma en el desarrollo de este trabajo y le dedicaron un minuto de su tiempo.

Muchas Gracias ☺.



La Habana, 03 de junio de 2019

## Opinión de Tutor

**Título:** *Herramienta de Graficación de Datos no Estructurados.*

**Diplomante:** Elias Bestard Lorigados

**Tutor:** Lic. Pedro Quintero Rojas.

Durante los últimos meses Elias ha venido desarrollando la tesis de diploma "*Herramienta de Graficación de Datos no Estructurados*" que constituye uno de los requisitos para convertirse en licenciado en ciencias de la computación. El trabajo de Elias consiste en una herramienta capaz de generar gráficos a partir de fuentes de datos que pueden ser completamente desconocidos. El sistema intenta entender los datos que recibe y brinda un conjunto de gráficos de salida que el usuario determina si les son útiles o no. Esta herramienta puede ser muy útil para cualquier tipo de investigador que como resultado de su investigación tenga muchos datos, en forma de tablas, excels, base de datos, etc.

Durante ese período Elias fue capaz de leer bibliografía actualizada, aprender tecnologías modernas, utilizar patrones de diseño y buenas prácticas de programación. Elias trabajó de manera muy independiente y en las mayorías de las situaciones fue capaz de resolver los problemas a los que se enfrentó con mucha destreza lo que demuestra la solidez de los conocimientos adquiridos durante la carrera. Integró conocimientos y habilidades de diferentes asignaturas y disciplinas de varios años.

Creemos que está listo para obtener su título de licenciado en ciencias de la computación y enfrentarse a la vida profesional como una persona muy capaz y con sobrados conocimientos técnicos y una elevada calidad humana.

Por todo lo antes expuesto, considerando la actualidad del trabajo realizado, su gran utilidad, propongo la calificación de excelente.

---

Lic. Pedro Rubén Quintero Rojas

# Resumen

Una investigación científica, análisis financieros, reportes de ventas, etc. generan grandes volúmenes de datos. La extracción de información a partir de ellos puede ser compleja, en especial, si se desconoce la estructura de los mismos. Por lo general, los datos son procesados manualmente y se pierde tiempo creando hojas de cálculos para analizarlos y visualizarlos mediante gráficos, que permitan interpretarlos fácilmente. El desarrollo de una aplicación que procese conjuntos de datos y los visualice, sin necesidad de que el usuario tenga gran conocimiento sobre gráficos, puede ser de gran utilidad al agilizar el proceso de análisis e interpretación de datos.

En el presente Trabajo de Diploma se brindan herramientas y la metodología para desarrollar un programa que permita: reconocer estructuras en conjuntos de datos desconocidos, procesarlos para extraer información, normalizarla y mostrar la mayor cantidad de gráficos posibles. Al usuario se le brinda una visualización gráfica de la información, facilitando su interpretación. Para esto proponemos un conjunto de módulos que interactúan entre sí. Se diseña un sistema que permite obtener un conjunto de gráficos a partir de datos y permite seleccionar los mejores, basándose en experiencias anteriores para visualizar la información. El diseño basado en módulos facilita la extensibilidad del mismo. El programa logrado da al usuario la posibilidad de adicionar su criterio sobre los gráficos generados; nos aporta una base para explotar el procesamiento de datos estructurados y no estructurados pues permite agregar nuevas capas que profundicen más en el desarrollo de esta rama.

# Abstract

Scientific research, financial analysis, sales reports, etc. generate high volumes of data. The extraction of information from them can be very complex, especially if they are unstructured. Generally, data is processed manually and time is lost creating spreadsheets to analyze and visualize them by means of charts, which allow to interpret them easier. The development of a software that allows to process data sets and visualize them, without requiring much knowledge about charts from the users, can be very useful by accelerating the data analysis and interpretation process.

In the present Research Thesis, tools and methodology are provided to develop a software that allows to: recognize structures in unknown data sets, process them to extract information, normalize it and show it in as many charts as possible. A graphic visualization of the information is presented to the user, facilitating its interpretation. For this, we propose a set of modules that interact amongst themselves. The program designed is a system that creates a collection of charts from data and facilitates the selection of the best results, based on previous experiences, for visualizing information. The module-based design facilitates its extensibility. The accomplished program gives the user the possibility to add their feedback on the generated charts. It provides a base layer to allow for better data analysis and visualization of information from structured and unstructured data, as it supports us to adds new layers that deepen the development of these branches of research.

# Tabla de Contenidos

Introducción .....	9
Datos e Información .....	9
Gráficos.....	9
Objetivo General .....	10
Objetivos Específicos .....	10
Estructura del Documento .....	11
Convenciones.....	11
Capítulo 1.    Estado del Arte .....	13
1.1    Procesamiento y Extracción de Datos.....	13
1.2    Visualización de Datos.....	15
1.3    Herramientas de Visualización y Procesamiento de Información .....	17
Epílogo.....	18
Capítulo 2.    Diseño del Programa .....	19
2.1    Módulo de Interacción con el Usuario .....	19
2.2    Módulo Identificador de Datos.....	20
2.3    Módulo Formatos Conocidos .....	22
2.4    Módulo Graficar (Generación de Gráficos).....	23
2.5    Retroalimentación .....	25
Epílogo.....	25
Capítulo 3.    Detalles de Implementación.....	27
3.1    Interacción con el Usuario.....	27
3.2    Identificador de Datos .....	28
3.3    Formatos Conocidos (KF) .....	30
3.4    Graficar.....	31
3.5    Retroalimentación .....	35
3.6    Estructura .....	36

Ejemplo .....	37
Epílogo.....	39
Capítulo 4. Evaluación del Programa.....	40
Conclusiones .....	43
Recomendaciones .....	44
Bibliografía.....	45

# Tabla de Figuras

Figura 1. Diagrama de flujo general del programa. ....	19
Figura 2. Diagrama de flujo de la Interacción con el Usuario.....	20
Figura 3. Diagrama de flujo del Identificador de Datos (hilo principal). ....	21
Figura 4. Diagrama de flujo del Identificador de Datos (generar juegos de datos).....	21
Figura 5. Relación entre los datos y los parsers.....	22
Figura 6. Relación entre parsers y gráficos con Formatos Conocidos.....	22
Figura 7. Relaciones entre estructuras de los módulos. ....	23
Figura 8. Diagrama de flujo del módulo Graficar (hilo principal). ....	24
Figura 9. Diagrama de flujo del módulo Graficar (generar juegos de datos). ....	24
Figura 10. Diagrama de flujo del sub-módulo Selección Inteligente de Gráficos. ....	25
Figura 11. Diagrama de flujo de la Retroalimentación. ....	25
Figura 12. Relación de las estructuras que interfieren en los módulos del sistema. ....	26
Figura 13. Interacción del usuario con el CLI. ....	27
Figura 14. Ejemplo de archivo “ <b>config.ini</b> ”.....	28
Figura 15. Patrón que debe implementar un parser. ....	29
Figura 16. Clase KnownF, modelo de Formato Conocido. ....	31
Figura 17. Patrón que deben seguir los gráficos. ....	32
Figura 18. Sugerencias de gráficos acorde los mensajes que transmite [17]. ....	34
Figura 19. Fragmento de la base de datos del programa. ....	36
Figura 20. Distribución del programa. ....	36
Figura 21. Resultado de Ejemplo de gráfico de línea. ....	39
Figura 22. Base de datos. ....	40
Figura 23. Ejemplo de gráfico generado útil. ....	41
Figura 24. Ejemplo de gráfico generado no útil. ....	41



# Introducción

El uso de gráficos para visualizar información resulta de gran utilidad, facilitando el análisis y comprensión de datos. Procesar grandes conjuntos de datos, estructurados o no estructurados, puede resultar difícil, más aún, si se realiza de forma manual o no ordenada. De esta forma se corre riesgo de pérdida de información valiosa o una mala interpretación de la misma.

## Datos e Información

Entre los datos y la información existe una relación palpable. Los datos se pueden ver como la materia prima en bruto, pueden existir en cualquier forma (utilizable o no) y no tienen un significado por sí mismos [1]. Mientras que la información está constituida por datos a los cuales se les ha asignado un significado por medio de una conexión relacional, siendo datos que tienen un “valor” [2]. Los datos se transforman en información cuando son interpretados por quien los recibe.

Una investigación científica, análisis financieros, reportes de ventas, etc. generan grandes volúmenes de datos. Por lo general estos están representados en juegos de datos generados por programas, tablas con estadísticas finales, resultado del trabajo diario o de eventos de la vida cotidiana. Gran parte de esta información es archivada sin estructura alguna, de forma que su uso y procesamiento se complejiza.

En algunos casos, los datos son generados y almacenados bien estructurados, pero no transmiten de forma sencilla su contenido. Es común encontrar que la presentación de resultados obtenidos se realiza de forma manual. Esto trae la pérdida de ciertas perspectivas e información útil, que puedan ser utilizados en el futuro o por un público más especializado.

## Gráficos

El sistema visual de los seres humanos es el más sofisticado procesador de información jamás desarrollado. A través de gráficos podemos hacer un buen uso de este sistema para obtener una visión más profunda de la estructura de los datos [3]. Ejemplo de esto lo encontramos en lo sencillo que le resulta al ojo humano reconocer datos agrupados en gráficos de puntos (reconocer *clusters*). Grandes cantidades de información puede converger en gráficos, donde nuestro sistema visual puede resumir la información rápidamente y extraer resultados, además de ser capaz de enfocarse en pequeños detalles.

Los gráficos son representaciones visuales de información numérica o espacial y mayormente son más fáciles de leer e interpretar que una lista de números o tablas. Pueden tener gran

densidad de información, a veces sin perder datos. Estos permiten una rápida asimilación del resultado general.

Un mismo gráfico puede ser visto a diferentes niveles de detalle (ejemplo la impresión general, el acercamiento y ubicación exacta de varios puntos adyacentes). Los gráficos pueden mostrar fácilmente complejas relaciones a través de varias variables (en dos o tres dimensiones). Estas razones hacen que los gráficos sean una parte importante de casi todo experimento o campo de base de tesis, investigaciones, artículos científicos o presentaciones de conferencias [4].

Existen ramas de las ciencias, tanto sociales como exactas, que estudian cómo procesar y extraer información de documentos y algunas, cómo visualizar la información de la mejor manera posible. Ejemplo de estas son: el periodismo de datos, minería de datos, minería de textos, etc. El enriquecimiento de estas ramas ha dado lugar al desarrollo de diversas herramientas, en muchos casos con fines específicos y con algunas limitantes.

Disponer de un conjunto de datos que pueda brindar información valiosa y no contar con una herramienta que agilice su procesamiento y lo automatice, es un problema que enfrenta La Facultad de Matemática y Ciencia de la Computación de la Universidad de La Habana. A fin de extraer la mayor cantidad de información de datos sin necesidad de un procesamiento manual de ellos, la Facultad se ha propuesto el desarrollo de un programa que permita procesar y graficar datos. Partiendo de la creación de una aplicación de nuestra autoría, que permita su adaptación a nuestras principales necesidades.

## Objetivo General

Desarrollar un programa que permita visualizar, con la mayor cantidad de gráficos, información a partir de datos, de los cuales se pueden desconocer su estructura.

## Objetivos Específicos

1. Estudiar herramientas y técnicas en la literatura actual que permitan solucionar el problema.
2. Desarrollar un mecanismo de extracción de datos sencillo que reconozca patrones en conjuntos de datos.
3. Proponer un diseño de un sistema que permita extraer información y poder graficarla.
4. Desarrollar un mecanismo de aprendizaje sencillo para seleccionar gráficos de forma inteligente partiendo de un conjunto de datos.
5. Implementar un prototipo del modelo establecido con una interfaz de usuario amigable.

6. Evaluar a partir de conjuntos de datos reales la funcionabilidad y utilidad del programa.

## Estructura del Documento

El contenido del presente Trabajo de Diploma está organizado de la siguiente forma:

La Introducción muestra los elementos que avalan el desarrollo del trabajo de tesis, así como el problema que lo origina y se enuncian los objetivos que la misma se propone.

Capítulo 1 aborda una revisión bibliográfica sobre los temas de procesamiento, extracción de información, visualización de datos, selección de mejores gráficos; mostrando herramientas que puedan ser de utilidad para el desarrollo del presente trabajo.

Capítulo 2 muestra el diseño del programa, dando una explicación de cómo está modelado y cómo funciona cada fase.

Capítulo 3 plantea los detalles interesantes de implementación, mostrando las herramientas auxiliares y cómo fue llevada a cabo la elaboración del programa. Finaliza con los detalles de un ejemplo sencillo de una ejecución de la aplicación.





Capítulo 4. expone una evaluación del programa, mostrando su utilidad en una base de datos real, así como los criterios de un usuario real. Se analizan los resultados generales de esta.

Las Conclusiones de la Tesis de Diploma, describen el cumplimiento de los objetivos propuestos.

Las Recomendaciones presenta algunos aspectos potenciales a desarrollar en futuras investigaciones, tanto para ampliar el software propuesto como para explotarlo al máximo.

## Convenciones

En el desarrollo del documento se utilizan varios diagramas que, siguen la siguiente convención:

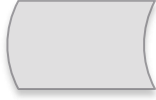
Símbolo	Significado
	Referencia a un módulo externo.
	Muestra el inicio o la entrada de datos y a su vez marca el fin del flujo.
	Representa un módulo.
	Es una etapa o fase que compone un módulo.



Decisión.



Estructura que influye en el desarrollo de un módulo (objeto).



Conjunto de Datos.



Base de Datos.

De igual forma se usan diversos formatos de letra para referirse a algunos aspectos técnicos:

**myParser**: Nombres de clases, métodos, y propiedades

**my\_library**: Aspectos técnicos de programación, rutas de archivos, nombres de archivos y librerías.

**my\_params**: Parámetros de métodos.

# Capítulo 1. Estado del Arte

Para visualizar la información contenida en documentos de texto mediante gráficos existen diversas etapas que son de vital importancia: procesar los datos, extraer información de interés y crear los gráficos. En cada etapa existen ramas de investigación dedicadas a su estudio, las cuales fueron abordadas para el desarrollo del programa, objeto de este trabajo de tesis.

## 1.1 Procesamiento y Extracción de Datos

La información puede estar contenida en varios tipos de archivos: bases de datos, documentos de textos, hojas de cálculo, etc. Estos archivos se pueden clasificar en dos tipos, de acuerdo a cómo almacenan los datos, estructurados o no estructurados.

Los archivos estructurados contienen los datos almacenados con una estructura interna, fácil de localizar, donde se conoce qué hay en cierta posición del archivo. Se puede ver como un archivador perfectamente organizado que mantiene todo identificado y etiquetado. Ejemplo de ello puede encontrarse en algunas bases de datos y archivos de textos como CSV<sup>1</sup>. Mientras que los datos no estructurados no contienen estructura interna identificable, constituyen un conjunto desorganizado de varios objetos que se almacenan sin valor alguno hasta que se identifican. A modo de ejemplo se encuentran los archivos de texto, hojas de cálculo, correos electrónicos, etc.

Cuando se parte de un documento del cual se desconoce su estructura y contenido en sí, se hace complejo poder extraer datos que sean de utilidad. En este punto es donde se busca reconocer estructuras en los datos, acomodarlos para partir de algo conocido y extraer información que resulte valiosa. Actualmente existen ramas de la inteligencia artificial y de recuperación de información que estudian como extraer datos y procesarlos, transformándolos en estructuras comprensibles posteriormente, como es el caso de la minería de datos.

La minería de datos es el proceso de obtener conocimiento nuevo a partir de grandes cantidades de datos [5]. Constituye un campo interdisciplinario, al que contribuyen muchas áreas, como son: la estadística, el aprendizaje de máquinas, recuperación de la información, reconocimiento de patrones y la bioinformática. El objetivo de la minería de datos es el descubrimiento de patrones, perfiles, anomalías y tendencias a través del análisis de los datos. Las principales técnicas en este campo son enfocadas al aprendizaje de máquinas.

---

<sup>1</sup>Archivos de datos estructurados donde los valores son separados por comas (CSV, *Comma-separated values* por sus siglas en inglés).

Entre las áreas que abarca la minería de datos se encuentra la minería de textos, en la que los datos son documentos de texto. La minería de textos se caracteriza por poseer la información muy poco estructurada y confusa, mientras que la minería de datos trabaja con bases de datos estructuradas. Dentro de la minería de datos, la minería de textos es considerada como una de las áreas de mayor potencial.

De acuerdo a lo planteado, podemos definir la minería de textos como el proceso de extracción de patrones y conocimientos de interés y no triviales a partir de textos no estructurados. Es, por tanto, un campo multidisciplinario que incluye conocimientos de recuperación de información, análisis de textos, extracción de información, *clustering*, categorización, aprendizaje de máquinas, etc. [6].

En el presente estudio resulta de interés observar los pasos en los que se descomponen estas dos ramas. La minería de datos, en general, se divide en varias fases e incluso profundiza más en la extracción de conocimiento en sí. Aunque este tema no es objeto del presente trabajo, sería interesante en un futuro profundizar en estas técnicas y extraer mayor cantidad de información con deducción de conocimiento y añadírselas al programa.

En el desarrollo de nuestro programa nos apoyamos solamente de las siguientes fases:

1. Limpiar y pre-procesar los datos, acomodándolos.
2. Clasificar los datos según sus estructuras.
3. Normalizar los datos creando una estructura intermedia que con la cual se sepa trabajar.
4. Seleccionar datos interesantes.

Tanto la minería de textos y la minería de datos ayudan a comprender el proceso para reconocer estructuras y extraer los datos que consideremos importantes para mostrarlos luego. Todo esto en su conjunto, brinda una base para definir una arquitectura sustentada en módulos, guiada por las mismas etapas descritas. Una arquitectura que finalmente permita reconocer estructuras en textos, extraer información descriptiva y mostrarla de la mejor forma posible.

Para reconocer estructuras en el texto es necesario estudiar el uso de gramáticas y expresiones regulares, con el fin de verificar la estructura de la información antes de pasar a descomponerla. Existen diversas librerías para el desarrollo de gramáticas y expresiones regulares. En esta tesis se hará uso de “**re**” y “**ply**” para definir expresiones regulares y gramáticas respectivamente.

De manera general, existen ramas de la ciencia que se dedican al estudio de reconocimiento de estructuras, extracción de datos, información e incluso llegar a extraer conocimiento en sí.

Sin llegar a tanta profundidad, atendiendo a los objetivos del presente trabajo, es preciso analizar los pasos planteados que componen la minería de datos, determinar qué tan importante es acomodar los datos, para luego identificar herramientas como gramáticas y expresiones regulares que permitan extraer datos, normalizándolos y posteriormente visualizarlos.

## 1.2 Visualización de Datos

A la hora de graficar resulta importante definir la mejor manera de mostrar la información. Los gráficos ayudan mucho a comprender los datos numéricos de un texto. La rama del periodismo de datos se centra en estudiar cómo mostrar la información y contar una historia con datos. El periodismo de datos utiliza las herramientas estadísticas y de visualización de datos con el objetivo de contar las viejas historias de otra forma, de un modo más claro para el público, y descubrir nuevas evidencias, ocultas hasta entonces [7].

Según Giannina Segnini, directora de la Maestría de las Ciencias de Periodismo de Datos de la Universidad de Columbia, Nueva York, en el periodismo de datos deben contemplarse cinco pasos básicos [8]:

1. Obtención de los datos.
2. Limpieza de los datos.
3. Análisis.
4. Verificación de la información.
5. Visualización.

Estos pasos son fundamentales a la hora de crear un gráfico. Se puede asumir que la obtención y limpieza de datos se lleva a cabo con el desarrollo de la minería de datos, fusionándose así los primeros dos pasos. Con el paso 3 se eligen, de todos los datos obtenidos, aquellos que se consideran importantes, haciéndole análisis estadísticos, de acuerdo con el mensaje que se quiera transmitir. Luego un periodista verifica la veracidad de la información, y se pasa a la visualización.

Para visualizar los datos se debe elegir entre un conjunto de gráficos. Esto puede ser complejo, pues una mala representación trae consigo una mala interpretación de los datos. Para ayudar a esta elección se categorizan los gráficos, acorde al mensaje que se quiera mostrar para luego seleccionar el tipo de gráfico que se adapte mejor a los datos.

Existen cuatro tipos de mensaje que podemos transmitir con datos: una relación, una distribución, una comparación y una composición, cada uno con un significado diferente [9] [10]:

1. Una relación muestra una conexión o una correlación entre una o varias variables que se han analizado. Por ejemplo, si colocamos la edad de los trabajadores en el eje “x” y el número de hijos en el eje “y”, podemos ver si existe una conexión entre ellos.
2. Una distribución muestra una colección de datos (relacionados o no) para comprobar si existe alguna interacción entre ellos o si están correlacionados. Por ejemplo, el número de bajas maternales de cada mes durante el año.
3. Una comparación muestra cómo una serie de variables reaccionan frente a una variable común. Por ejemplo, el número de mujeres y hombres en una empresa a lo largo de los años o el radio de rotación entre diferentes empresas.
4. Por último, una composición recoge diferentes tipos de variables que pueden englobarse dentro de un mismo umbral y las muestra todas juntas. Por ejemplo, si se realiza una encuesta entre los trabajadores y se quiere mostrar el porcentaje de respuesta de cada una de las posibles respuestas.

Para implementar un mecanismo de selección inteligente de gráficos, a partir de un conjunto de datos y el mensaje a transmitir, se estudiaron técnicas de la inteligencia artificial. Se puede analizar la clasificación de los gráficos de acuerdo con la información obtenida, con árboles de decisión, *clustering* o sistema en base a reglas (SBR). Este último se toma para el desarrollo de selección inteligente de gráficos, al no tener mucha información de la que partir para poder aplicar herramientas más profundas.

Un Sistema en Base a Reglas usa un modelo simplista basado en un conjunto de declaraciones **IF/THEN** para implementar un sistema de experto. Estos sistemas son ampliamente usados en muchos campos con el concepto principal que el conocimiento de un experto está codificado en el conjunto de reglas. Cuando el sistema de experto se encuentra un conjunto de datos, debe comportarse de la misma manera que el experto que llenó la base de datos. Los sistemas basados en reglas están en desventaja cuando la base de reglas es bastante grande, pues se torna difícil administrar y la base de reglas en sí puede consumir demasiada memoria. Los SBR generalmente están restringidos con valores discretos, lo cual se suele afrontar mediante el uso de lógica difusa [11].

Tomando como idea un SBR se pudiera implementar un conjunto de reglas para ayudar a seleccionar qué gráfico proponerle al usuario.

De manera general, hasta aquí se pueden apreciar los objetivos del periodismo de datos. Asimismo, hemos podido conocer en qué pasos nos podemos apoyar a la hora de narrar información mediante gráficos. Se mostraron las diferentes formas de clasificar los gráficos, de acuerdo con el mensaje que se desea transmitir, para luego seleccionar el gráfico que más



se adapta a los datos. Observamos técnicas de la inteligencia artificial profundizando en los SBR.

### 1.3 Herramientas de Visualización y Procesamiento de Información

En la actualidad el periodismo de datos ha alcanzado gran éxito a nivel internacional [12]. Ello ha permitido la creación de herramientas nuevas que han diversificado los usos de las que ya conocidas.

Existen diversas aplicaciones que permiten visualizar datos mediante gráficos, mapas, tablas, etc. Estas herramientas tienen fines diversos, algunas son simplemente librerías de lenguajes de programación o herramientas online. En su mayoría estos procesan la información de una base de datos o un archivo cuya estructura conocen, dígase archivo CSV o archivos Excel.

Entre los ejemplos más utilizados de aplicaciones resaltan:

1. Tableau, un software que se especializa en técnicas de visualización, enfocado en inteligencia empresarial, analiza y procesa volúmenes de datos, se enfoca en datos bien estructurados, establece un formato, bases de datos relacionales y hojas de cálculo [13].
2. SPSS una herramienta desarrollada por IBM ofrece un avanzado análisis estadístico, varios algoritmos de *machine-learning* y análisis de textos [14], muy usado en las ciencias sociales y exactas. Este puede ser utilizado con múltiples tipos de bases de datos, bases de datos relacionales y textos. En caso de recibir texto el usuario especifica de qué forma se analiza, especificando la estructura [15].
3. LightningChart, herramienta de visualización de datos de Visual Studio<sup>2</sup>.

La mayoría de estas herramientas exigen un pago por su uso, incluso muchas de ellas están enfocadas a algún campo, sea de negocios u otro.

Algunos de los programas más utilizados por los periodistas de datos son: Data Wrangler, una aplicación de la Universidad de Stanford que permite explotar hojas de cálculo; Google Fusion Tables, que convierte bases de datos en mapas y geolocaliza direcciones en ellos; Google Refine, cuyos filtros depuran la información, homogeneizando las bases de datos utilizadas; Statwing, recomendado para el análisis de datos estadísticos y cruces de variables; Data Wrapper, especialmente indicado para la creación de gráficos, y Piktochart, que también permite crear infografías [16].

Herramientas más simples permiten generar gráficos de igual manera, sin hacer mucho énfasis en la extracción de información, como Infigr.am que es una herramienta para crear

---

<sup>2</sup> Entorno de desarrollo integrado (IDLE por sus siglas en ingles) de Microsoft.

infografías, reportes, gráficos, etc., donde se puede importar los datos de archivos CSV o XLS<sup>3</sup>; Google Charts, una aplicación de Google para realizar estadística web con buen monto de gráficos; Microsoft Excel herramienta bien conocida que permite crear hojas de cálculo, genera gráficos, entre otras utilidades.

## Epílogo

Se pueden apreciar varias herramientas y métodos que se usan en la actualidad detallándolo para cada proceso. Se observa que, para procesar y extraer información de documentos, la minería de datos en general aborda estos aspectos, apoyándose principalmente en técnicas de inteligencia artificial. De igual forma para graficar, el periodismo de datos utiliza varias herramientas y ha creado la necesidad del surgimiento de nuevas aplicaciones que faciliten la creación de gráficos e infografías. Fusionando varias de estas herramientas para mostrar información. También se analizaron los pasos que deben contemplarse para mostrar buenos resultados en gráficos. Con esta base se puede diseñar una arquitectura basada en módulos para el desarrollo del software objetivo.

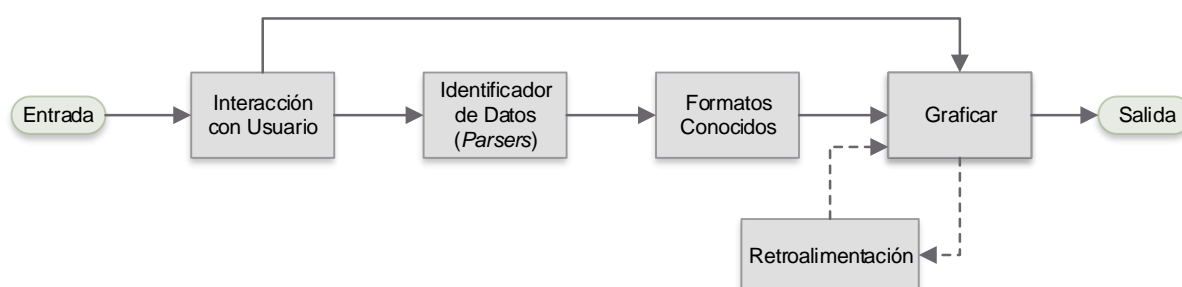
---

<sup>3</sup> Extensión de un archive por defecto de Microsoft Excel.

## Capítulo 2. Diseño del Programa

El programa consta de cinco módulos principales: el módulo de Interacción con el Usuario, el Identificador de Datos, una capa intermedia de Formatos Conocidos (KF<sup>4</sup>), el módulo de Graficar y un módulo extra para retroalimentar el proceso.

En la Figura 1 se puede apreciar el flujo general del programa:



*Figura 1. Diagrama de flujo general del programa.*

A continuación, se detallan cada uno de los módulos del programa, con las etapas que los componen.

### 2.1 Módulo de Interacción con el Usuario

Este módulo se encarga de procesar los parámetros de entrada del programa y establecer la configuración que tendrá el mismo durante todo el proceso. Para ello se descompone en las siguientes etapas (Figura 2):

**Establecer Configuración:** el módulo recibe los parámetros del usuario, estos son mezclados con valores de archivos externos, procesándolos para establecer la configuración del programa, la cual se almacena hasta el fin del proceso.

Con la configuración establecida se pasa a mostrar ayudas, generar juegos de datos, o cargar ficheros para su análisis. Sólo se realiza una opción, de las anteriores, en caso de solicitar varias, las mismas se realizarán de acuerdo con el orden descrito.

**Mostrar Ayuda:** en esta etapa el usuario tiene acceso a una ayuda del sistema. El usuario puede acceder a esta ayuda de forma explícita o en caso de tener algún error en los parámetros de entrada. Luego de mostrar las ayudas el proceso termina en este módulo.

---

<sup>4</sup> KF surge de las palabras en inglés Known Formats.

**Generar Juegos de Datos:** en esta etapa se generan gráficos o juegos de datos, en los módulos correspondientes (Gráficos o Identificador de Datos), dependiendo de la configuración. Estos juegos de datos se generan de forma aleatoria. Crea ejemplo de gráficos o archivos de datos a procesar. Al generarlos termina el proceso en el módulo correspondiente.

**Cargar Archivo a Procesar:** esta etapa es la que mantiene el hilo principal del programa. Se carga la información a procesar y se pasa al Identificador de Datos donde se continúa el flujo del programa hasta su fin.

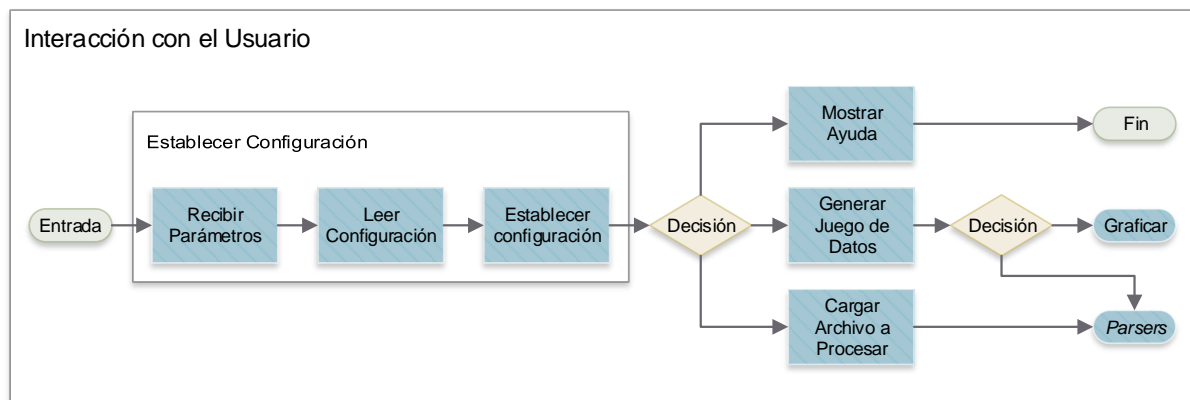


Figura 2. Diagrama de flujo de la Interacción con el Usuario.

## 2.2 Módulo Identificador de Datos

Este módulo ejecuta uno de dos procesos, puede ser accionado por la generación de datos o por el procesamiento de datos (hilo principal), Figura 3 y Figura 4.

El hilo principal del Identificador de Datos trata de reconocer estructuras en los datos, normalizándolos (con Formatos Conocidos), para graficarlas posteriormente. En la Figura 3 se ve la interacción de las etapas que lo componen.

**Pre-procesar Datos:** En esta etapa se hace un pre-procesamiento de los datos, limpiándolos y acomodándolos para facilitar la extracción de información interesante.

**Procesar Información Completa:** Se intenta procesar los datos completos. Si se logra con éxito se procede a **Crear KF** para luego pasar al próximo módulo.

**Procesar Información Fragmentada:** En caso de que la etapa anterior no fuera exitosa, se procede a procesar los datos de forma fragmentada.

**Separar Fragmentos Desconocidos:** Esta etapa separa los fragmentos desconocidos y no relevantes.

**Crear KF:** Esta etapa se alcanza con líneas que fueron reconocidas. Se procede a extraer la información valiosa y crear todos los KF posibles, por cada línea.

**Agrupar KF:** Esta última etapa, al tener un conjunto de KF, procede a compactarlos. Se trata de unir los KF similares en líneas adyacentes. Al tener ya los KF finales se procede al siguiente módulo.

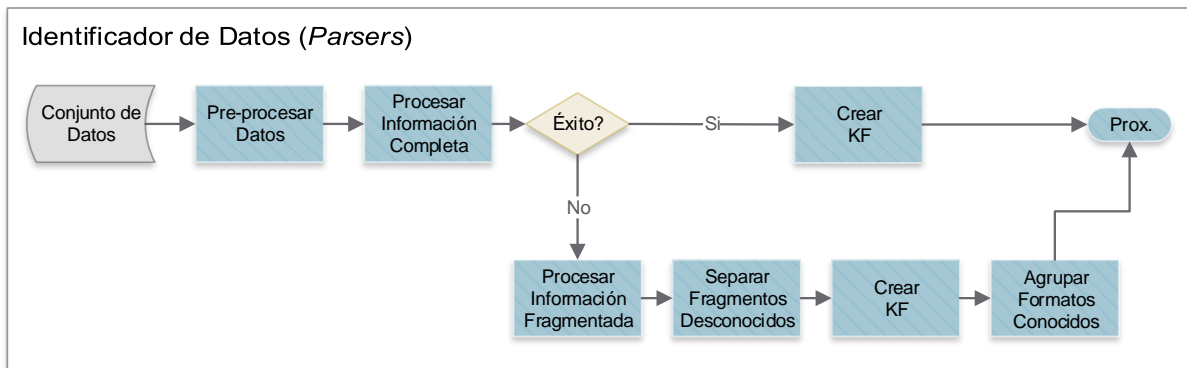


Figura 3. Diagrama de flujo del Identificador de Datos (hilo principal).

En caso de que el módulo haya sido accionado por la generación de datos, se procede a generar juegos de datos aleatorios que sirven como ejemplos de datos a procesar, Figura 4 .

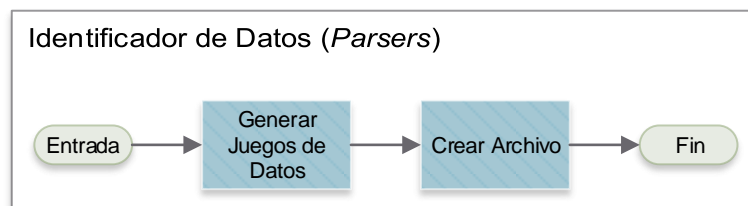


Figura 4. Diagrama de flujo del Identificador de Datos (generar juegos de datos).

Se señala que este módulo se apoya en un conjunto de *parsers* para procesar y extraer información de los datos iniciales. Un mismo conjunto de datos puede ser procesado por diversos *parsers*, Figura 5.

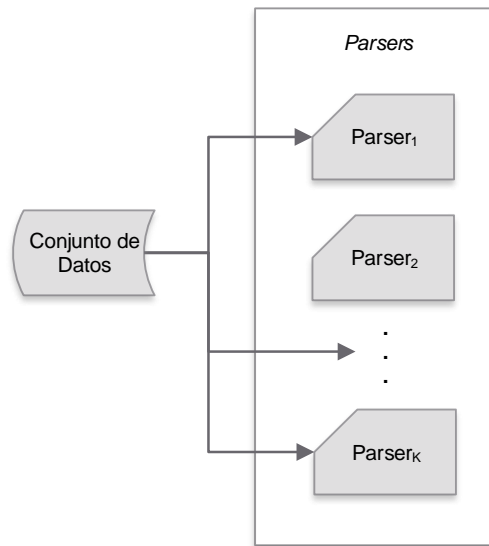


Figura 5. Relación entre los datos y los parsers.

## 2.3 Módulo Formatos Conocidos

Los Formatos Conocidos constituyen una capa intermedia. Este módulo está compuesto solamente por objetos que definimos como Formatos Conocidos, independientes de otro módulo en sí. Estos constituyen los pilares del programa.

Todos los *parsers* saben qué tipos de KF generan, y cada gráfico sabe qué KF acepta para poder graficar (Figura 6).

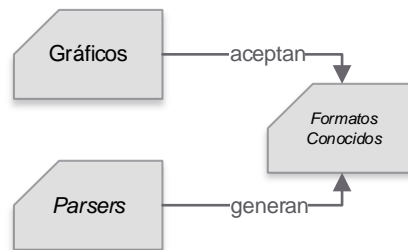


Figura 6. Relación entre parsers y gráficos con Formatos Conocidos.

Los Formatos Conocidos son estructuras que sirven como capa traductora de un módulo al otro, pues se necesita transformar la información extraída a un objeto que se conozca su estructura y se sepa trabajar. De esta forma, se definen un conjunto de KF de acuerdo a los tipos de datos diferentes que se necesiten para poder graficar la información.

Los KF están en constante relación con los *parsers* y los gráficos. Cada *parser* es capaz de generar varios KF, Figura 7 A. De igual forma, un KF es capaz de generar un conjunto de gráficos, Figura 7 B. Explicadas estas relaciones, se puede afirmar que, partiendo de un *Parser<sub>A</sub>* se pueden generar varios Gráficos.

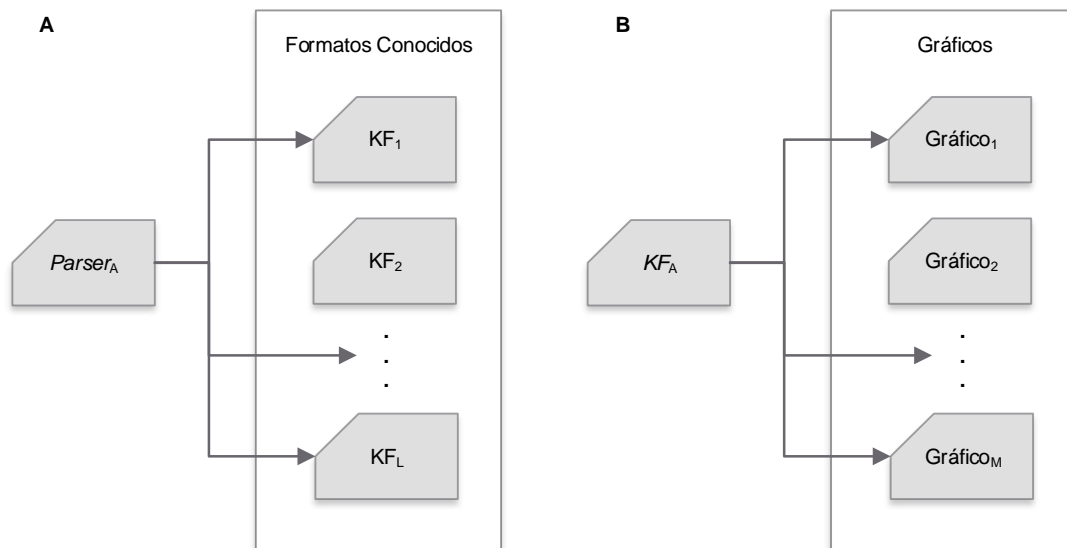


Figura 7. Relaciones entre estructuras de los módulos.

A: Relación entre un  $Parser_A$  y los Formatos Conocidos.

B: Relación entre un Formato Conocido ( $KF_A$ ) y los Gráficos.

## 2.4 Módulo Graficar (Generación de Gráficos)

A la ejecución de este módulo se llega por dos vías: puede ser accionado desde la Interacción con el Usuario o por el flujo principal del programa para graficar los KF que se han creado (hilo principal), Figura 8 y Figura 9.

El módulo Graficar está compuesto por un conjunto de gráficos definidos. Este parte de un Formato Conocido y el objetivo del módulo es crear todos los gráficos posibles, Figura 8. Este módulo contiene varias etapas:

**Buscar Gráficos Compatibles:** Esta es la etapa inicial del módulo donde se parte de un KF y se buscan los gráficos compatibles con él. Luego, en dependencia de la configuración del programa, en caso de tener activada la selección de gráficos inteligentes se procede a este sub-módulo, o a generar el archivo de salida directamente con todos los gráficos que acepten este KF.

**Selección Inteligente de Gráficos:** Está diseñada con ideas de un Sistema Basado en Reglas (SBR). En la Figura 10 se explica detalladamente su diseño y se analiza con detalle este sub-módulo.

**Generar Archivo de Salida:** Etapa final del módulo y concluye con el flujo del programa. Crea el archivo de salida con los gráficos.

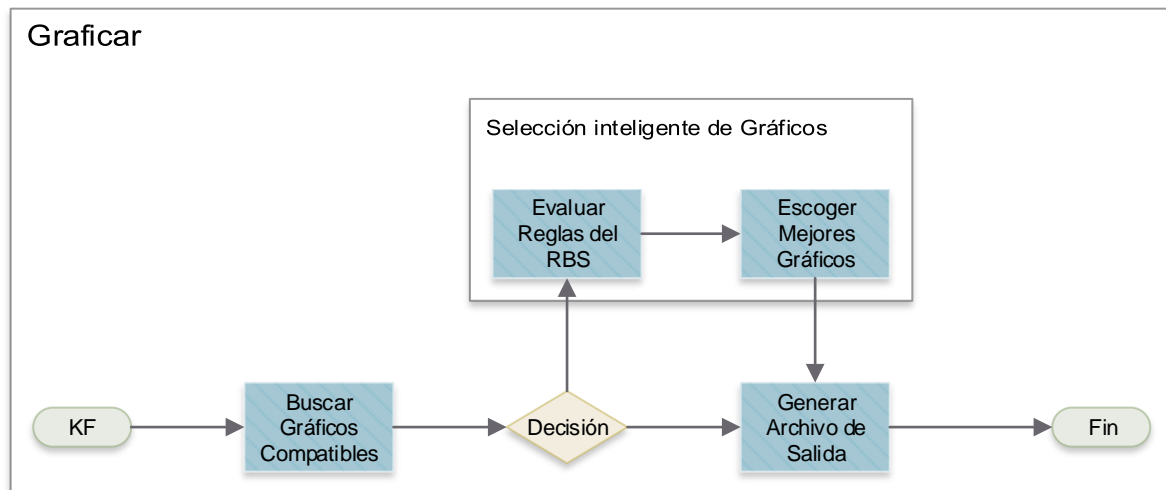


Figura 8. Diagrama de flujo del módulo Graficar (hilo principal).

En caso de que el módulo haya sido accionado por la generación de datos, se procede a generar gráficos con valores aleatorios como muestra de los gráficos implementados, Figura 9.

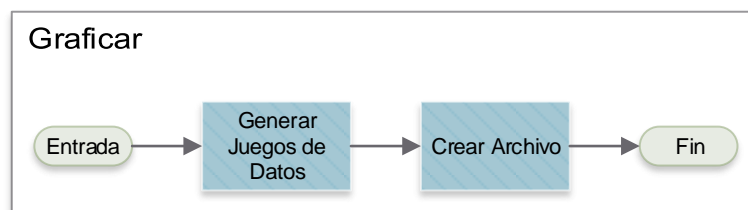


Figura 9. Diagrama de flujo del módulo Graficar (generar juegos de datos).

#### 2.4.1 Selección Inteligente de Gráficos

Las etapas de la Selección Inteligente de Gráficos son (Figura 10):

**Evalúa Reglas en Gráficos:** al conjunto de gráficos seleccionados, a partir del KF inicial, se le evalúan las reglas del SBR.

**Feedback en Gráficos:** en esta etapa se hace una búsqueda de los gráficos útiles en ejecuciones anteriores para ayudar a la selección.

**Evalúa Reglas en Mensajes:** de igual forma, se evalúan las reglas para ver qué tipo de mensaje se quiere transmitir.

**Feedback en Mensajes:** se hace una búsqueda de los gráficos útiles, por cada tipo de mensaje en particular, en ejecuciones anteriores para ayudar a la selección.

**Seleccionar Mensaje:** apoyándose de las dos fases anteriores, se selecciona el mensaje a transmitir, terminando el sistema de puntuación del SBR.



**Escoger los K-Mejores Gráficos:** se escogen los k-mejores gráficos con mejor puntuación, acorde al SBR, para mostrarlos.

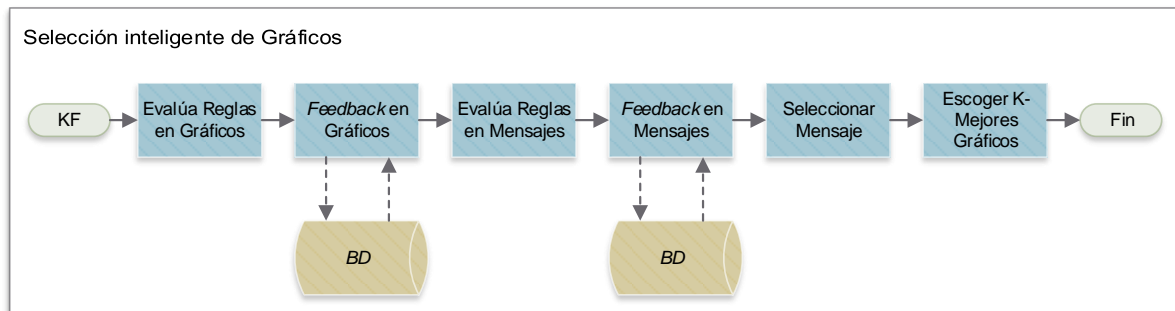


Figura 10. Diagrama de flujo del sub-módulo Selección Inteligente de Gráficos.

## 2.5 Retroalimentación

Este módulo es una pequeña aplicación que muestra un archivo generado en el proceso principal y permite al usuario seleccionar los gráficos que fueron de utilidad. Guarda esta información en una base de datos, para utilizarla en próximas ejecuciones a fin de seleccionar con mayor precisión los gráficos a mostrar, Figura 11.

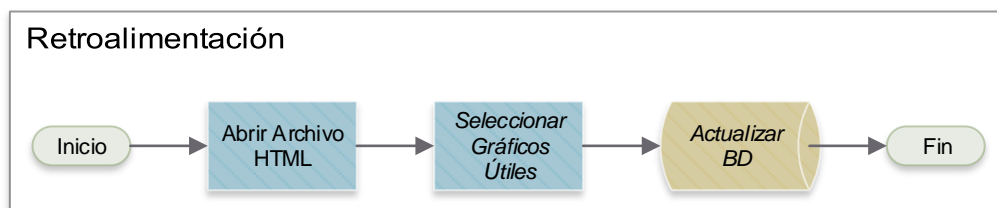


Figura 11. Diagrama de flujo de la Retroalimentación.

## Epílogo

De esta forma se observa que el flujo principal del programa se resume en:

1. Recibir los parámetros mediante la interacción con el usuario y establecer la configuración del sistema.
2. Cargar el archivo a procesar, limpiarlo y acomodarlo para su análisis.
3. Examinar y procesar el conjunto de datos completo, fragmentado o no
4. Crear Formatos Conocidos.
5. Mostrar los gráficos al usuario.

En un final se crea el fichero de salida y se cuenta con una aplicación que permite visualizar el archivo y seleccionar si es útil o no. Teniendo un diseño modular del sistema propuesto.

La Figura 12, muestra una visión general de las relaciones de las estructuras particulares que interfieren en los módulos del sistema, auxiliándose de la Figura 5 y la Figura 7.

Partiendo de un **Conjunto de Datos**, se tienen **K-Parsers** que los procesan. Este procesamiento puede ser por fragmentos del conjunto inicial. Cada *parser* genera un conjunto de KF, por lo que, de los **K-Parsers** se crean **L-KF**. De estos **L-KF** se pueden obtener **M-Gráficos**, ya que cada KF tiene un conjunto de gráficos que lo aceptan como entrada. Esto garantiza que se generen la mayor cantidad de gráficos posibles de un mismo conjunto de datos, mostrando información en diversos puntos de vista.

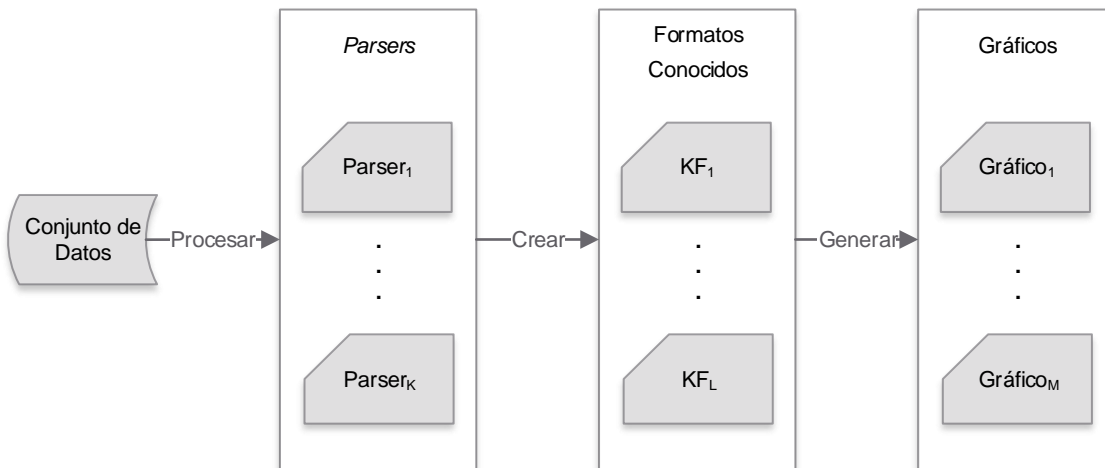


Figura 12. Relación de las estructuras que interfieren en los módulos del sistema.

## Capítulo 3. Detalles de Implementación

Una vez visto el diseño del programa, se observarán los detalles de implementación. Para esto se parte de un recorrido por los principales módulos y se muestra su estructura. Se explica cómo están distribuidos e implementados, cuáles son las herramientas de las que se auxilian y los aspectos esenciales necesarios para garantizar el funcionamiento del software. Concluyendo el capítulo con la explicación de un ejemplo de ejecución.

Para integrar los módulos del software fueron seleccionadas un conjunto de herramientas que facilitaban el desarrollo del mismo. De acuerdo con los objetivos de este trabajo de tesis se utiliza Python como lenguaje de programación.

### 3.1 Interacción con el Usuario

La interacción con el usuario es la encargada de leer los parámetros de entrada y establecer la configuración del programa, es la primera etapa.

El archivo “**input\_parser.py**” es el encargado del módulo de interacción con el usuario, el que está implementado con una interfaz de líneas de comando (CLI por sus siglas en inglés). Se apoya de la librería **argparser**<sup>5</sup> y así nuestra interfaz recibe un monto de parámetros y ofrece una ayuda al usuario. Un ejemplo de interacción con esta interfaz se ve en la Figura 13, donde el usuario manda a procesar el fichero de texto: “**.\data\_generator\prueba.txt**” con dos *parsers* específicos y solamente habilita la generación de un tipo de gráfico.

```
> python .\an.py -f .\data_generator\prueba.txt -p p_numbers_list,p_num_pair_list -g g_line_chart
```

*Figura 13. Interacción del usuario con el CLI.*

De acuerdo con los parámetros recibidos, se establece la configuración y pasa a mostrar ayuda, generar juegos de datos o analizar un archivo, acorde al pedido del usuario. Para establecer la configuración, se utiliza una clase “**Config()**”, la que inicialmente contiene los parámetros por defecto del programa y almacenará la configuración durante todo el flujo de la misma. Para establecer los parámetros de esta clase, primero se busca la existencia de un fichero “**config.ini**”, en caso de existir, se lee sus parámetros y se actualiza la clase. Luego se toman los parámetros de la entrada estándar y se sobrescriben los anteriores, estableciendo una prioridad en el proceso de ajuste de la configuración: primera prioridad son los parámetros de la entrada estándar, seguido del archivo “**config.ini**”, y en un final los parámetros definidos por defecto en “**Config()**”.

---

<sup>5</sup> Es un módulo de *parseo* por comando de línea recomendado por la librería estándar de Python [19].

El archivo “**config.ini**”, es un archivo compuesto por secciones (establecidas entre corchetes) y valores (definen parámetros y su valor), lo cual facilita la extracción de sus valores. Un ejemplo de este fichero se observa en la Figura 14, donde se establece la sección **PARSERS** y tiene parámetros “**path**”, “**available**”, “**help**”, “**list**”, con sus respectivos valores. Este archivo no tiene necesidad de existir, en ese caso se establece la configuración de acuerdo con los parámetros de la entrada estándar. No todos los valores tienen que estar presentes en el archivo.

```
[PARSERS]
path = ./api/parsers
available = p_csv
help =
list= 0
```

Figura 14. Ejemplo de archivo “**config.ini**”.

La clase “**Config()**” tiene como singularidad que implementa el patrón *singleton*, así solo se puede crear una instancia de la misma y proporciona un acceso global a esta.

## 3.2 Identificador de Datos

El Identificador de Datos es el encargado de reconocer estructuras en los datos a procesar y extraer la información que luego se graficará. Se analizará su composición, las herramientas utilizadas y una explicación del flujo del módulo, expuesto previamente en la Figura 3, haciendo hincapié en la implementación de este.

### 3.2.1 Composición

El módulo está compuesto por un conjunto de *parsers*, métodos para limpiar los datos y un conjunto de métodos para procesar la información y extraer los datos que resulten de interés. Esta fase del programa es administrada por el archivo “**an\_identify.py**”. Actualmente todos los *parsers* implementados procesan datos en cadenas de texto, resultaría interesante poder procesar diversas fuentes de datos en trabajos futuros.

Para el desarrollo del programa se implementan los *parsers* como clases del lenguaje. Los *parsers* deben cumplir ciertos requisitos para poder ser utilizados:

1. El nombre del archivo debe seguir el siguiente formato: “**p\_<nombre\_clase\_parser>**”
2. De igual forma la clase se debe llamar “**<NombreClaseParser>()**”.
3. Debe contener un método **parse(self, data\_to\_process)** que retorne una lista de Formatos Conocidos.

4. Debe contener un método `help(self)` que retorna una cadena de texto con un ejemplo de datos que procesa. Este método es necesario para mostrar ayuda al usuario.

A fin de facilitar el desarrollo de la aplicación, estos aspectos fueron estandarizados. De igual forma, las clases deben seguir este patrón, que se ejemplifica en la Figura 15. De no cumplir con estos requisitos, el archivo no se considera un *parser* válido y no se tendrá en cuenta en el curso del programa.

De manera opcional, un *parser* puede tener un método para generar juegos de datos. De existir el método se debe nombrar “`data_generator(self,path,amount,on_top,below)`” donde los parámetros de entrada y requisitos son:

1. **path**: ruta del directorio donde se creará el archivo.
2. **amount**: cantidad de series.
3. **on\_top, below**: para generar números  $X$ :  $on\_top \leq X \leq below$ .
4. Crea un archivo con los datos generados, que el *parser* reconozca.

```
class NombreClaseParser:
    def parse(self,data_to_process:str):
        pass

    def help(self):
        pass
```

*Figura 15. Patrón que debe implementar un parser.*

Este módulo se desarrolló de forma tal que pueda ser extensible fácilmente. En él se importan dinámicamente los *parsers*, que son definidos en el directorio que se estableció en la configuración. Por defecto el directorio establecido es “`./api/parsers`”, así para adicionar nuevos *parsers* sólo se debe crear el nuevo fichero “`.py`”, con el formato explicado anteriormente.

Nuevos métodos de procesar el archivo pueden ser agregados con facilidad. Luego de tener su implementación, se procedería a agregarlos al archivo principal del módulo y adicionarlo a la lista “`parsers_phases`” que se recorre para analizar los datos con cada método.

### 3.2.2 Herramientas Auxiliares

Para analizar cadenas de texto y verificar las estructuras de estas, los *parsers* definidos se auxilian de gramáticas o expresiones regulares para reconocer estructuras. Para esto se

utilizaron las librerías “ply” y la librería “re”, respectivamente. En el uso de gramáticas, “ply” trabaja con gramáticas libres del contexto de la forma BFN<sup>6</sup>.

### 3.2.3 Flujo

Nos podemos auxiliar de la Figura 3, para la explicación de esta fase del programa.

Para el funcionamiento de esta etapa son importados dinámicamente los *parsers* definidos en la ruta establecida en la configuración. Luego se pasa a limpiar la cadena, donde se eliminan las líneas en blanco, y los caracteres especiales (áéíóúüñ;/;()!@#\$\$%^&\*?><~` ). Con la cadena acomodada se recorren todos los métodos de análisis y extracción de información definidos (actualmente son dos, se tiene la idea futura de incrementar este número) y se procesa la cadena creando Formatos Conocidos.

Para procesar el texto se tienen dos métodos que se auxilian de los *parsers* definidos. Uno toma la cadena de texto completa para ver si algún *parser* habilitado la reconoce y normaliza la información. El otro método es más profundo, analiza línea a línea el texto inicial, procesa esa línea con todos los *parsers* habilitados y va creando KF por cada línea. Al terminar con el texto completo, recorre todos los KF creados y si en líneas adyacentes hay formatos conocidos similares los compacta.

De esta forma, obtuvimos un módulo fácilmente extensible, el que trata de reconocer estructuras dentro de un texto que no necesariamente sabemos cómo está conformado. Extrae los datos que reconoce y crea Formatos Conocidos.

## 3.3 Formatos Conocidos (KF)

Los Formatos Conocidos son las estructuras de datos definidas para normalizar la información, creándose por la fase anterior para poder analizar mejor los datos y poder graficarlos luego. Además, conforman la capa intermedia del programa, teniendo gran importancia en el funcionamiento del mismo.

En este acápite se muestra cómo fueron implementadas estas estructuras, y qué modelo se debe seguir para la implementación de nuevos KF.

Los Formatos Conocidos están implementados en la ruta “./api/known\_formats/”. Estos son clases que heredan de “KnownF()”, como puede observarse en la Figura 16. Están compuestos por la propiedad “elements” que almacena en un diccionario las series de los futuros gráficos y sus respectivos nombres y otras propiedades como la cantidad de series

---

<sup>6</sup> (Backus-Naur Form) es un metalenguaje usado para expresar gramáticas libres del contexto.

que almacena. Tienen un método “**extend**” que su objetivo es poder extender dos formatos conocidos iguales, agregando los elementos de uno en otro y actualizando las propiedades.

```
class KnownF:
    def __init__(self, s_values=[],series_names=[]):
        self.elements = {} #almacena las series a plotear
        self.min_value = 9999999999 #valor mínimo
        self.max_value = -9999999999 #valor máximo
        self.count = len(s_values) #cantidad de series
        self.categories = [] #nombre de las categorías
        self.keys = []
    def extend(self,kf_to_extend):
        pass
```

Figura 16. Clase KnownF, modelo de Formato Conocido.

## 3.4 Graficar

Graficar es la etapa final del programa, en esta se parte de un Formato Conocido y la intención es generar los gráficos necesarios para mostrar la información y seleccionar cuáles son los mejores gráficos acorde a los datos extraídos. En esta sección se explica la composición de esta fase, las herramientas auxiliares, el modelo presentado de selección inteligente de gráficos y el flujo (Figura 8) haciendo hincapié en la implementación del mismo.

### 3.4.1 Composición

Este módulo está compuesto por un conjunto de gráficos y un sub-módulo de selección inteligente de gráficos. Este módulo es administrado por el archivo “**an\_graphs.py**”.

Los gráficos son clases del lenguaje que se han definido. El objetivo de estas clases es apoyarse en los KF para crear el gráfico pertinente a la clase, actualmente solo genera código para crear gráficos en HTML.

Cada gráfico debe cumplir ciertos requisitos para poder ser utilizados:

1. El nombre del archivo debe seguir el formato: “**g\_<nombre\_clase\_grafico>.py**”.
2. La clase se debe llamar “**<NombreClaseGrafico>()**”.
3. Debe tener la propiedad **self.type** con el tipo de gráfico que es.
4. Debe tener la propiedad **self.message**, una lista que dice los tipos de mensajes que el gráfico transmite.
5. Debe tener la propiedad **self.kf\_permited**, una lista que dice los diferentes tipos de KF que el gráfico acepta graficar.

6. Un método `graphic(self, g_id, kf)`, que recibe el id del gráfico a generar y el KF que va a graficar. Retorna una cadena de texto con el código HTML del gráfico generado.
7. Un método `evaluate_rules(self, kf)`, que recibe un KF que va a evaluar en las reglas definidas del SBR. Retorna un número que representa cuanto se parecen los datos a graficar al gráfico actual.

Se estandarizaron estos aspectos para facilitar el desarrollo del programa. Por otro lado, las clases deben cumplir con el patrón que se puede ver en la Figura 17. Si no cumple con estos requisitos el archivo no se considera un gráfico válido y no se tiene en cuenta.

De forma opcional la clase puede tener un método para generar un gráfico con valores aleatorios, de existir el método se debe nombrar “`generate(self, id)`”, que recibe el `id` del gráfico a generar y retorna el código HTML del gráfico.

```
class NombreClaseGrafico:
    def __init__(self):
        self.type='Line'
        self.message=[]
        self.kf_permited =[]

    def graphic(self,g_id, format_known):
        pass

    def evaluate_rules(self, kf):
        pass
```

*Figura 17. Patrón que deben seguir los gráficos.*

Añadir nuevos gráficos a la estructura es bastante sencillo, solo se tiene que crear el archivo y colocarlo en el directorio del cual se están leyendo los gráficos. Este módulo importa dinámicamente los gráficos definidos en el directorio que se estableció en la configuración. Por defecto el directorio establecido es “`./api/graphs`”.

### 3.4.2 Herramientas Auxiliares

Para la implementación de este módulo nos apoyamos de la biblioteca de Java Script (JS) **Highcharts**, una librería con un amplio conjunto de gráficos y mapas, fácil de implementar.

### 3.4.3 Selección Inteligente de Gráficos

A la hora de graficar se tienen muchos gráficos donde cada uno transmite, con los mismos datos, puntos de vistas diferentes y mensajes diferentes. En este sub-módulo se plantea una



estrategia para seleccionar la mejor propuesta de gráfico para visualizar los datos, teniendo en cuenta un Sistema Basado en Reglas.

La idea general es auxiliarse de los usos de los gráficos para elegir uno de ellos de acuerdo con la forma que tienen los datos. Por este motivo se definen un conjunto de reglas (métodos) que dado un KF se evalúan en **True** o **False**, si cumple la regla o no. Utilizando esto se propone un sistema en el cual cada gráfico evalúa las reglas pertinentes de acuerdo con sus propiedades, tipo de gráfico y mensaje que transmite, en el método `"evaluate_rules()"` (Figura 17) retornando un valor numérico. Se evalúan las reglas para cada gráfico dado un KF y se asume que el mejor gráfico es el que mayor puntuación haya alcanzado.

Para complementar esta estrategia y teniendo en cuenta que los gráficos transmiten mensajes, estos se clasificaron acorde al mensaje que se quiera transmitir. Un gráfico puede transmitir más de un mensaje.

Existen cuatro tipos de mensajes que se pueden transmitir: una relación, una distribución, una comparación y una composición. De esta forma, si se sabe el mensaje que el usuario quiere transmitir, se incrementan puntos a los gráficos que transmiten este mensaje. En caso de no saberlo, se trata de intuir como mismo se hace con los gráficos, se evalúan las propiedades de cada mensaje en reglas definidas y nos quedamos con el mensaje que más puntos alcanzó.

De igual forma, el proceso se apoya de la base de datos y la interacción del usuario con esta. Se buscan todos los gráficos que fueron generados a partir del mismo tipo de KF y nos quedamos con la relación de gráficos útiles de los anteriores con respecto al total anterior. Este mismo proceso se realiza con los gráficos y los mensajes que transmitían en la base de datos, así se añade el criterio del usuario a la selección de gráficos, ayudando a esta.

Para definir las reglas nos auxiliamos de los usos de los gráficos y el tipo de mensaje que cada uno muestra. Así se definen un conjunto de reglas que se muestran a continuación en la Figura 18, [17] :

1. Cantidad de categorías del gráfico.
2. Cantidad de series.
3. Si es una comparación sobre el tiempo.
4. Si muestra composición de un total.
5. Cantidad de divisiones de las categorías.
6. Números continuos o discretos.
7. Series en igual intervalo de tiempo.
8. Muchos puntos por serie.
9. Series de diversos tamaños.

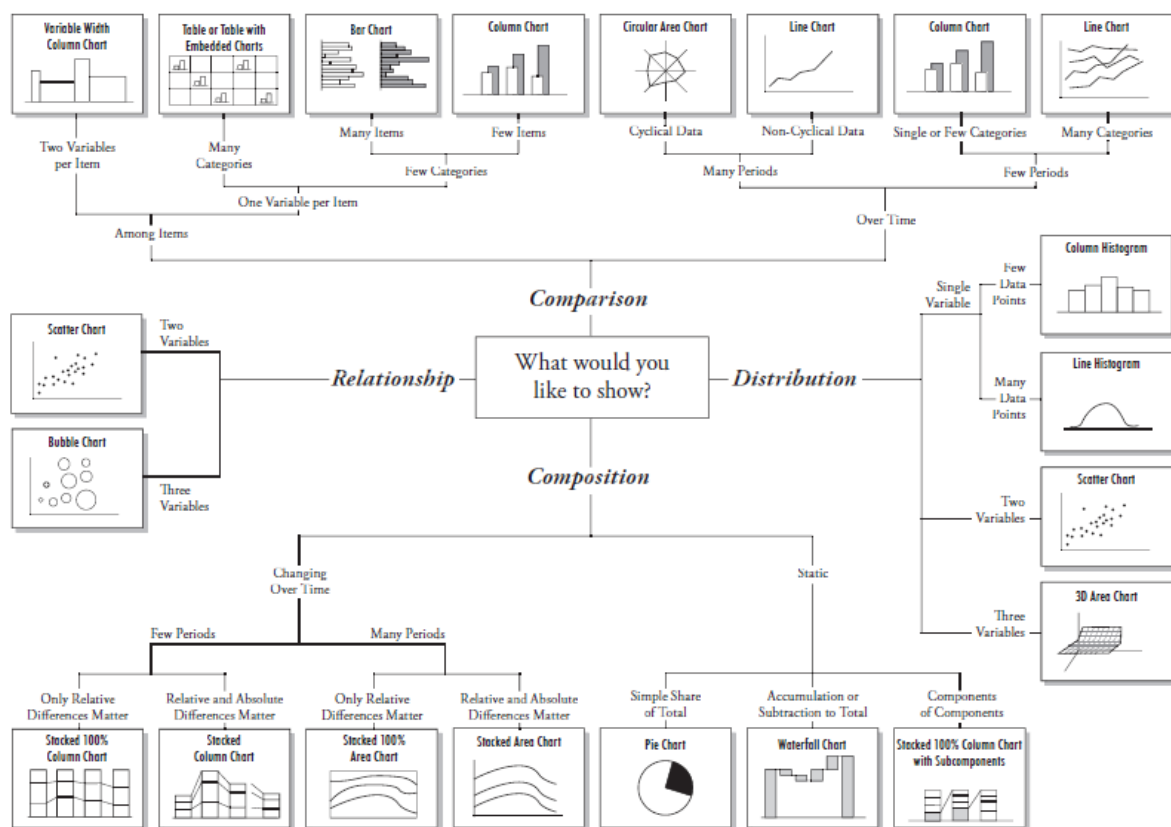


Figura 18. Sugerencias de gráficos acorde los mensajes que transmite [17].

### 3.4.4 Flujo

Con la ayuda de la Figura 8, a continuación se explica esta fase del programa, y se describe detalladamente cómo está concebido el flujo del mismo.

Para el funcionamiento de esta etapa se importan dinámicamente los gráficos definidos en la ruta establecida en la configuración. Asumiendo que la selección inteligente esta activada, se pasa a buscar, utilizando el SBR, cuáles son los gráficos que mejor visualizan los datos para generarlos. En caso que no esté activada esta selección, se buscarán todos los gráficos que aceptan este KF y se generan. Al generar los gráficos, estos se almacenan en la base de datos con sus propiedades.

De esta forma, se obtiene un módulo fácilmente extensible, el que trata de generar los mejores gráficos, donde se pueden agregar fácilmente tanto gráficos como reglas a nuestro SBR, para mejoras futuras.

## 3.5 Retroalimentación

La experiencia que puede transmitir el usuario es de vital importancia. Con el objetivo de no desperdiciar el criterio del usuario con los gráficos generados, se decidió elaborar una pequeña aplicación que le permita al usuario seleccionar qué gráficos le fueron de utilidad y cuáles no. En este acápite se analizan las herramientas utilizadas para la elaboración de este módulo de la aplicación, de igual forma vemos el flujo de esta, cómo interactúa con el resto de los módulos y se muestra cómo está compuesta la base de datos del programa.

### 3.5.1 Herramientas Auxiliares

Para el desarrollo de este módulo, se intentó implementar, a partir de los mismos archivos HTML que se generan, un mecanismo que permita almacenar en una base de datos los gráficos que le fueron de utilidad al usuario. Esto no es posible hacerlo mediante código JavaScript (JS por sus siglas) sin utilizar un servidor. Se ha escogido **electron** para desarrollar una aplicación que visualice el archivo generado y permita al usuario elegir los gráficos que le fueron de utilidad.

**Electron** es un *framework* que permite escribir aplicaciones de escritorio multiplataforma utilizando JS, HTML, CSS. Se basa en Node.JS<sup>7</sup> y Chromium<sup>8</sup> [18].

### 3.5.2 Flujo

Este módulo presenta un flujo sencillo. Es una aplicación de escritorio con la que se puede cargar un archivo HTML generado por el flujo principal del programa, la cual lo visualiza y permite al usuario marcar los gráficos que le fueron de utilidad. Luego el usuario busca la base de datos de la aplicación para actualizarla.

### 3.5.3 Base de Datos

La base de datos del programa se encuentra ubicada en “./api/utills/data\_base.json”. Es un archivo “**.json**”<sup>9</sup> el cual almacena los gráficos generados por el programa con su identificador y propiedades necesarias. En la Figura 19 se muestra cómo se almacena un gráfico de tipo columna, generado a partir de un KF llamado **NumSeries** el que transmite los mensajes de comparación y distribución, y ha sido marcado como útil.

---

<sup>7</sup> Entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor, usando JS.

<sup>8</sup> Navegador de código abierto.

<sup>9</sup> Acrónimo de JavaScript Object Notation, es un formato de texto ligero para el intercambio de datos.

```
{
    "id": 22,
    "kf": "NumSeries",
    "message": ["comparison", "distribution"],
    "properties": {...},
    "type": "column",
    "useful": true
}
```

Figura 19. Fragmento de la base de datos del programa.

De esta forma tenemos un módulo que interactúa con la generación de gráficos y con el usuario, el cual permite apoyarse del criterio del usuario para poder seleccionar mejores gráficos según los datos con que se cuenta.

### 3.6 Estructura

A continuación se muestra la estructura y distribución del programa, mostrada en la Figura 20.

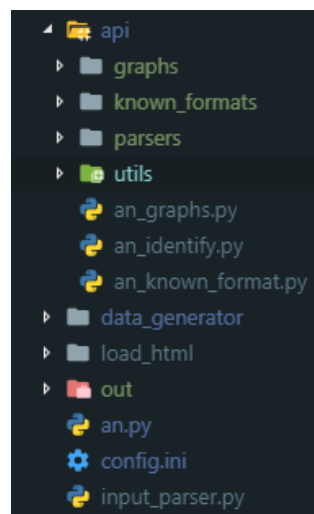


Figura 20. Distribución del programa.

La raíz del programa consta de:

1. Un archivo **"an.py"**, que es el archivo principal del programa el que inicia el flujo del mismo, un archivo **"input\_parser.py"** y un archivo **"config.ini"**.
2. La carpeta **"api"** contiene:
  - a. Los módulos principales del programa en los directorios: **"graphs"**, **"known\_formats"** y **"parsers"**; donde estos son gestionados por los ficheros **"an\_graphs.py"**, **"an\_identify.py"**, **"an\_known\_format.py"**.

- b. La carpeta “**utils**” la cual posee archivos útiles para la implementación de la aplicación y una base de datos de la cual nos apoyamos para la retroalimentación de la aplicación.
3. Las carpetas “**data\_generator**” y “**out**” contienen los archivos con juegos de datos que se generan y los archivos resultantes al terminar el proceso.
4. La carpeta “**load\_html**” es la encargada de la retroalimentación.

## Ejemplo

A continuación, se muestra a modo ejemplo una ejecución completa del programa. Partiendo de una entrada por el CLI y culminando en los gráficos generados.

Entre los *parsers* implementados se encuentran: “**p\_num\_pair\_list**” y “**p\_numbers\_list**”. Estos reconocen cadenas de texto que contienen listas de pares de números y listas de números en cada línea, respectivamente. Se auxilian de una gramática para reconocer esta estructura y extraer las listas de la cadena. El *parser* “**p\_num\_pair\_list**” genera un solo KF: “**PairsSeries()**”; mientras que, “**p\_numbers\_list**” genera dos tipos de KF: “**NumSeries()**” y “**BoxplotSeries()**”.

Entre los gráficos implementados se encuentra el gráfico “**g\_line\_graph**” que genera gráficos de línea y entre los KF que acepta se encuentra **NumSeries()** y “**PairsSeries()**”.

Supongamos que tenemos un archivo de texto “.\\data\_generator\\prueba.txt” son los datos:

```
“[[2,2],[1,16],[0,9]]
```

```
[[0, 5],[1,12],[2,7]]
```

```
[5,20,46,12]
```

```
[15,2,26,13]”
```

Asumamos que la entrada del usuario es mostrada en la Figura 13 y el archivo “**config.ini**” es el mostrado en Figura 14.

### Interacción con el Usuario

Por la entrada estándar, el usuario pide que se analicen los datos solamente con los *parsers*: **p\_numbers\_list** y **p\_num\_pair\_list**; mientras que en el fichero está habilitado el *parser* **p\_csv**, por lo tanto, la configuración final tendrá como *parsers* habilitados **p\_numbers\_list** y **p\_num\_pair\_list**. El archivo a analizar es “.\\data\_generator\\prueba.txt” y el gráfico que se utilizará para mostrar la información extraída es **g\_line\_graph**.

## Identificador de Datos

Con esta configuración el programa en el módulo Identificador de Datos:

1. No reconoce la cadena completa.
2. Pasa a ser procesada por fragmentos, tratando de analizar línea por línea para ver si se reconoce alguna estructura. En este paso “`p_num_pair_list`” reconoce las dos primeras líneas creando su KF en cada una y “`p_numbers_list`” reconoce las dos últimas creando sus Formatos Conocidos en ellas. Quedando como representación de las líneas:
  - a. [`“PairsSeries()”`]  
[`“PairsSeries()”`]  
[`“NumSeries()”, “BoxplotSeries ()”`]  
[`“NumSeries()”, “BoxplotSeries ()”`]
3. Luego se pasan a compactar estos KF. Para esto se ve línea a línea los KF adyacente que son de la misma clase y estos se fusionan. Queda como resultado:
  - a. [`“PairsSeries1()”, “NumSeries1()”, “BoxplotSeries1 ()”`]
  - b. Siendo las series de cada KF:  
`Serie1 = [[2,2],[1,16],[0,9]], Serie2=[[0, 5],[1,12],[2,7]]`  
`Serie1 = [5,20,46,12], Serie2 = [15,2,26,13]`  
`Serie1 = [5,20,46,12], Serie2 = [15,2,26,13]`

## Graficar

En este punto del programa con esta lista de KF, se recorre uno a uno buscando la mayor cantidad de gráficos posibles que los grafiquen como se explica:

1. Se recorren todos los gráficos habilitados por el usuario para ver cuáles son los gráficos compatibles con el KF a analizar. Solo se habilitó el gráfico “`g_line_graph`” y este es compatible con estos KF, a excepción de “`BoxplotSeries()`”. Por lo que nos quedamos solamente con: “`PairsSeries1()`”, “`NumSeries1()`”.
2. El SBR no hace diferencia pues solamente se mostrará un solo tipo de gráfico.
3. Se genera el archivo el cual se puede visualizar en la Figura 21.

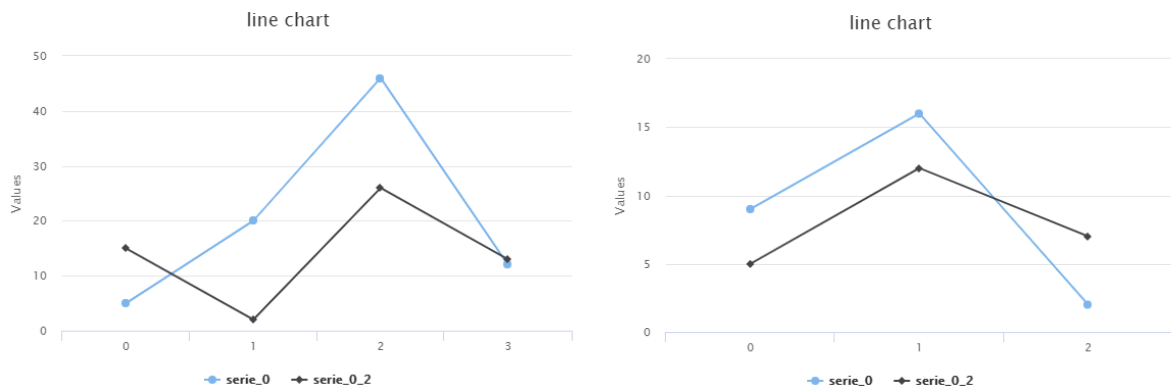


Figura 21. Resultado de Ejemplo de gráfico de línea.

## Epílogo

En este capítulo se detalló cómo fue implementado el sistema descrito en el Diseño del Programa. Se analizan las particularidades de cada módulo y se muestra un ejemplo de una interacción con el programa, desde que el usuario entra los parámetros hasta la generación de gráficos.

Como fue detallado en el capítulo, para la ejecución correcta del programa es necesario que el usuario tenga los siguientes requisitos: python3 con las librerías **configparser**, **json**, **re**, **ply**; **Highcharts JS**; **electron**.

## Capítulo 4. Evaluación del Programa

Con el objetivo de ejemplificar la utilidad y funcionabilidad de este programa se utilizó una base de datos perteneciente al departamento de Inmunoquímica del CIREN<sup>10</sup>, un estudio de una rama de investigación de las Neurociencias, donde se evalúan diferentes variables en un grupo de pacientes.

La base de datos esta almacenada en un archivo CSV (Figura 22). Se espera que el programa lo reconozca en el primer análisis de los datos y no pase a fragmentarlo, lo cual hace correctamente. Luego genera dos KF como era de esperar, y los gráficos. Al ejecutarse el programa con la selección inteligente, se muestran 4 tipos de gráficos por cada KF (8 en total), de los cuales 5 resultaron útiles. Mientras que, sin esta selección, muestra 9 por cada KF (18 en total), de los cuales 8 resultaron de utilidad. En resumen, de 18 gráficos 8 son de utilidad y de estos 8, 5 son seleccionados por el selector de datos. De esta forma se evidencia la utilidad de mostrar la mayor cantidad de gráficos posibles y así brindar al usuario diversas opciones para su elección.

Un ejemplo de los gráficos que fueron de utilidad se muestra en la Figura 23. De igual forma, en la Figura 24 se muestra uno de los gráficos que no resultó de utilidad, por el hecho que mostraba demasiadas series y categorías complejizando su interpretación.

Nombre	Tiempo en	Numero d	C3	C4	PCR	MDA	Nox	Glutation	AOPP	Vit C
PAC1	12	5	86.5	12.5	0.691	17.5	5.152	33.05	30.375	0.149
PAC2	16	12	112	30.1	0.285	25.02	4.564	29.7	71.237	0.539
PAC3	8	3	86.6	23.4	0.1	29.13	3.741	32.79	36.331	0.19
PAC4	9	2	105	24.4	0.158	40.53	5.818	39.39	219.299	0.607
PAC5	2	3	81.3	12.5	0.136	50.4	28.226	26.44	43.358	0.709
PAC6	15	12	166	27.3	1.67	87.76	4.564	55.34	225.39	0.194
PAC7	40	9	90	23	1	69.08	12.83	29.35	230.722	0.918
PAC8	15	12	177	25.3	0.639	58.03	3.154	23.89	36.833	0.136
PAC9	17	1	131	18.4	0.175	34.42	8.129	26.62	51.308	0.534
PAC7	40	9	120	12	0.6	69.08	12.83	29.35	230.722	0.918
PAC1	13	0	70.1	14	0.4	10	3.5	42	23.5	0.87
PAC3	9	0	56.1	22.2	0.2	30.1	2.12	31.4	24.7	0.9
PAC4	10	1	78	21	0.78	34.6	4.8	23.5	67.9	1.05
PAC5	3	0	56.7	23	0.12	40.1	18.6	12.5	23.7	0.98
PAC6	16	6	89	28.7	1.67	56.8	3.8	45.7	124.3	0.76
PAC8	16	0	106	21.2	0.4	34.7	2.1	12.45	14.67	0.87
PAC9	18	0	78.5	14.6	0.2	21.6	6.9	23.5	43	0.89

Figura 22. Base de datos.

<sup>10</sup> Centro Internacional de Restauración Neurológica.



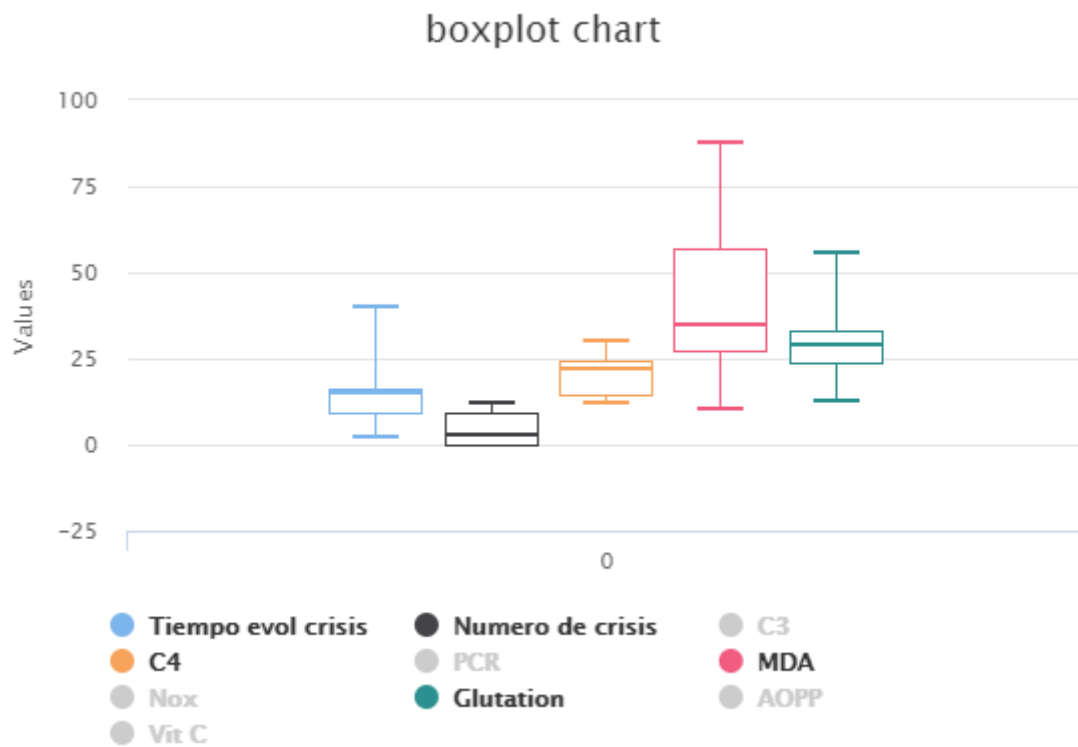


Figura 23. Ejemplo de gráfico generado útil.

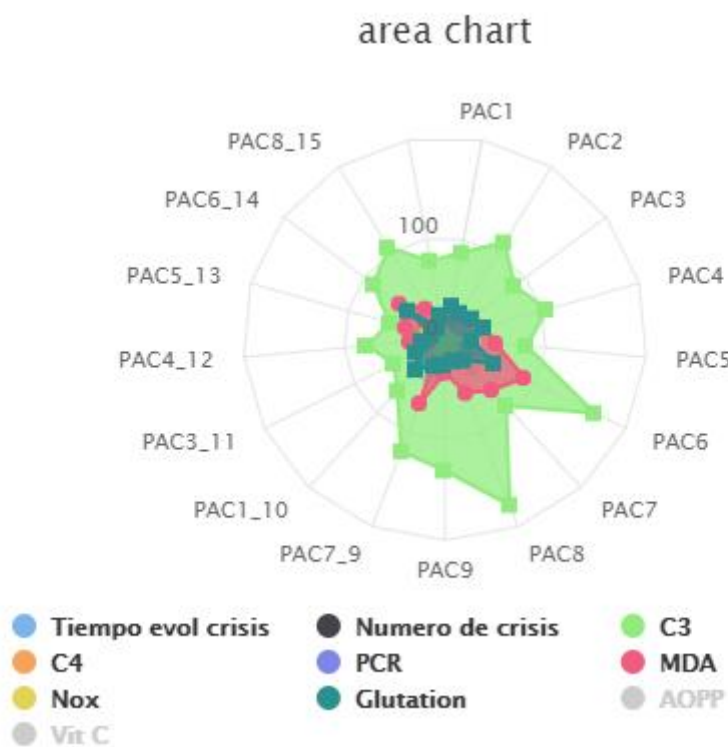


Figura 24. Ejemplo de gráfico generado no útil.

Los resultados de la ejecución del programa con una base de datos real fueron útiles al usuario. Se destaca la facilidad que brinda poder ver y comparar la evolución de los pacientes

con los tratamientos recibidos. El programa recibió sugerencias por parte del usuario para añadirle, como es el caso de poder seleccionar las columnas o categorías a mostrar.

La interacción de un usuario con el programa nos deja como resultado que el mismo es útil. Es capaz de procesar conjuntos de datos, extraer posible información útil y luego generar gráficos para facilitar la interpretación de esta información. Genera un conjunto de gráficos para que el usuario, posible científico, investigador, analista de datos, periodista, etc. pueda elegir el que mejor le resulte para su objetivo final. Sienta las bases de un software futuro con amplios campos de desarrollo como la minería de datos, visualización de datos y selección de gráficos acorde a los datos iniciales.

Es bueno destacar que si se amplía el conjunto de *parsers* el programa puede profundizar a gran escala en la minería de datos, análisis de datos, etc. Pudiendo incluso procesar datos de audios, videos, bases de datos, entre otros. Esto brinda muchos campos para desarrollar y explotar las ventajas de esta aplicación. De igual forma se puede mejorar la selección inteligente de gráficos, profundizando en este aspecto con técnicas de inteligencia artificial. También la extensibilidad de los módulos, como el módulo de Graficar, permite adicionar nuevos gráficos, infografías, mapas e incluso diagramas que brinden nuevos puntos de vista al usuario.

## Conclusiones

Se realizó un estudio de las herramientas, técnicas y estado del arte del campo de estudio y desarrollo de esta Tesis de Diploma.

Se propone un diseño de una aplicación con una interfaz amigable al usuario (CLI). Esta se auxilia de módulos y permite extraer información para visualizarla en gráficos.

Se define un mecanismo que nos permite reconocer datos, los cuales no se sabe exactamente qué estructura tienen, procesarlos y extraer información. Este se apoya en *parsers* mediante gramáticas y expresiones regulares.

El programa define un mecanismo de aprendizaje, que sirve como prueba de concepto, para la selección de gráficos apoyándose en un SBR. Permite tener en cuenta el criterio del usuario (elección de cuales gráficos son de utilidad y cuáles no), para lograr una retroalimentación.

La distribución e implementación del programa posibilita una fácil adaptación y extensibilidad, permitiendo añadir nuevos *parsers*, Formatos Conocidos y gráficos de manera amena para futuras mejoras, cumpliendo así con buenas prácticas de programación. De igual forma, el modelo propuesto para la selección inteligente de gráficos se desarrolló con el propósito de adicionar nuevas reglas para aumentar la precisión de las decisiones, y mejorar así las propuestas de gráficos que se le realizan al usuario.

Se evaluó una base de datos que permitió corroborar la funcionabilidad y utilidad del programa. Esta evaluación aportó algunos criterios de recomendaciones futuras y muestra el potencial del programa con su desarrollo futuro.

## Recomendaciones

En el desarrollo del presente Trabajo de Diploma, consideramos oportuno presentar las siguientes propuestas a modo de recomendaciones para futuras investigaciones que permitan el desarrollo de este programa:

1. Expandir tipos de entrada de datos del programa, no solo cadenas de texto. Poder extraer información de bases de datos, hojas de cálculo Excel, etc.
2. Ampliar el sistema de selección inteligente de gráficos.
3. Agregar diversos tipos de salida del programa como puede ser Matplotlib.
4. Agregar nuevas estrategias de procesamiento de datos más profundas, apoyándose en técnicas de minería de datos.

## Bibliografía

- [1] G. Bellinger, D. Castro y A. Mills, «Data, Information, Knowledge, and Wisdom,» 2004. [En línea]. Available: <http://www.systems-thinking.org/dikw/dikw.html>. [Último acceso: 20 05 2019].
- [2] N. F. Kock, R. J. McQueen y J. L. Corner, «The nature of data, information and knowledge exchanges in business processes: implications for process improvement and organizational learning,» *The Learning Organization*, vol. 4, nº 2, pp. 70-80, 1997.
- [3] J. M. Chambers, W. S. Cleveland , B. Kleiner y P. A. Tukey, *Graphical Methods for Data Analysis*, Florida: CRC Press, 2018.
- [4] D. Kelly, J. Jasperse y I. Westbrooke, «Designing science graphs for data analysis and presentation. The bad, the good and the better,» Science & Technical Publishing, New Zealand Department of Conservation, 2005.
- [5] J. Han, M. Kamber y J. Pei, *Data Minins Concepts and Techniques*, Watham: Elsevier Inc, 2012.
- [6] E. A. Doctor Bracho, «Técnicas Estadísticas en Minería de Textos,» Universidad de Sevilla, Sevilla, 2018.
- [7] J. Flores Vivar y C. Salinas Aguilar, «Sinergias en la construcción del Nuevo Periodismo derivadas del Data Journalism y el Transmedia Journalism,» *Actas III Congreso Internacional Comunicación 3.0*, Salamanca, 2012.
- [8] F. Ramírez, «LAPATRIA.com,» 15 05 2019. [En línea]. Available: <http://www.lapatria.com/blogs/periodismo-de-datos-periodismo-de-siempre>.
- [9] G. Zelazny, *Say it with Charts*, The McGraw-Hill Companies, Inc., 2001.
- [10] M. Tatay, «Prisma,» 10 agosto 2016. [En línea]. Available: [beprisma.com/como-elegir-la-mejor-grafica-mostrar-datos](http://beprisma.com/como-elegir-la-mejor-grafica-mostrar-datos).
- [11] T. R. Newman, J. B. Evans y A. M. Wyglinski, «Reconfiguration, adaptation, and optimization,» de *Cognitive Radio Communications and Networks. Principles and Practice*, Elsevier Inc., 2010, pp. 177-198.

- [12] J. Flores Vivar y C. Salinas Aguilar, «El periodismo de datos como especialización de las organizaciones de noticias en Internet,» *Correspondencias & Análisis*, pp. 15-36, 2013.
- [13] Tableau, «Tableau,» 15 05 2019. [En línea]. Available: <http://www.tableau.com/learn/get-started/data-structure>.
- [14] IBM, «IBM,» 15 05 2019. [En línea]. Available: <http://ibm.com/products/spss-statstics>.
- [15] IBM, «Open source and IBM SPSS modeler The Best of the word,» IBM Analytics.
- [16] M. Á. Chaparro Domínguez, «Nuevas formas informativas: el periodismo de datos y su enseñanza en el contexto universitario,» Universidad Internacional de La Rioja, 2014.
- [17] A. V. Abela, *The Presentation A Story About Communicating Successfully With Very Few Slides*, 2010.
- [18] GitHub.inc, «GitHub,» GitHub, [En línea]. Available: [www.github.com/electron/electron](http://www.github.com/electron/electron). [Último acceso: 28 05 2019].
- [19] G. Van Rossum, «Argparse Tutorial,» Python Software Foundation, 2018.