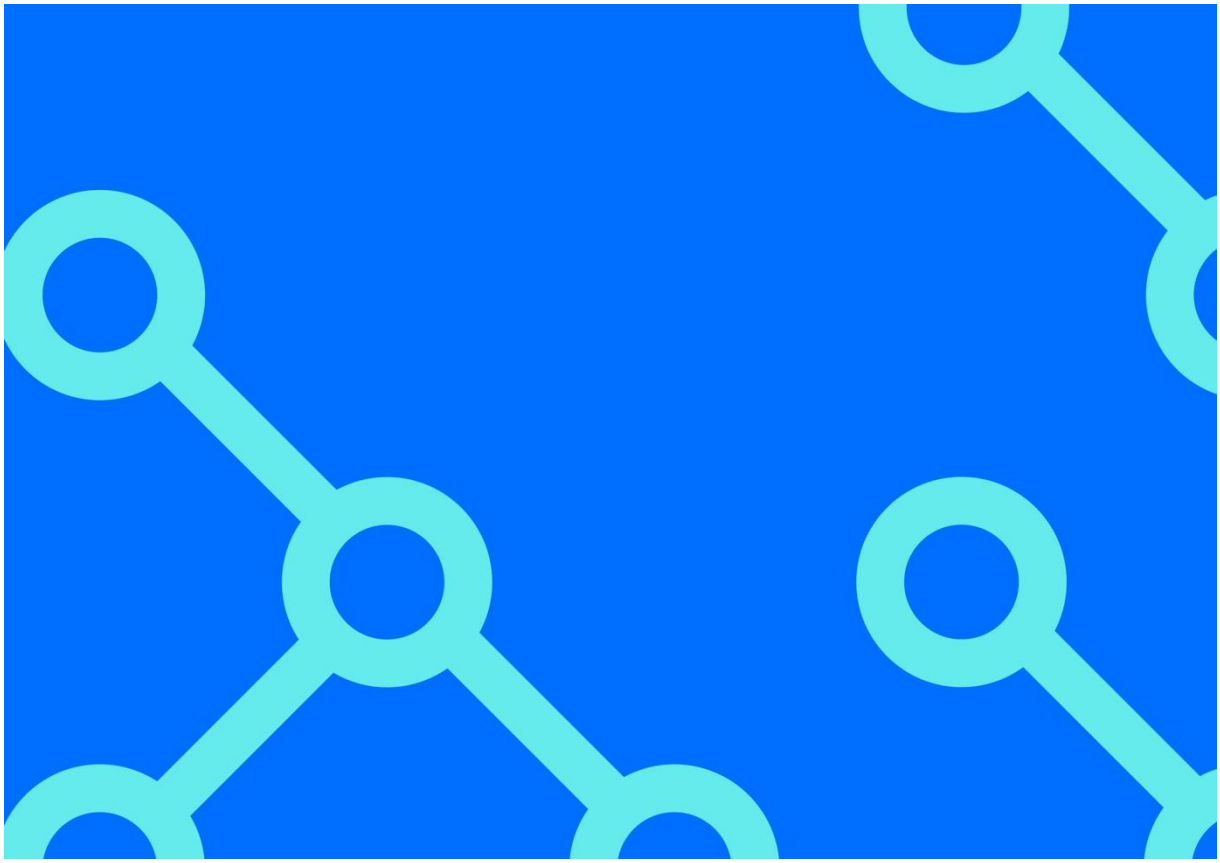
AKA 02-02/ TAREA 2.2 GNOSS HÉCTOR SERNA

Código	AKA 02-02
Cliente	GNOSS AKADEMIA
A la atención de	JAVIER AREVALO ROYO
Fecha de entrega	12/09/2024
Versión del documento	0001
Persona de contacto GNOSS	HÉCTOR SERNA GUTIÉRREZ
Teléfono	941 24 89 05
Mail	hectorserna@gnoss.com administracion@riam.es

ÍNDICE



1 METODOLOGÍA

1.1 Origen de los datos.

La selección del data set se ha hecho siguiendo la recomendación del enunciado, accediendo al repositorio de GITHUB: (https://github.com/awesomedata/awesome-public-datasets). Tras leer el abstract inicial que aparece elegí buscar entre los temas de economía y decidí usar el de índice de libertad económica puesto que al tratar con países iba a poder manejar datos que seguían el estandar ISO 3166 que se ha mencionado en la teoría.

El dataset se accede desde el Instituto Fraser https://www.fraserinstitute.org/economic-freedom/dataset . Un hecho que me parecía interesante de este dataset es que se podía seleccionar el rango de años de estudio , además de tener en la parte inferior derecha apartados de mapas y gráficos que facilitaban la investigación previa al proceso ETL, pudiendo hacerme una idea ligera de que me iba a encontrar.

Otra nota interesante del dataset en cuestión es que el formato en el que se descargaban los datos variaba según si fracionabas por rango de años (un csv) o descargabas todo su conjunto(xsl). Yo decidí fraccionarlo en un rango que iba desde el inicio del milenio en el 2000 hasta el último año del que se tenían datos, esto me llevó a trabajar con un csv.

1.2 Proceso de extracción

La decisión de herramientas y formatos que se eligen en este momento es crucial para todo el proceso ETL pues marca la pauta a seguir e influye notoriamente en el ritmo en el que se va desarrollar el proyecto.

1.2.1 Herramientas y lenguaje

Evidentemente en este caso para seguir las indicaciones corporativas Gnoss usé Visual Studio para extraer la información del fichero . Generé un repositorio local y añadí el dataset, posteriormente lo vincule al sistema remoto de github, todo esto através del gestor de verwsiones embebido dentro del propio Visual Studio.

Posteriormente modifiqué ligeramente el código entregado en la teoria para que solo leyera la primera línea del archivo ; puesto que esta tenía el nombre de los campos columnas. Así fue como comprobé de manera sencilla lo que había vaticinado en la revision manual.

```
String line= File.ReadLines(filePath).ToList().First();
string[] fields = line.Split(',');
```

```
for (int i = 0; i < fields.Length; i++)
{
    Console.WriteLine($"Campo {i + 1}: {fields[i]}");
}
```

Después modifiqué y leí la segunda linea también para observar a que tipos debía parsear para que hubiese coherencia en la estructura de datos de la clase a tratar.

Al tratar de usar el código de manejo de CSV con el paquete NuGet de Toolkit, encontré problemas al instanciar un CsvConfig pues no reconocía adecadamente todos los subpaquetes de la dependencia y parecía no haber sido creado. Por ello opté por la altenativa de usar CsvHelper en su lugar y generar un mappeo de los datos para poder construir los objetos de manera más eficiente. La opción de mappear me pareció adecuada pues permite una mayor separación de funciones y es más escalable si en un futuro la estructura de datos se amplia. En este proyecto se trata con un solo data set y por simplicidad y comodidad la clase de datos de registro país, el mapa y el programa principal de consola se han implementado en un solo archivo; pero esta forma de proceder se puede seguir en proyectos más grandes generando carpetas separadas con clases propias tanto para mapas como para las clases que instancian.

```
public class RegistroPaisMap : ClassMap<RegistroPais>
{   public RegistroPaisMap()
    {       Map(m => m.Year).Name("Year").Optional();
        Map(m => m.ISOCode).Name("ISO Code");
        Map(m => m.Countries).Name("Countries").Optional();
        Map(m => m.EconomicFreedomIndex).Name("Economic Freedom Summary Index").Optional();
        Map(m => m.Rank).Name("Rank").Optional();
        Map(m => m.Quartile).Name("Quartile").Optional();
        Map(m => m.SizeOfGovernment).Name("Size of Government").Optional();
        Map(m => m.LegalSysPropertyRights).Name("Legal System & Property Rights").Optional();
        Map(m => m.SoundMoney).Name("Sound Money").Optional();
        Map(m => m.FreedomTradeInternationally).Name("Freedom to trade internationally").Optional();
        Map(m => m.Regulation).Name("Regulation").Optional();
    }
}
```

Introduje con los Nuget de Csv el mapa al contexto y pude empezar a instanciar clases con los datos de cada registro.

```
using (var reader = new StreamReader(filePath))
using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture))
{

    csv.Context.RegisterClassMap<RegistroPaisMap>();
   var Paises = csv.GetRecords<RegistroPais>().ToList();
    foreach (var pais in Paises)
    {
       Console.WriteLine($"Pais:{pais.Countries}\t Año:{pais.Year}\t  Rank:{pais.Rank}");       }
}
}
```

Al tratar de trabajar con el mapa la primera ejecución dio lugar a muchas excepciones en las que el mapa intentaba asignar un valor nulo a un campo. Esto se debía a que al principio no había visto ni previsto que hubiesen campos vacíos pues el enlace original desde github estaba marcado en verde, cosa que la leyenda indicaba que era un buen dataset. De esta breve esperiencia he aprendido varias cosas:

- Un dataset no tiene que tener toda la información en cada registro para ser considerado bueno; y que la mayoria de dataset no lo están. Esta era una reflexión que ya había hecho previamente en otras experiencias pero con el tiempo y la falta de uso había olvidado.
- No fiarte de lo que indica la fuente al 100% pues p'uede que la información no esté actualizada o sea erroneo.
- Aunque el problema se solventó rápida y sencillamente debido a que el dataset constaba de escasa columnas y los mensajes de excepciones eran bastante claros en Visual Studio, para futuros datasets de mayor embergadura convendría hacer una gestion de excepcion de errores de este tipo las primeras veces para que se localice de una manera eficiente que campos pueden ser nulos o haya elegido inadecuadamente el tipo. Si el dataset consta de muy pocas columnas habría que valorar si conviene o no hacer los try-catch o no segun la velocidad que requiera el proyecto.

El problema se solucióno indicando en la clase y el mapa como opcionales las clases que habían mostrado algun valor nulo en algun momento.

Mientras iba generando el código mantuve algunos comentarios útiles del código tomado de la teoría, añadí comentarios propios y a forma de costumbre aunque todo era bastante autoexplicativo genere los comentarios adecuados para posteriormente autogenerar la documentación. Para esta última parte de la documentación usé Doxygen que tenia en mi equipo personal de años anteriores. No se personalizó ni css y el contenido generado en esa documentación es casi trivial debido a lo autoexplicativo de las clases atributos y métodos pero

fue mñas una demostración de buenas prácticas que de algo qu evaya a ser de utilidad en esta tarea en concreto.

El código en git hub:

https://github.com/hectorsernagutierrez/PruebaGNOSSRecoleccionDatos.git

Documentación DOxygen:

. html/index.html