

|  |
| --- |
| **Universidad Internacional de La Rioja**  **Escuela Superior de Ingeniería y Tecnología**  **Máster Universitario en Análisis y Visualización de Datos Masivos** |
|  |
| Captura y procesamiento de información sobre declaración tributaria Ecuador 2020-2022 |

**Trabajo Fin de Máster**

**Tipo de trabajo:** Desarrollo Software

**Presentado por:** Del Pino Guadalupe, Byron

**Director/a:** Martí, Julio Marcelo

**Resumen**

En la sociedad actual, con el advenimiento de nuevas tecnologías computacionales y el *Big Data*, organizaciones de todo tipo y tamaño buscan implementar soluciones basadas en los datos para la toma de decisiones que faciliten las estrategias del negocio o mejoras en su rendimiento. El análisis de datos en tiempo real permite una toma de decisiones oportuna, que para el caso de declaraciones de impuestos constituyen una obligación para la identificación de patrones o conductas como la elusión y evasión fiscal, o aquellos sectores que requieren de correctas políticas gubernamentales para impulsar su crecimiento económico que coadyuven a una mayor recaudación fiscal. En este trabajo se diseñó y construyó una arquitectura para capturar y procesar datos de declaraciones de impuestos en tiempo real, consolidándolos con fuentes históricas agregadas, y se representó en cuadros de mando que permiten la toma de decisiones. También se implementaron dos modelos de *Machine Learning* de clusterización para la identificación de grupos de cantones que, a través de sus distintas realidades, se determine el grado de afectación que la pandemia del COVID-19 pudo afectar en sus declaraciones de impuestos.

**Palabras Clave:**

*Big Data*, *Machine Learning*, *Dashboard*, Tiempo Real, Declaraciones de Impuestos

**Abstract**

In today's society, with the advent of new computing technologies and Big Data, different types and sizes of organisations seek to implement data-driven solutions for decision-making in order to either facilitate business strategies or improve performance. Real-time data analysis allows timely decision-making, which in the case of tax returns is compulsory to identify patterns or behaviors such as tax avoidance and tax evasion, or those sectors that require correct government policies to promote economic growth that contribute to a greater tax collection. In this project it was designed and constructed an architecture that captures and processes tax return data in real time, consolidating it with aggregated historical sources, and its representation in dashboards that allows decision making. Additionally, two clustering Machine Learning models are implemented for the identification of groups of cantons that, through their different realities, determine the degree of affectation that the COVID-19 pandemic could affect their tax returns.

**Keywords:**

Big Data, Machine Learning, Dashboard, RealTime, Tax return

**Índice de contenidos**

[1. Introducción 9](#_Toc108603098)

[1.1 Justificación 9](#_Toc108603099)

[1.2 Planteamiento del trabajo 10](#_Toc108603100)

[1.3 Estructura de la memoria 12](#_Toc108603101)

[2. Contexto y estado del arte 13](#_Toc108603102)

[2.1. Arquitecturas para el procesamiento de datos 13](#_Toc108603103)

[2.1.1 Arquitectura Lambda 13](#_Toc108603104)

[2.1.2 Arquitectura Kappa 14](#_Toc108603105)

[2.2. Bases de Datos NoSQL 15](#_Toc108603106)

[2.2.2 Características de una BDD NoSQL y CAP 15](#_Toc108603107)

[2.2.3 Tipos de Bases NoSQL 16](#_Toc108603108)

[2.3. Visualización de datos para la toma de decisiones 16](#_Toc108603109)

[2.4. Herramientas de aprendizaje automático para clusterización: Scikit-Learn 18](#_Toc108603110)

[2.5. Casos de Uso 19](#_Toc108603111)

[2.5.1 Arquitectura Kappa: Uber 19](#_Toc108603112)

[2.5.2 Apache Spark: Casos de uso para el procesamiento de datos 20](#_Toc108603113)

[2.5.3 ELK (Elasticsearch, Logstash y Kibana) para el análisis y reportes 21](#_Toc108603114)

[2.5.4 Scikit-Learn en la Educación 23](#_Toc108603115)

[2.6 Contribución 24](#_Toc108603116)

[3. Objetivos concretos y metodología de trabajo 26](#_Toc108603117)

[3.1. Objetivo general 26](#_Toc108603118)

[3.2. Objetivos específicos 26](#_Toc108603119)

[3.3. Metodología del trabajo 27](#_Toc108603120)

[4. Desarrollo específico de la contribución 29](#_Toc108603121)

[4.1. Carga de Información Histórica 30](#_Toc108603122)

[4.1.1 Modelo de Datos 30](#_Toc108603123)

[4.1.2 Importación de Datos Históricos 32](#_Toc108603124)

[4.2. Generación de datos de detalle 39](#_Toc108603125)

[4.2.1 Modelo de Datos 39](#_Toc108603126)

[4.2.2 Definición de estructuras en Oracle 40](#_Toc108603127)

[4.2.3 Interfaz gráfica 44](#_Toc108603128)

[4.2.3 Resultados de generación de datos de declaraciones a detalle 47](#_Toc108603129)

[4.3. Captura de información a detalle 48](#_Toc108603130)

[4.3.1 Apache Kafka 48](#_Toc108603131)

[4.3.2 Configuración Apache Kafka- Confluent 50](#_Toc108603132)

[4.4. Procesamiento y persistencia de información de declaraciones 54](#_Toc108603133)

[4.4.1 Apache Spark 54](#_Toc108603134)

[4.4.2 Integración Kafka - Spark - ElasticSearch 55](#_Toc108603135)

[4.5. Cuadro de Mando en tiempo real 63](#_Toc108603136)

[4.6. Clusterización 68](#_Toc108603137)

[4.6.1 Análisis y procesamiento de la información 68](#_Toc108603138)

[4.6.3 Código Fuente Relevante 80](#_Toc108603139)

[5. Conclusiones y trabajo futuro 82](#_Toc108603140)

[5.1. Conclusiones 82](#_Toc108603141)

[5.2. Líneas de trabajo futuro 83](#_Toc108603142)

[6. Bibliografía 84](#_Toc108603143)

[Anexos 87](#_Toc108603144)

[Anexo I. Configuración Esquema BDD Oracle XE 87](#_Toc108603145)

[Anexo II. Instalación y configuración Elasticsearch, Logstash y Kibana (ELK) 89](#_Toc108603146)

[Anexo III. Instalación y configuración de Apache Spark 94](#_Toc108603147)

[Anexo IV. Instalación de Geopandas en Anaconda-Jupyter 96](#_Toc108603148)

[Anexo V. Análisis de Componentes Principales 97](#_Toc108603149)

[Anexo VI. Repositorio 99](#_Toc108603150)

**Índice de tablas**

Tabla 1 Principios de un Gráfico Efectivo 17

Tabla 2 Descripción equipos utilizados 29

Tabla 3 Diccionario de datos de declaraciones 30

Tabla 4 Definición de Actividades Económicas en Ecuador 31

Tabla 5 Diccionario de datos de declaraciones al Detalle 39

Tabla 6 Diccionario de datos de declaraciones al Detalle 40

Tabla 7 Clases Java de interfaz gráfica 45

Tabla 8 Fragmentos de código clase Generacion\_Declaraciones 45

Tabla 9 Fragmentos de código clase hilo\_provincia 46

Tabla 10 Componentes gráficos interfaz declaraciones 47

Tabla 11 Variables de configuración Kafka 50

Tabla 12 Componentes del Dashboard de Declaraciones 66

Tabla 13 Código relevante Python para clusterización 80

Tabla 14 Descripción de componentes repositorio Github 99

**Índice de figuras**

Figura 1 Arquitectura para captura y procesamiento de Declaración Tributaria 11

Figura 2 Arquitectura Lambda 14

Figura 3 Arquitectura Kappa 14

Figura 4 Teorema CAP 15

Figura 5 Uber y Kafka 19

Figura 6 Distribución de Compañías usando Kibana en la Industria 23

Figura 7 Clusterización de estudiantes por Media y Desviación 23

Figura 8 Metodología de Trabajo 27

Figura 9 Topología de declaraciones 29

Figura 10 Consola de LogStash para carga de datos 37

Figura 11 Creación de una Vista en Kibana 38

Figura 12 Visualización de Número de Registros por Provincia 38

Figura 13 Dashboard con datos de declaraciones históricos 39

Figura 14 Muestra de datos ubicaciones geográficas 42

Figura 15 Interfaz gráfica declaraciones 44

Figura 16 Datos generados de declaraciones 48

Figura 17 Arquitectura Kafka 49

Figura 18 Muestra de datos capturados por Kafka 53

Figura 19 Arquitectura Apache Spark 54

Figura 20 Operación left\_anti join 60

Figura 21 Operación Inner Join 60

Figura 22 Errores en ejecución continua de datos 62

Figura 23 Interfaz Spark - Jobs en Microbatch 63

Figura 24 Vista declaraciones\_2022 64

Figura 25 Campo calculado MES\_AÑO 64

Figura 26 Gráfico de barras en Kibana 65

Figura 27 Join entre índice geográfico y declaraciones 65

Figura 28 Dashboard declaraciones 2022 66

Figura 29 Refrescamiento de dashboard declaraciones 68

Figura 30 Pasos para la aplicación de modelos de Clusterización 69

Figura 31 Descripción de variables de los datasets importados 70

Figura 32 Identificación y eliminación de datos nulos 71

Figura 33 Estadísticas datos numéricos de Declaraciones 71

Figura 34 Gráficas de cajas de los totales de compras y ventas por provincia 72

Figura 35 Total de compras y ventas por año 72

Figura 36 Varianza acumulada por componentes 73

Figura 37 Definición del número de clústeres mediante el método del codo y coeficiente de silueta 74

Figura 38 Representación de las instancias para 2 y 3 clústeres calculados con K-Means 75

Figura 39 Resumen gráfico de declaraciones por cada clúster 75

Figura 40 Dendongrama y evolución del coeficiente de silueta 76

Figura 41 Representación de las instancias para 2 y 3 clústeres calculados con Algoritmo Aglomerativo 77

Figura 42 Resumen gráfico de declaraciones por cada clúster 77

Figura 43 Comparación de modelos: K-Means vs Aglomerativo 78

Figura 44 Resumen de medias por métricas de declaraciones por cada clúster 78

Figura 45 Mapa del Ecuador distribuido por clústeres 79

Figura 46 Información de los 10 cantones más grandes del Ecuador 79

Figura 47 Creación nueva conexión Oracle 87

Figura 48 Ejecución Inicial de ElasticSearch 90

Figura 49 Página Inicial ElasticSeach 90

Figura 50 Inicio de Servicio de Kibana 91

Figura 51 Configuración Kibana- Token de enrolamiento 91

Figura 52 Página de inicio de autenticación de Elastic 92

Figura 53 Página de Inicio Kibana 92

Figura 54 Versión py4j dentro de SPARK\_HOME 95

Figura 55 Reducción de dimensionalidad con PCA 97

Figura 56 Varianza acumulada por componentes 98

Figura 57 Repositorio Github 99

# 1. Introducción

## 1.1 Justificación

El Servicio de Rentas Internas del Ecuador (SRI) es la institución pública encargada de gestionar la política tributaria del país, incorporando controles y normas que permitan una eficiente recaudación de impuestos y la detección de prácticas fraudulentas como la evasión y elusión fiscal. Como organismo rector de la recaudación fiscal, busca optimizar cada una de estas tareas a través del uso de nuevas tecnologías.

Al ser una institución cuya misión es fomentar la equidad económica de la sociedad ecuatoriana a través de un correcto pago de impuestos, los datos que esta almacena toman especial importancia a la hora de fomentar políticas y tomar decisiones para determinar qué procesos requieren correcciones y mejoras. Muchos de esos datos son generados en tiempo real, por lo que el proceso de captura y procesamiento de la data generada debe responder a las demandas tanto de autoridades como usuarios que consumen dicha información.

El procesamiento de grandes volúmenes de información en tiempo real, así como datos provenientes de fuentes históricas es de utilidad a fin de conocer las fortalezas y debilidades de una organización, y sus posibles oportunidades de mejora. La información resultante podrá ser explotada con total certidumbre a través de distintas herramientas de visualización que faciliten un análisis descriptivo del proceso de negocio, o un análisis predictivo mediante técnicas de *Machine Learning* e Inteligencia Artificial.

El SRI incorpora periódicamente un conjunto de datos públicos en su página web institucional https://www.sri.gob.ec/datasets , esto en adición al fortalecimiento de la transparencia y confianza hacia las instituciones públicas, hace un llamado a que la sociedad y la academia consuman y exploten esta información. Entre los *datasets* públicos, que se encuentran agregados y anonimizados, representan información relacionada ainformación sobre declaraciones de compras y ventas por provincia, mes y actividad económica en el periodo 2017-2021.

A pesar de que esta información histórica pudiera ser relevante en el contexto de soluciones *data-driven*, al ser retroactiva, la toma de decisiones puede no ser acertada o pertinente bajo escenarios en que datos en tiempo real afecten considerablemente un monto o valor clave. Por ejemplo, en el año 2020 con la aparición del COVID-19, las actividades económicas se redujeron drásticamente, afectando el empleo y capacidad adquisitiva de los contribuyentes y la consecuente recaudación del Servicio de Rentas Internas. En ese periodo de tiempo, la ausencia de datos que conlleven a un análisis en tiempo real dificultó al Estado Ecuatoriano y la sociedad en general comprender el impacto del COVID-19 en la economía y qué decisiones debían ser las correctas para atenuar las consecuencias de la pandemia.

A raíz de los programas de vacunación impulsado por el Gobierno Nacional en el 2021, se visualiza una reactivación económica. Un estudio publicado por el Banco Central del Ecuador resalta que la economía ecuatoriana, que en el 2020 tuvo una contracción del 7.8%, estimando que en el 2022 crecerá un 2.54%. Si bien se vislumbra una recuperación, aún los efectos del COVID-19 se sienten en distintos sectores productivos; por lo que, ante este contexto de oportunidades, los usos de tecnologías para la toma de decisiones tienden a ser un factor clave para obtener un análisis verídico de la economía y la recaudación y el fomento de políticas que busquen aceleran ese crecimiento.

## 1.2 Planteamiento del trabajo

La disponibilidad de data consolidada en tiempo real proporciona los medios para la toma de decisiones pertinente y eficiente. Para la consecución de este objetivo, el presente Trabajo Final de Máster propone el diseño e implementación de una arquitectura que capture y procese datos de declaraciones a detalle en tiempo real junto a los *datasets* históricos publicados por el SRI en el periodo 2020-2021. La consolidación de estas dos fuentes de información de forma persistente en una base de datos será visualizada en un cuadro de mando. Adicionalmente la data histórica será de insumo para la construcción de un modelo de clusterización para identificar aquellas instancias que comparten características similares.

La Figura 1 despliega el modelo propuesto para el proceso de captura y procesamiento de declaraciones tributarias, cuyas etapas se detallan a continuación:

1.- El archivo *dataset* publicado en el portal de datos públicos del SRI será cargado en una base de datos no relacional. Esta data es de tipo agregada y anonimizada.

2.- La Información a detalle que proviene de declaraciones tributarias serán datos ficticios que se almacenarán en una base de datos transaccional, se autogenerará información en el rango de miles de registros por minuto.

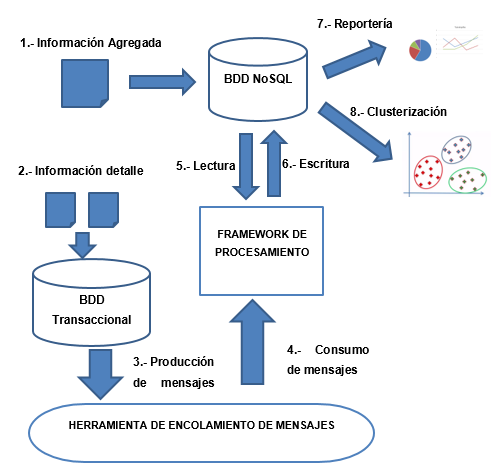
3.- A través de una herramienta de encolamiento de mensajes en tiempo real bajo el modelo publicador-suscriptor, la base de datos transaccional depositará en un bus de datos los mensajes relativos a declaraciones a detalle.

4,5,6.- Con una herramienta de procesamiento, se generará un cliente que consume los mensajes provenientes del bus de datos; simultáneamente, se procederá a leer la data agregada que proviene de la base de datos no relacional. Se harán cálculos de agregación y se procederá a escribir (*update*) en la base de datos los datos recalculados.

7.- Se implementará un *dashboard* que permita ver en tiempo real como varía los datos consolidados. Este reporte desplegará la información agregada por las distintas categorías (año, mes, provincia, e/o).

8.- A partir de la información histórica se implementará un modelo de clusterización que permita identificar clústeres con provincias con características similares de declaración.

Figura 1 Arquitectura para captura y procesamiento de Declaración Tributaria



**Nota**: La figura correspondiente a la sección 8 de Clusterización, fue extraída de

https://rpubs.com/JosueEmmanuel/Kmean

## 1.3 Estructura de la memoria

En esta memoria se describe la estructura del presente proyecto de Fin de Máster, comenzando con la definición del problema de captura y procesamiento de datos de declaración en tiempo real, y el planteamiento de una solución que consolide esta data autogenerada con aquella histórica/real publicada en el sitio web del SRI.

En el capítulo 2 se desarrolla el contexto y estado de arte, donde se hace referencia a literatura relacionada con arquitecturas, soluciones y tecnologías *Big Data* ya existentes para el procesamiento y almacenamiento de grandes y variados volúmenes de datos.

En el siguiente apartado, se definirá los objetivos generales y específicos, los mismos que brindan un enfoque y alcance sobre la contribución que realiza este TFM para la *data* *streaming*. Finalmente se plasmará la metodología de trabajo, donde se detalla las tareas y pasos requeridos para la consecución del presente proyecto.

A continuación, se prosigue con el desarrollo de la contribución, donde se bosqueja las tecnologías a utilizar para la captura, procesamiento, y persistencia de distintos datos aplicando tecnologías *Big Data*; y su consiguiente tratamiento y análisis para la toma de decisiones mediante herramientas de visualización y algoritmos de clusterización exclusiva.

Para finalizar, se cuenta con una sección que plasma las conclusiones levantadas durante el desarrollo de este trabajo, así como posibles líneas de trabajo futuras en el marco de la implementación de soluciones *data-driven* con data generada en tiempo real e histórica.

# 2. Contexto y estado del arte

El desarrollo e implementación de este Trabajo Final de Máster engloba las siguientes líneas de estudio, las cuales han sido abordadas a través de distintas tecnologías y propuestas de solución:

* Arquitecturas para el procesamiento de datos
* Bases de datos NoSQL
* Visualización de datos para la toma de decisiones
* Herramientas de aprendizaje automático para clusterización: Scikit-Learn

## 2.1. Arquitecturas para el procesamiento de datos

El término *Big Data*, que surge como un conjunto de herramientas que brindan una solución a aquellos problemas que son difícilmente abordados con tecnologías tradicionales, ha acaparado la atención de empresas y corporaciones de todo tipo, quienes hoy en día son más conscientes de la importancia de los datos para incrementar su competitividad.

Los datos tienden a ser de diversos tipos, tamaños y se generan a gran velocidad, planteando un desafío interesante a la hora de analizarlos y computarlos para generar conocimiento.

*Big Data* está definido por tres dimensiones: Variedad, Volumen y Velocidad, siendo esta última para (Hasani, 2014) la más difícil de tratar de una forma efectiva; recalcando que una solución *Big Data* debe ser capaz de abarcar procesamiento en *streaming* en una ratio de millones de registros por segundo.

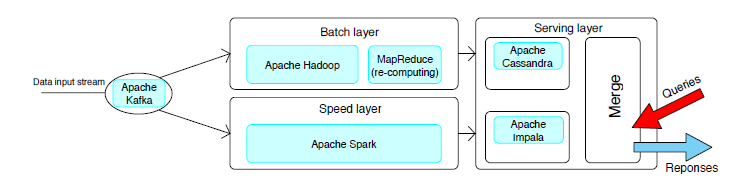
(Feick, Kleer, & Kohn, 2018) resaltan dos arquitecturas *Big Data* a la hora del procesamiento de datos: Lambda y Kappa.

### 2.1.1 Arquitectura Lambda

Lambda es una arquitectura de propósito general conformada por tres capas: Capa de *Batch*, responsable del manejo y cálculo de *datasets* históricos inmutables; la Capa de Velocidad, para el tratamiento de datos cuya ingesta es en tiempo real, y una Capa de Servicio, encargada de consolidar e indexar ambas fuentes de datos y su puesta a disposición para consumo de otras aplicaciones.

En la Figura 2, se visualiza una etapa con data entrante mediante un encolador de mensajes (Kafka), la capa de *Batch* y Velocidad empleando herramientas del ecosistema Hadoop y Spark y la capa de Servicio que consolida la data empleando una Base de Datos (Cassandra) junto a un motor de consultas SQL de procesamiento paralelo (Impala).

Figura 2 Arquitectura Lambda



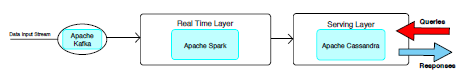
**Fuente**: Martin Feick, Niko Kleer, and Marek Kohn (Hrsg.): *Fundamentals of Real-Time Data Processing Architectures Lambda and Kappa*

Lambda Facilita un escalamiento horizontal y tolerante a fallos en términos de infraestructura y errores humanos; donde los datos entrantes se replican en las capas de Velocidad y *Batch* las cuales deben ser implementadas en dos sistemas separados y sincronizados.

### 2.1.2 Arquitectura Kappa

Kappa, se presenta como una arquitectura desarrollada a partir de Lambda, que busca suplir la alta latencia que genera una capa de *Batch*. La arquitectura Kappa está constituida por dos capas: Tiempo Real, que captura y procesa la data de entrada en tiempo real, y una capa de Servicio, cuya funcionalidad es la consulta y respuesta a peticiones desde otros aplicativos. La Figura 3, describe la arquitectura Kappa, donde las tecnologías utilizadas son las mismas con respecto a Lambda, omitiendo la capa de *Batch*.

Figura 3 Arquitectura Kappa



**Fuente**: Martin Feick, Niko Kleer, and Marek Kohn (Hrsg.): Fundamentals of Real-Time Data Processing Architectures Lambda and Kappa

Kappa brinda mayor simplicidad al contar con un único sistema que consolida los procesamientos por lotes(*Batch*) y en tiempo real (Velocidad)**,** así como código fuente unificado.

## 2.2. Bases de Datos NoSQL

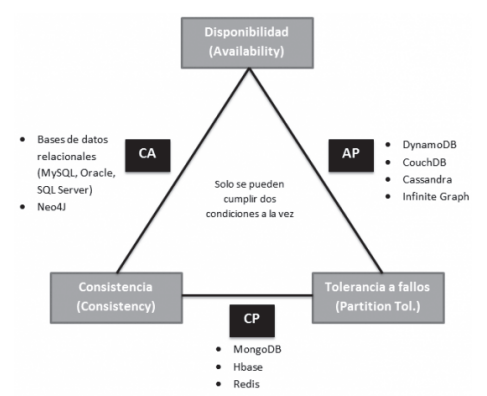
Una base de datos NoSQL (*Not Only SQL*) es un tipo de base no relacional, concebida para el manejo actual de la información en el marco de los grandes volúmenes de datos a gran velocidad y en un ambiente distribuido. (Castro & Gonzalez, 2012) justifican el uso de base de datos NoSQL al constante crecimiento de la información disponible y a la demanda sin precedentes de los usuarios por consumirla y explotarla.

### 2.2.2 Características de una BDD NoSQL y CAP

* **Escalabilidad horizontal:** Facilidad para incorporar nuevos nodos en un sistema distribuido para mejorar el rendimiento.
* **Habilidad de distribución:**  Característica de las bases NoSQL para hacer réplicas de información en los nodos que la conforman, presentando datos descentralizados.
* **Libertad de Esquema:**  No existe restricción sobre la data que ingresa, facilitando su modelado.
* **Simplicidad:** Las consultas resultan más eficientes.

Las bases de datos en general están regidas por el teorema CAP o de Weber, el cual expresa que a fin de que una base de datos distribuida sea veloz y flexible, debe sacrificar al menos una de las características relativas a Disponibilidad, Consistencia y Tolerancia a Fallos Parciales, siendo imposible conservar todas de forma simultánea. La Figura 4 muestra las tres características que conforman el teorema de CAP y ejemplos de bases de datos que se acoplan a dos de ellas.

Figura 4 Teorema CAP



**Fuente**: Manjarrez Antaño, Ángeles: Migración de Bases de Datos SQL a NoSQL

### 2.2.3 Tipos de Bases NoSQL

* **Clave-Valor:**  Base de datos cuyo registro está definido por una clave (llave única) asociada a un valor (generalmente una cadena), siendo simples, pero altamente eficientes para procesos que requieren velocidad en las consultas o registros altamente cambiables. Operaciones como junturas o agregaciones están restringidas. *Ejemplos*: Redis, DynamoDB
* **Columnares:**  A diferencia de las bases de datos SQL que están orientadas a almacenar los registros mediante filas, su organización de tipo columnar no almacena la data en tablas sino en arquitecturas masivamente distribuidas cuya clave se asocia con varios atributos. Una base columnar responde a escenarios que demandan un análisis (agregación) o minería de grandes volúmenes de datos. *Ejemplos:* Cassandra, BigTable.
* **Orientada a documentos:**  Bases de datos similares a las de tipo Clave-Valor, cuya diferencia es que esta se almacena en forma de documentos de tipo json o xml más complejos. Debido a su naturaleza de libre esquema, estas bases de datos son aptas para aplicaciones cuya data no esté normalizada o altamente relacionada. Un documento ofrece alta escalabilidad horizontal y rendimiento. *Ejemplos*: MongoDB, CouchDB, Elasticsearch.
* **Orientada a Grafos:** Almacenan los datos en forma de un grafo, compuesto por nodos (objetos) y vértices (relaciones). Su principal objetivo es encontrar relaciones existentes entre los datos. *Ejemplo:* Neo4j.

## 2.3. Visualización de datos para la toma de decisiones

De acuerdo a (Burnay, Dargam, & Zarate, 2019), en la sociedad actual, diariamente líderes y administradores de cualquier compañía tienen el desafío de tomar decisiones, las mismas que resultan complejas bajo el escenario de contar con datos distribuidos y poco estructurados. Hoy en día, alguien que tome decisiones, difícilmente tendrá éxito en su propósito si estas se basan en la intuición y no en los propios datos que genera la organización.

Gracias a las tecnologías de la Información como la Inteligencia de Negocios (*Business Intelligence*) o el *Big Data*, los procesos de toma de decisiones resultan más fáciles y eficientes ya que a través de la consolidación de los datos y la generación de conocimiento, que es accesible empleando herramientas de visualización, se tiene una mejor compresión de la empresa y su entorno.

Las herramientas de visualización de datos brindan un soporte importante para resolver un problema del negocio. Sin embargo, estas tienen como principal desafío el ser capaces de comunicar con claridad los resultados, despertando el interés y reflexión en el usuario para obtener conclusiones y estrategias válidas en la toma de decisiones. (Burnay, Dargam, & Zarate, 2019) resalta el rol principal del diseño de reportes, que es proveer de información útil y no en crear gráficos estéticamente llamativos e irrelevantes que pueden conducir a la confusión e incomprensión.

En ese sentido, (Moore, 2017) describe los principios para crear gráficos efectivos que coadyuven a una rápida interpretación de los datos representados. La Tabla 1 detalla algunos de esos principios y su explicación para la generación de gráficos efectivos.

Tabla 1 Principios de un Gráfico Efectivo

|  |  |
| --- | --- |
| **Principio** | **Explicación** |
| Principio de Compatibilidad de Aproximación | Las tareas más integradas (relacionadas) se facilitan por gráficos con alta proximidad visual, mientras que tareas más aisladas o independientes son facilitadas por gráficas con baja proximidad visual. |
| Principio de Relevancia de Gráficos | Presentar únicamente la información que se necesita, ni más ni menos. |
| Principio de aprehensión y discriminación | Un gráfico visual tiene que ser exactamente percibido. Uso de animaciones visuales que sean comprensibles y dimensiones visuales que sean juzgadas con precisión.  Gráficos visuales que muestren la diferencia de dos variables deben ser lo suficientemente claros para que muestren esa diferencia. |
| Principio de compatibilidad | Los gráficos deben ser compatibles con la información que se representa |
| Principio de prominencia y cambio de información | Diseños gráficos que se enfoquen en la información más relevante.  Evitar grandes cambios en propiedades de un gráfico que no tenga información. |
| Principio de conocimiento apropiado | Asegurar que el usuario tenga el suficiente conocimiento para entender la data representada. |
| Principio del momento visual | Desarrollar gráficos de una forma consistente y proveer a los usuarios de ayudas para hacer conexiones referenciales entre las gráficas, evitando la desorientación en visualizaciones animadas e interactivas. |

**Fuente:** Traducido y adaptado de Jeanne Moore, Data Visualization in Support of Executive Decision Making

En adición a los criterios expresados, los usuarios buscan que los gráficos sean claros y precisos con una visualización desde varios puntos de vista de una forma dinámica, presentando los datos con criterios de integridad, accesibilidad y velocidad.

## 2.4. Herramientas de aprendizaje automático para clusterización: Scikit-Learn

Python, unos de los lenguajes más populares en el mundo de la computación científica, ofrece un conjunto de módulos para el desarrollo de algoritmos y análisis de datos (Pedregosa, Varoquaux, & Gramfort, 2011), cuenta con el módulo Scikit-Learn, el cual integra un amplio conjunto de algoritmos de *Machine Learning* para la resolución de problemas supervisados y no supervisados.

Scikit-Learn es un paquete de propósito general, basado en las librerías como Numpy (estructura de datos), Spicy (algebra lineal) y Cython (conectividad con C), ofreciendo soluciones para escenarios comerciales y académicos. Scikit-Learn cuenta con las siguientes características:

* Calidad de código y simplicidad
* Soporte comunitario
* Documentación oficial
* Libertad para su uso
* Multiplataforma (Windows, Linux)

Entre de los modelos que Scikit-Learn ofrece soporte se encuentran los algoritmos No Supervisados, quienes a través de un grupo de datos no etiquetados buscan una apropiada representación de su distribución. Dentro de estos se encuentra la clusterización, cuya misión es limitar dentro de un clúster aquellas instancias cuyas características o atributos son similares, a la vez que estas deben estar separadas del resto de instancias.

Uno de los métodos de clusterización más conocidos es K-Means, el cual se basa en un grupo inicial de centroides, más adelante se procede con un ciclo repetitivo donde se asigna cada instancia al punto más cercano, y se recalcula los nuevos centroides, tomando la media de todas las instancias asignadas a los anteriores centroides; este proceso se da hasta alcanzar la convergencia (los nuevos centroides no cambian con respecto a los anteriores).

## 2.5. Casos de Uso

Tomando como punto de partida la literatura citada previamente, donde se detalla arquitecturas y tecnologías existentes para la captura, análisis y visualización de datos, el siguiente apartado describe algunos casos de uso reales de soluciones similares a la arquitectura propuesta en este TFM:

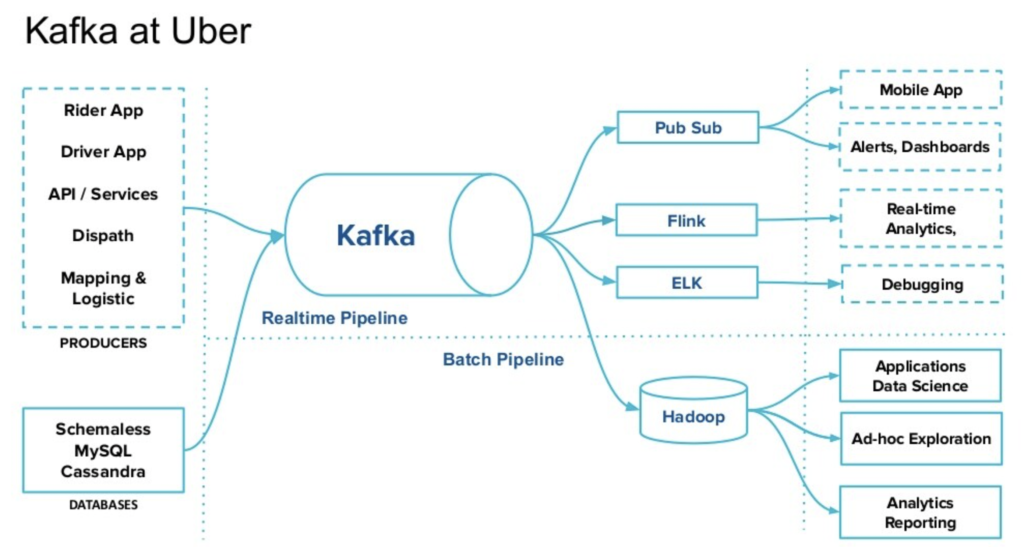
* Arquitectura Kappa para soluciones *Big Data* en tiempo real, con Kafka como su actor principal.
* Procesamiento de datos masivos a través de Apache Spark
* Persistencia y reportes en tiempo real con ELK (**E**lasticsearch, **L**ogstash y **K**ibana)
* Algoritmos de aprendizaje de máquina mediante Python: Scikit-learn.

### 2.5.1 Arquitectura Kappa: Uber

Uber, la prominente empresa americana de movilidad, publica con frecuencia las tecnologías que utiliza para solventar sus necesidades de negocio, entre ellas destaca la implementación de la arquitectura Kappa para el procesamiento de más de 3PB de datos y 4 trillones de mensajes por día.

La Figura 5, describe la arquitectura empleada por Uber, siendo el sistema de mensajes en tiempo real Kafka el principal autor a la hora de captura los datos provenientes de distintos aplicativos y su posterior consumo en procesos de análisis, alertas, reportes.

Figura 5 Uber y Kafka



**Fuente:** https://www.kai-waehner.de/

Kafka trabaja bajo un modelo Productor/Suscriptor, siendo para el caso de Uber los productores, aquellas fuentes de datos concernientes a los aplicativos del conductor, usuario, servicios API, logística, bases de datos (relacionales y NoSql) , e/o; esta información viaja por el bus de Kafka para que los suscriptores, entre estos plataformas para la analítica y procesamiento de datos en tiempo real como Apache Flink , ELK ( **E**lasticsearch, **L**ogstash y **K**ibana) habiliten la construcción de cuadros de mando, sistemas de alertas y *logs*, o incluso la ingesta de los datos resultantes a un servicio de Publicación/Suscripción para su uso desde aplicativos móviles. Adicionalmente, se cuenta con un flujo de datos en lotes, para la persistencia de los datos en un *data lake* (Hadoop HDFS), utilizado para la implementación de aplicaciones de ciencia de datos y reportes analíticos o de Inteligencia de Negocios.

Uber hace referencia de muchos desafíos para que su plataforma cumpla con principios como eficiencia, rendimiento, confianza y recuperación ante fallos. Entre las medidas de contingencia y continuidad aplicadas se describe la tenencia de clústeres Kafka en diferentes regiones, que permiten la replicación de los datos entrantes, y procesos para que los consumidores de una región de Kafka afectada sepan cómo y cuándo consumir de un clúster réplica ubicado en otra región en caso de indisponibilidad.

### 2.5.2 Apache Spark: Casos de uso para el procesamiento de datos

De acuerdo con el portal (ProjectPro, 2022), Apache Spark ha sido adoptado por un sinnúmero de compañías para la implantación de soluciones analíticas *Big Data*. Entre los casos de uso se detallan:

* **Industria Financiera**: Para el análisis de perfiles, correos electrónicos e historial crediticio, e/o; Spark es utilizado por los bancos en sus procesos de análisis de riesgo de historial crediticio, segmentación de clientela. Un ejemplo bien conocido es el análisis en tiempo real de presuntos fraudes al realizar transacciones con tarjetas de crédito, para ello se comprueba la transacción entrante con data histórica de bases de datos, junto a modelos previamente entrenados; al final en caso de existir probables anomalías el operador se comunicará con el cliente inmediatamente.
* **Comercio electrónico**: Shopify requirió de Spark para realizar procesos de minería con el propósito de identificar posibles tiendas socias. En contraste al uso de tecnologías convencionales sobre su base de datos, cuyo tiempo fue considerable, Spark permitió el análisis de más de 67 millones de registros en pocos minutos.

EBay, otra importante tienda electrónica, utiliza Spark para instaurar procesos de recomendación que mejoren la experiencia de usuario. EBay cuenta con un clúster Spark de más de 2000 nodos y 100TB de memoria RAM, los cuales son gestionados a través del gestor YARN.

Pinterest, que permite la creación de tableros personales, repositorios de imágenes, eventos o incluso almacenar sus gustos, aborda Spark para realizar análisis en tiempo real sobre el grado de compromiso o pertenencia que tiene los usuarios con los *Pins* a fin de construir sistemas de recomendación que les permitan seleccionar qué productos comprar o un plan de viaje a múltiples destinos.

* **Cuidado de la salud:**  Instituciones como hospitales o proveedores de servicios sanitarios cuentan con Spark para el análisis de historiales médicos, esto con el objetivo de escatimar costos a la hora de dar de un alta a un paciente y conocer si este tendrá o no recaídas. Otro caso de aplicación es el análisis del genoma humano, cuya data procesada solía tomar semanas, ahora Spark permite su consecución en unas pocas horas.
* **Industria del entretenimiento:** Yahoo, emplea Spark para sistemas de recomendación y la personalización de páginas web, de esta forma un usuario tendrá acceso a páginas cuyos contenidos corresponden a sus gustos o intereses. Previamente, un recomendador contenía miles de líneas de código en lenguaje C+, hoy en día con Spark sobre el lenguaje Scala, el código resultante se ha reducido a 120 líneas de código, cuya ejecución es sobre aproximadamente 100 millones de registros.

Estos casos de uso real son una muestra de la versatilidad de Spark que aborda todo tipo de industrias; para este TFM se han citado estos ejemplos a fin de resaltar características como el procesamiento de grandes volúmenes de datos en tiempo real e históricos, así como la optimización y reducción de código, que mejora drásticamente los procesos de mantenimiento y corrección de errores si los hubiera.

### 2.5.3 ELK (Elasticsearch, Logstash y Kibana) para el análisis y reportes

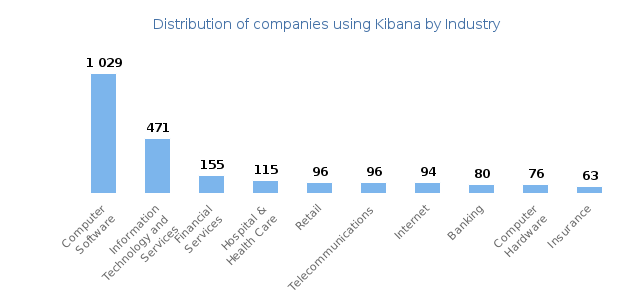
De acuerdo con el portal de Elastic, ELK hace referencia a la unión de tres proyectos de código abierto: ElasticSearch, un motor de búsqueda y análisis de datos; Logstash, herramienta para el procesamiento y de múltiples fuentes de datos a través de tuberías (*pipelines*) y Kibana, para la visualización de cuadros de mando.

Entre las aplicaciones de ELK se encuentran el análisis de *logs*, la generación de búsqueda de documentos basados en texto y la generación de métricas. A continuación, se describe algunos ejemplos reales de aplicación:

* **Análisis de *Logs*:** Mediante la consolidación y centralización de los datos de *logs* y eventos generados por aplicativos y la posterior identificación de errores. Reconocidas compañías como Facebook, Cisco y Netflix incorporan ELK en sus procesos, esta última cuenta con más de 85 clústeres para el monitoreo y análisis de operaciones de servicio al cliente y *logs* de seguridad.
* **Motores de búsqueda**: ElasticSearch, cuyo desarrollo se base en el lenguaje java a partir del API de recuperación de texto Lucene, incluye múltiples funciones para la búsqueda de texto similares a la de un navegador de Google. Entre las empresas que hacen uso de estas herramientas se encuentran GitHub, Wikipedia o Amazon que requieren de tiempos excesivamente cortos de respuesta por parte de los usuarios. Otra aplicación se encuentra en el mundo del desarrollo tecnológico, con el *scrapping,* donde los programadores hacen búsquedas masivas en tiempo real de fragmentos de código de páginas web; o el caso de ingenieros de infraestructura, a la hora de resaltar los *logs* más importantes para resolver un problema.
* **Análisis de Métricas:** Kibana, una plataforma de visualización para la toma de decisiones trabaja sobre el motor de búsqueda ElasticSearch, con ello se logra la creación de distintos y variados reportes, que cuentan con funcionalidades como navegación iterativa, análisis multidimensional a través de distintos tipos de gráficos. Esto conlleva un mejor enfoque sobre los datos a la hora de ejecutar procesos de toma de decisiones.

La Figura 6, describe las distintas industrias cliente de Enlyft que utilizan Kibana en proyectos de visualización, se observa que las industrias del desarrollo de software y servicios de TI representan aproximadamente el 66% del total con 1500 empresas.

Figura 6 Distribución de Compañías usando Kibana en la Industria



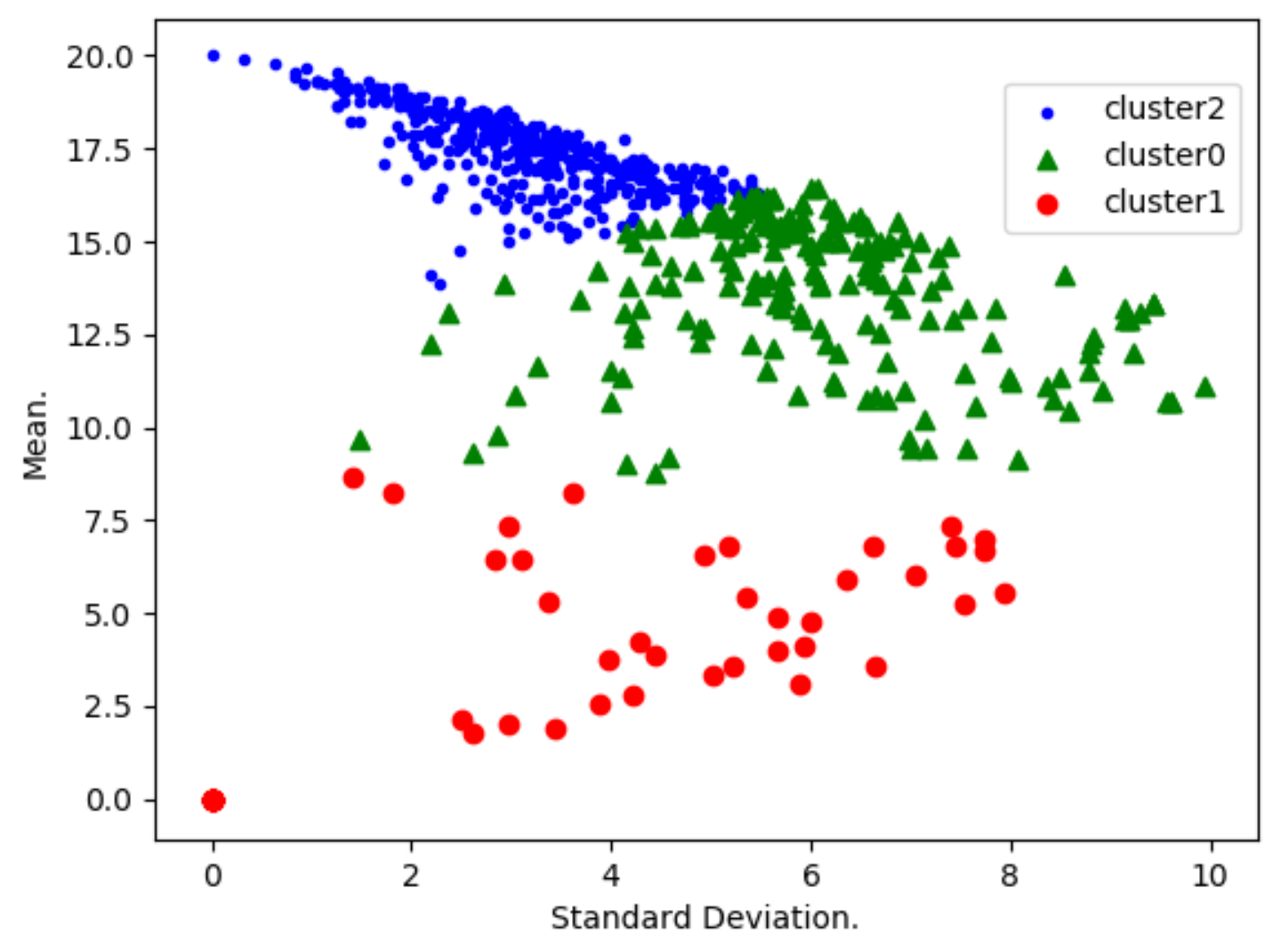
**Fuente:** https://enlyft.com/tech/products/kibana

### 2.5.4 Scikit-Learn en la Educación

Un estudio publicado por (Tuyishimire, Mabuto, & Gatabazi, 2022) ,emplea el lenguaje de programación Python y su librería Scikit-learn para detectar aquellos grupos de estudiantes de la Universidad de Johannesburgo, con problemas de aprendizaje o que requieren motivación con el propósito de mejorar sus calificaciones.

La Figura 7 describe las tres categorías identificadas a partir de la ejecución de un algoritmo de K-Means para 703 instancias (estudiantes) tomando en consideración la media y la desviación estándar de las evaluaciones (10 en total) realizadas por cada estudiante; siendo estas: Jóvenes con la más alta motivación para estudiar (Clúster 2); aquellos motivados (Clúster 0), y el grupo de estudiantes desmotivados (Clúster 1).

Figura 7 Clusterización de estudiantes por Media y Desviación



**Fuente:** https://www.mdpi.com/

Esta investigación en el ámbito académico revela la facilidad de la herramienta Scikit-learn a la hora de implementar modelos de aprendizaje automático que permitan la toma de decisiones de una efectiva y oportuna; esta última debido a que la clusterización se desarrollaba de forma permanente, es decir a partir de cada evaluación se recalculaba los segmentos de estudiantes. Para este caso de uso, el desarrollo tecnológico involucró la constante comunicación de los resultados obtenidos a los tutores y demás autoridades educativas a fin de instaurar medidas correctivas.

**Otros Testimonios Scikit-Learn**

El sitio oficial Scikit-learn publica testimonios de varias empresas que hacen uso de esta versátil librería de *Machine Learning* en cada uno de sus procesos de negocio.

* **Spotify:** Implementación de sistemas de recomendación de música.
* **Betaworks:** Compañía americana dedicada a la inversión y capital de riesgo. Entre sus usos se encuentran los módulos para sistemas de recomendación y clusterización.
* **Booking.com:**  Emplea algoritmos de aprendizaje de máquina para sistemas de recomendación en destinos y hoteles, detección de reservas fraudulentas y calendarización de agentes de servicio al cliente.
* **Inria:** Institución pública francesa dedicada a la investigación, emplea Scikit-learn para neuroimagen, computación visual, análisis de imágenes médicas y seguridad.
* **Hugging Face:**  Un proveedor de servicios de Inteligencia Artificial, utiliza Scikit-learn para la construcción de redes neuronales y modelos probabilísticos que simulan un servicio chat.

### 2.6 Contribución

El Trabajo de Fin de Máster en su fase de implementación engloba muchos de los conceptos y herramientas tratadas en el Estado de Arte, se hace especial énfasis en la incorporación de la arquitectura Kappa para la captura, proceso y análisis de grandes volúmenes de datos, con Kafka como parte central para capturar los datos desde sistemas transaccionales, Apache Spark para su procesamiento, y ELK para la persistencia y reportes en tiempo real. Finalmente, Python. Scikit-learn para crear entrenar modelos de clusterización.

Si bien es cierto todas estas tecnologías se encuentran ampliamente utilizadas por muchas compañías de renombre mundial, con grandes equipos tecnológicos que de forma permanente buscan procesos de mejora y optimización; este TFM busca lo siguiente:

* Implementar una solución *Big Data* empleando herramientas *Open Source* con equipos *commodity,* que de alguna forma demuestre y visibilice a las tecnologías *Big Data* como una poderosa alternativa frente a un abanico de problemas reales de cualquier tipo de industria.
* En los subsiguientes capítulos se describirá la arquitectura, en ella se plantea el modelo que bosqueja todo el ciclo de vida del dato: naciendo desde una base de datos transaccional, cuyos datos viajan por herramientas de ingesta, procesamiento y persistencia en tiempo real o casi real; para finalizar se llega al diseño de reportes y modelos de *Machine Learning.*
* En a la introducción de esta memoria, se hizo énfasis en la problemática de no contar con información en tiempo real para la toma de decisiones; la pandemia del COVID-19 tuvo un impacto no solo en el marco sanitario, sino también en el social y económico; uno de los elementos de este TFM será el uso de modelos no supervisados de clusterización para segmentar las localidades y sus distintas realidades, en el contexto de declaraciones en los años 2020-2021.

# 3. Objetivos concretos y metodología de trabajo

## 3.1. Objetivo general

Desarrollar e implementar una arquitectura para capturar y procesar la información en tiempo real de datos de declaración y su consolidación con la data histórica, en el periodo 2020-2022, para el desarrollo de un reporte analítico que permita la toma de decisiones.

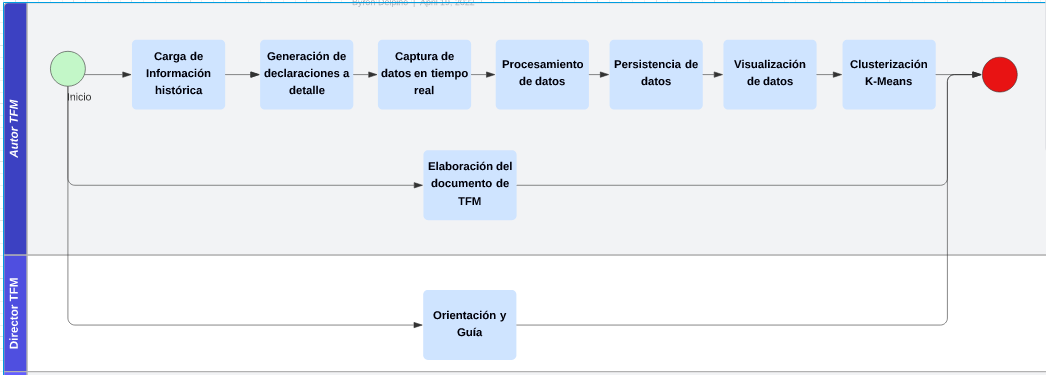
## 3.2. Objetivos específicos

* Almacenar en una base de Datos no relacional la información histórica y agregada de declaraciones tributarias en el periodo 2020-2021.
* Diseñar un modelo que permita la generación de datos ficticios de declaraciones a detalle del año 2022 y su almacenamiento en un base de datos transaccional.
* Implementar un sistema de encolamiento que capture y procese los datos almacenados en la base de datos transaccional y sean procesados por una herramienta *Big Data*.
* Consolidar la información histórica y la detalle y su persistencia en una base de datos no relacional.
* Construir un reporte de tipo analítico de información tributaria de los datos consolidados.
* Implementar modelos de clusterización de cantones del Ecuador con características similares de tributación.

## 3.3. Metodología del trabajo

Para el desarrollo de este apartado se ilustra la Figura 8, que define las etapas involucradas para el desarrollo de este proyecto y los involucrados en su consecución; se distingue el Autor del TFM, quien implementa la solución a la propuesta, así como la redacción del documento memoria; y el Director, quien da el acompañamiento y retroalimentación a las entregas realizadas.

Figura 8 Metodología de Trabajo



**Carga de Información histórica:** Los archivos públicos sri\_ventas\_2020.csv y sri\_ventas\_2021.csv que responden a la información agregada de declaraciones (compras y ventas) del periodo 2020-2021. Estos datos serán cargados a una base de datos no relacional. Esta fase cubre el análisis (diccionario de datos), selección, tratamiento y consolidación (cruce con otros catálogos de datos) de los campos constituyentes. **Tecnologías Aplicadas:** ElasticSearch junto a Logstash.

**Generación de declaraciones a detalle:**  La generación de *datasets* a detalle corresponde a información ficticia; para su consecución se implementará procesos PLSQL sobre una base de datos que corresponde a la base de datos transaccional. Los datos generados guardarán cierta relación (a nivel estructural) con la data histórica agregada. Con el objetivo de visualizar el funcionamiento de los *dashboard* y ver como la data entrante modifica los reportes se desarrollará una interfaz gráfica que pondere la carga de datos ficticios por provincia. **Tecnologías Aplicadas:** JavaSE,Base de datos Oracle

**Captura de datos en tiempo real:**  Para la captura de los datos que se ingesta en la base de datos relacional, se utilizará un conector de Kafka que realice un *pull* de los nuevos datos hacia el sistema de mensajería distribuida para su posterior consumo. **Tecnología Aplicada:** Apache Kafka.

**Procesamiento de datos:**  Los datos entrantes de Kafka cuyo *topic* se relaciona a las declaraciones a detalle, serán consumidos por una herramienta de cálculo distribuido, quien consolidará estos datos en *streaming* junto al *dataset* histórico almacenado en ElasticSearch. **Tecnología Aplicada:** Apache Spark.

**Persistencia de datos**: Los nuevos cálculos (agregaciones) efectuados sobre la consolidación de los datos serán guardados/actualizados en la base de datos. **Tecnología Aplicada:**  ElasticSearch.

**Visualización de datos:** Se construirán *dashboards* que permitan visualizar la información agregada por ubicación geográfica (Provincia, Cantón) y sector económico. Los *dashboards* deberán tener un refrescamiento periódico para verificar la volatilidad de los datos. **Tecnología Aplicada:**  Kibana.

**Clusterización:**  La información agregada pública, al ser data real, será empleada para crear modelos de clusterización con el método de K-Means y Aglomerativo para encontrar los clústeres por provincia. **Tecnología Aplicada:**  Python, Scikit-Learn.

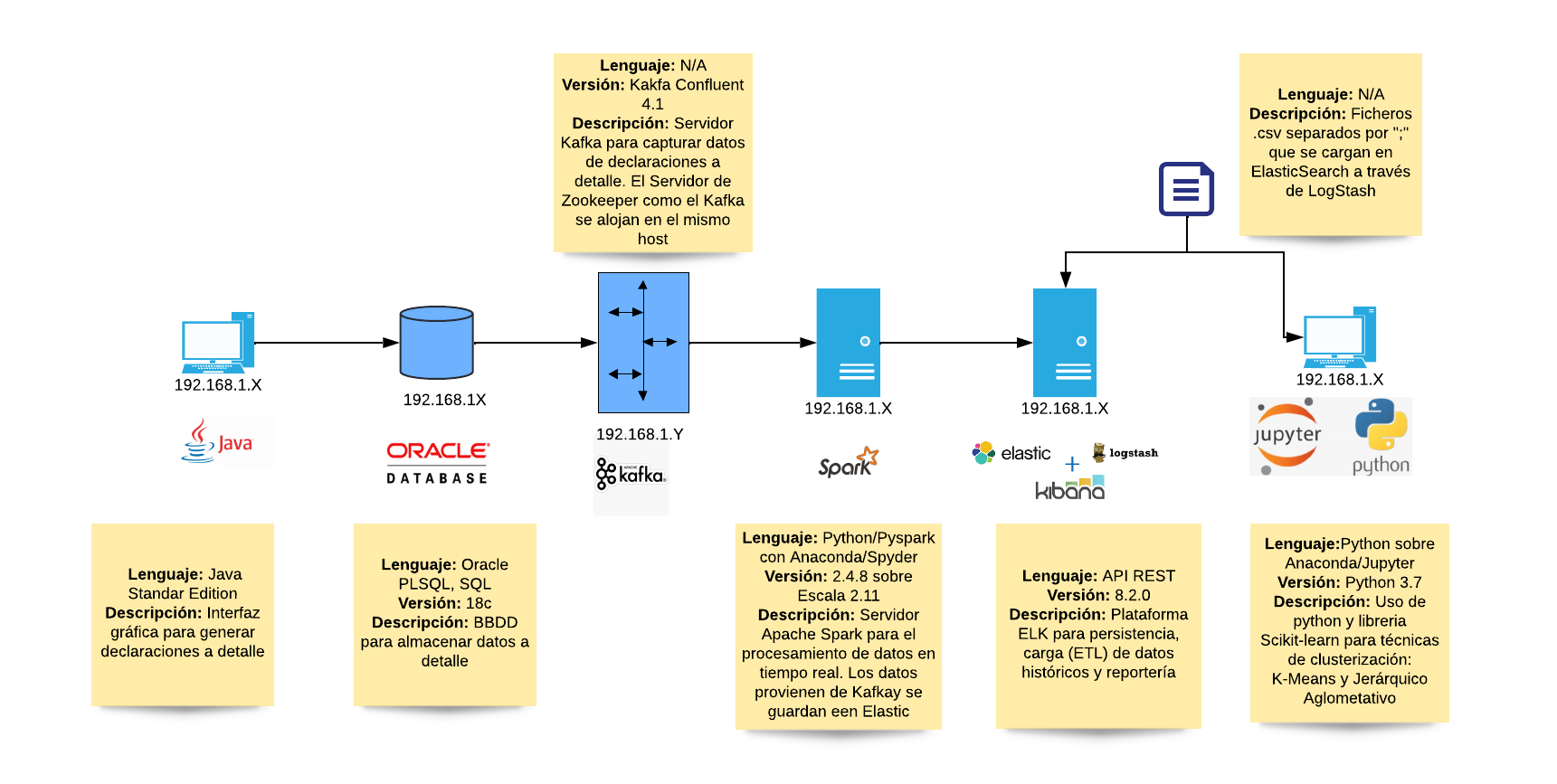
**Elaboración del documento de TFM:**  Los pasos anteriores concernientes a la implementación del modelo de Captura y Procesamiento de datos de Declaraciones Tributarias serán debidamente documentos y justificados.

**Orientación y Guía:**  El Director del TFM quien realiza las tareas de acompañamiento y orientación a través de reuniones de trabajo y retroalimentación respectiva, proporciona una guía para que el estudiante profundice y mejore aspectos relativos a su Proyecto Final de Máster.

# 4. Desarrollo específico de la contribución

Para el desarrollo del presente TFM se bosqueja la siguiente topología descrita en la Figura 9, la misma que detalla las tecnologías aplicadas, lenguajes de programación y sus versiones, así como una corta descripción de su funcionalidad:

Figura 9 Topología de declaraciones



La Tabla 2 describe las características de hardware, software y versiones de JDK de los equipos utilizados para el desarrollo:

Tabla 2 Descripción equipos utilizados

|  |  |  |
| --- | --- | --- |
| **EQUIPO** | **DIRECCION IP REAL** | **CARACTERÍSTICAS** |
| 192.168.1.Y | 192.168.1.53 | **Procesador:** Core i9, 11va generación  **Memoria Ram:** 16GB  **Disco Duro:** 1TB  **Sistema Operativo:** Windows 10 Home  **Versiones JDK:**  **1.8:** OpenJDK sobre Anaconda para Apache Spark  **18:** Para Interfaz gráfica, Oracle, ELK |
| 192.168.1.X | 192.168.1.47 | **Procesador:** Core i5, 3ra generación  **Memoria Ram:** 6GB  **Disco Duro:** 256GB  **Sistema Operativo:** Ubuntu 18 LTS  **Versión JDK:** 1.8 |

Se adjunta el Anexo VI, donde se detalla información del repositorio que contiene los códigos fuentes para la implementación de este TFM; así como un video que refleja las pruebas de ejecución para la captura de información de detalle en tiempo real y su despliegue en cuadros de mando.

# 4.1. Carga de Información Histórica

### 4.1.1 Modelo de Datos

El Servicio de Rentas Internas publica en el portal de datos abiertos https://www.sri.gob.ec/datasets , información de forma consolidada y anonimizada de declaraciones, que corresponde a las compras y ventas efectuadas durante un año. La Tabla 3 detalla el diccionario de datos de declaraciones: nombre de variables, descripción y tipo:

Tabla 3 Diccionario de datos de declaraciones

|  |  |  |
| --- | --- | --- |
| **Nombre de la variable** | **Definición de la variable** | **Formato del dato** |
| Año | Se refiere al año en el que fueron generados los valores. | Alfanumérico |
| Mes | Se refiere al mes en el que fueron generados los valores. | Alfanumérico |
| Codigo\_Sector\_N1 | Contiene la clasificación de los sectores según CIIU 4.0 Nivel 1. | Alfanumérico |
| Provincia | División político-administrativa conformada por la unión de dos o más cantones en donde fue recaudado el tributo. | Alfanumérico |
| Cantón | División político-administrativa conformada por la unión de dos o más parroquias en donde fue recaudado el tributo. | Alfanumérico |
| Ventas netas tarifa 12% | Valor registrado de todas las operaciones gravadas con tarifa diferente de cero | Numérico |
| Ventas netas tarifa 0% | Valor registrado de todas las operaciones gravadas con tarifa cero | Numérico |
| Exportaciones | Corresponde a las exportaciones de bienes y servicios | Numérico |
| Compras netas tarifa 12% | Corresponde a las compras y adquisiciones declaradas por el propio contribuyente con tarifa diferente de cero | Numérico |
| Compras netas tarifa 0% | Corresponde a las compras y adquisiciones declaradas por el propio contribuyente con tarifa cero | Numérico |
| Importaciones | Corresponde a las importaciones de bienes y servicios | Numérico |
| Compras\_Rise | Corresponde a las compras realizadas a los contribuyentes inscritos en el Régimen Simplificado (RISE) | Numérico |
| Total\_Compras | Corresponde al total compras y adquisiciones declaradas por el propio contribuyente | Numérico |
| Total\_Ventas | Valor registrado de todas las operaciones gravadas | Numérico |

**Fuente:** Adaptado a partir de la tabla original publicada por el SRI

El campo Codigo\_Sector\_N1 corresponde a un código clasificador de las actividades económicas registradas en Ecuador publicadas por el Instituto Nacional de Estadísticas y Censos (INEC); https://aplicaciones2.ecuadorencifras.gob.ec/SIN/descargas/ciiu.pdf , las mismas que se catalogan por niveles (1-6). En el catálogo de datos de declaraciones se encuentra información de actividades económicas de nivel 1; la Tabla 4 define la descripción de actividad por cada tipo de actividad económica.

Tabla 4 Definición de Actividades Económicas en Ecuador

|  |  |
| --- | --- |
| **CODIGO\_**  **SECTOR\_N1** | **DESCRIPCION** |
| A | AGRICULTURA, GANADERÍA, SILVICULTURA Y PESCA. |
| B | EXPLOTACIÓN DE MINAS Y CANTERAS. |
| C | INDUSTRIAS MANUFACTURERAS. |
| D | SUMINISTRO DE ELECTRICIDAD, GAS, VAPOR Y AIRE ACONDICIONADO. |
| E | DISTRIBUCIÓN DE AGUA; ALCANTARILLADO, GESTIÓN DE DESECHOS Y ACTIVIDADES DE SANEAMIENTO. |
| F | CONSTRUCCIÓN. |
| G | COMERCIO AL POR MAYOR Y AL POR MENOR; REPARACIÓN DE VEHÍCULOS AUTOMOTORES Y MOTOCICLETAS. |
| H | TRANSPORTE Y ALMACENAMIENTO. |
| I | ACTIVIDADES DE ALOJAMIENTO Y DE SERVICIO DE COMIDAS. |
| J | INFORMACIÓN Y COMUNICACIÓN. |
| K | ACTIVIDADES FINANCIERAS Y DE SEGUROS. |
| L | ACTIVIDADES INMOBILIARIAS. |
| M | ACTIVIDADES PROFESIONALES, CIENTÍFICAS Y TÉCNICAS. |
| N | ACTIVIDADES DE SERVICIOS ADMINISTRATIVOS Y DE APOYO. |
| O | ADMINISTRACIÓN PÚBLICA Y DEFENSA; PLANES DE SEGURIDAD SOCIAL DE AFILIACIÓN OBLIGATORIA. |
| P | ENSEÑANZA. |
| Q | ACTIVIDADES DE ATENCIÓN DE LA SALUD HUMANA Y DE ASISTENCIA SOCIAL. |
| R | ARTES, ENTRETENIMIENTO Y RECREACIÓN. |
| S | OTRAS ACTIVIDADES DE SERVICIOS. |
| T | ACTIVIDADES DE LOS HOGARES COMO EMPLEADORES; ACTIVIDADES NO DIFERENCIADAS DE LOS HOGARES COMO PRODUCTORES DE BIENES Y SERVICIOS PARA USO PROPIO. |
| U | ACTIVIDADES DE ORGANIZACIONES Y ÓRGANOS EXTRATERRITORIALES. |
| V | SIN ACTIVIDAD ECONOMICA – CIIU |
| W | BAJO RELACION DE DEPENDENCIA SECTOR PRIVADO |
| X | BAJO RELACION DE DEPENDENCIA SECTOR PUBLICO |

**Fuente:** Adaptado a partir de la tabla original publicada por el INEC

### 4.1.2 Importación de Datos Históricos

Para la importación de datos históricos dentro de una base de datos NoSQL así como la subsecuente persistencia de datos a detalle autogenerados y cuadros de mando en tiempo real se hace uso de ELK (ElasticSeach- LogStash- Kibana). Para su instalación y configuración se adjunta el Anexo II.

En esta sección corresponde la creación de un índice en ElasticSearch que representa una tabla, la importación de los datos históricos (archivos .csv) de las declaraciones del periodo 2020-2021 mediante LogStash y una representación gráfica con Kibana que despliegue un resumen sobre los datos cargados.

* **CREACIÓN DE ÍNDICE**

La creación del índice que almacenará los datos de declaraciones es a través de conjuntos clave: valor en formato json con información sobre:

* **Metadata “\_meta\_”:**  Metadatos sobre el índice a crear, el autor, e/o.
* **Campos “properties”:**  Campos que constituyen el índice, con el nombre y el tipo; para el índice declaraciones se tiene campos tipo keyword y long, que representan datos de tipo alfanumérico y numérico decimal respectivamente.
* **\_id:** El \_id representa un identificador único (clave primaria) del registro cuyo valor en este caso es autogenerado.

El índice será creado con la sentencia put /declaraciones cuya estructura se definirá dentro de la etiqueta **mappings**:

put /declaraciones

{

"mappings" : {

"\_meta" : {

"created\_by" : "Byron Del Pino"

},

"properties" : {

"ANIO" : {

"type" : "keyword"

},

"MES" : {

"type" : "keyword"

},

"CODIGO\_SECTOR\_N1" : {

"type" : "keyword"

},

"PROVINCIA" : {

"type" : "keyword"

},

"CANTON" : {

"type" : "keyword"

},

"VENTAS\_NETAS\_12" : {

"type" : "float"

},

"VENTAS\_NETAS\_0" : {

"type" : "float"

}

,"EXPORTACIONES" : {

"type" : "float"

},

"COMPRAS\_NETAS\_12" : {

"type" : "float"

},

"COMPRAS\_NETAS\_0" : {

"type" : "float"

},

"IMPORTACIONES" : {

"type" : "float"

},

"COMPRAS\_RISE" : {

"type" : "float"

},

"TOTAL\_COMPRAS" : {

"type" : "float"

},

"TOTAL\_VENTAS" : {

"type" : "float"

}

}

}

}

}

* **CARGA DE DATOS CON LOGSTASH**

LogStash permite la conexión a múltiples fuentes de datos, entre ellos destaca la creación de procesos de Extracción, Transformación y Carga (ETL) de datos almacenados en bases de datos, ficheros planos, .json, e/o.

Para la carga de datos de los archivos: sri\_ventas\_2020.csv y sri\_ventas\_2021 se ejecutaron las siguientes acciones:

**Configuración archivos de carga:**  Se crearon dos archivos .conf que contiene parámetros de configuración sobre la fuente de datos: ruta de origen, codificación; transformaciones intermedias (*filter*): separador, columnas que se leerán del archivo, conversiones (a *float*), y el destino, en este caso ElasticSearch: dirección IP, usuario, clave e índice donde se carga los *datasets.*

input {

file {

path => "<RUTA\_LOGSTASH>/config/sri\_ventas\_202**X**.csv"

start\_position => beginning

sincedb\_path => "NULL"

codec => plain { charset=>"UTF-8" }

}

}

filter {

csv {

columns => [

"ANIO",

"MES",

"CODIGO\_SECTOR\_N1",

"PROVINCIA",

"CANTON",

"VENTAS\_NETAS\_12",

"VENTAS\_NETAS\_0",

"EXPORTACIONES",

"COMPRAS\_NETAS\_12",

"COMPRAS\_NETAS\_0",

"IMPORTACIONES",

"COMPRAS\_RISE",

"TOTAL\_COMPRAS",

"TOTAL\_VENTAS"

]

separator => ";"

convert => {

"VENTAS\_NETAS\_12" => float

"VENTAS\_NETAS\_0" => float

"EXPORTACIONES"=> float

"COMPRAS\_NETAS\_12"=> float

"COMPRAS\_NETAS\_0"=> float

"IMPORTACIONES"=> float

"COMPRAS\_RISE"=> float

"TOTAL\_COMPRAS"=> float

"TOTAL\_VENTAS"=> float

}

}

} output {

elasticsearch {

hosts => ["http://localhost:9200"]

index => "declaraciones"

} stdout { }

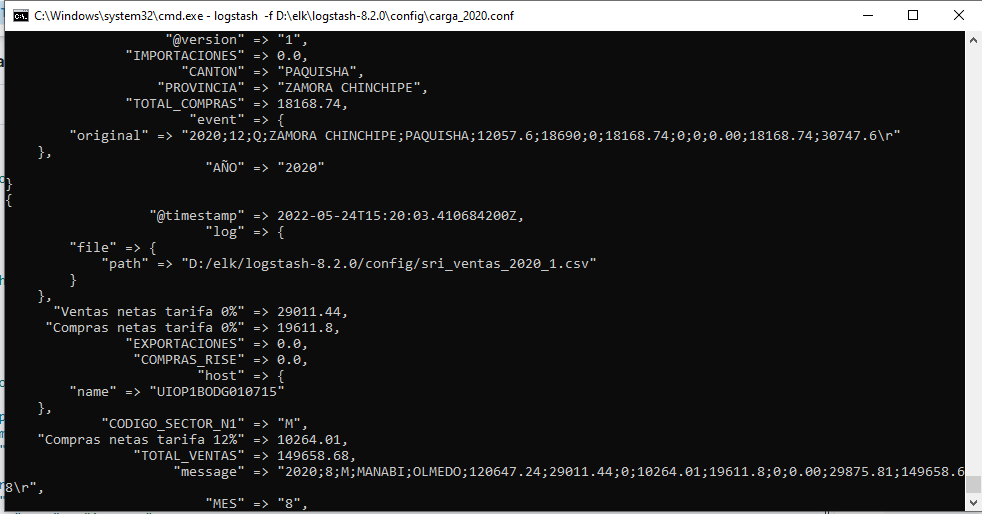
}

La ejecución de la carga se realiza a través de la siguiente sentencia sobre una línea de comandos:

<RUTA\_LOGSTASH>\bin>logstash -f D:\elk\logstash-8.2.0\config\carga\_202X.conf

La Figura 10 despliega la consola que genera el proceso de carga con LogStash, donde se visualiza los datos de los ficheros .csv para su ingesta en el índice declaraciones:

Figura 10 Consola de LogStash para carga de datos



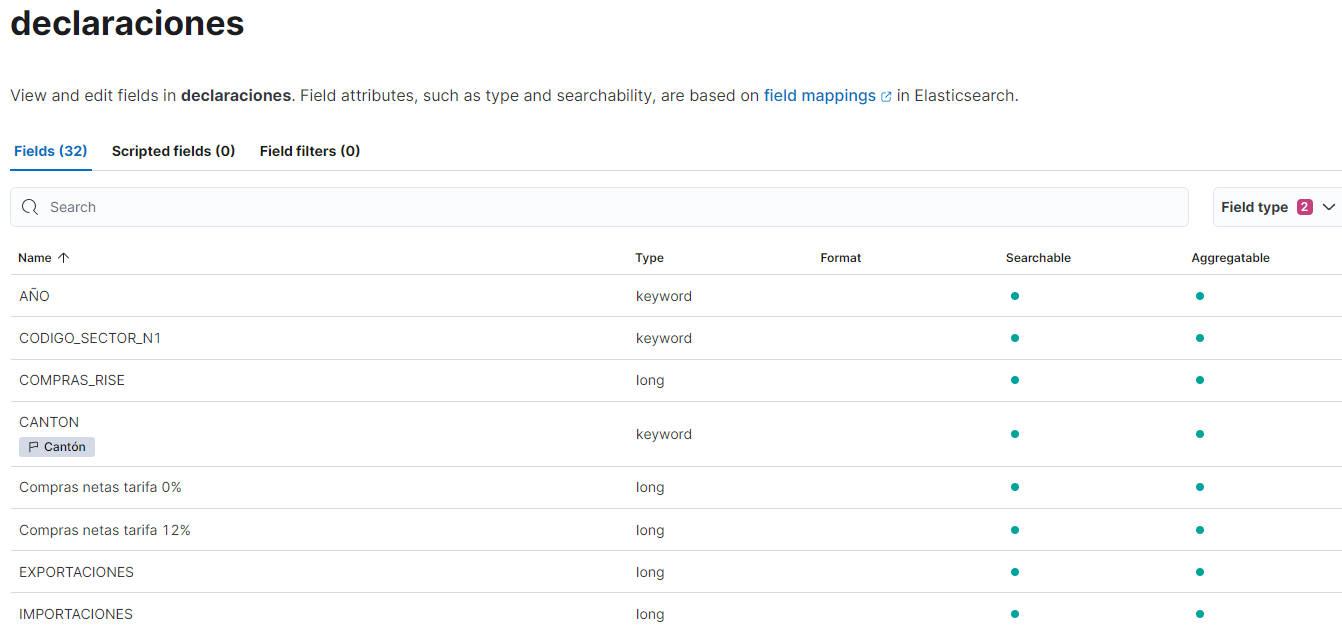
* **VISUALIZACIÓN DE DATOS CON KIBANA**

Con el objetivo de poder verificar que los datos se cargaron se bosqueja un sencillo *dashboard* el cual representa la información cargada. Para ello se ejecutaron las siguientes acciones:

**Creación de Vista (*View*):** A través de Kibana se generan vistas, las cuales son una representación lógica de los índices definiendo una capa semántica para la visualización de los campos constituyentes del índice bajo un lenguaje comprensible para el usuario, así como características para que los campos puedan ser agregables.

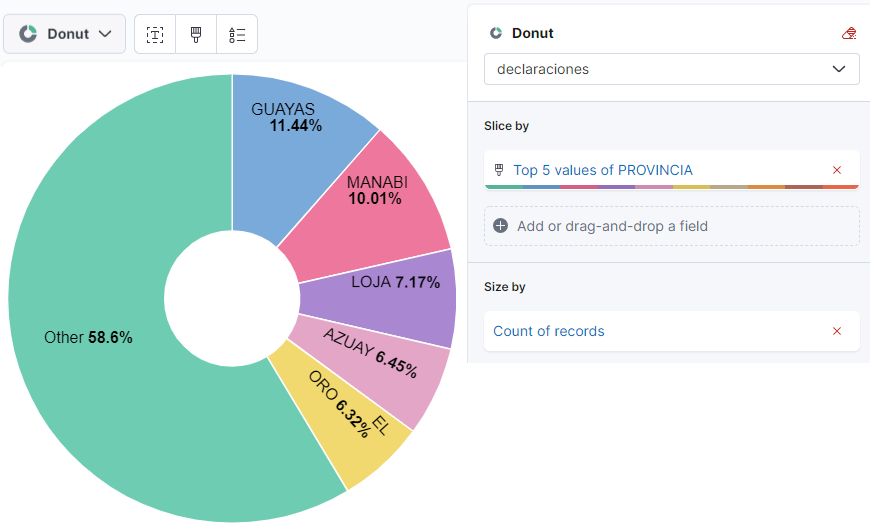
La Figura 11 representa la creación de la vista declaraciones, donde se enlista los campos que conforman el índice. Se puede verificar que el campo CANTON tiene una etiqueta adicional llamada “Cantón” para su visualización dentro del *dashboard*.

Figura 11 Creación de una Vista en Kibana



**Creación de Visualizaciones y *Dashboard*:**  Una visualización corresponde a un tipo de gráfica o tabla; un *dashboard*  constituye un conjunto de visualizaciones. La Figura 12 permite visualizar la creación de una gráfica tipo pastel (*Donut*) donde se obtiene el total de registros por provincia.

Figura 12 Visualización de Número de Registros por Provincia

****

Creado las visualizaciones, se construye el *dashboard*  tal cual se observa en la Figura 13, que contiene tres gráficas: una tabla con el total de registros por año, un gráfico de barras para obtener la sumatoria del Total de Compras y Ventas por año y un Pastel.

Figura 13 Dashboard con datos de declaraciones históricos

Interfaz de usuario gráfica

Descripción generada automáticamente

# 4.2. Generación de datos de detalle

### 4.2.1 Modelo de Datos

La Tabla 5, define el diccionario de datos correspondiente a la información a detalle de declaraciones, la misma guarda completa relación con la información agregada, a excepción del campo CODIGO, que corresponde a un campo auto incremental, útil para la posterior captura de los registros mediante Kafka. Adicionalmente, se incorpora el nombre de la variable a nivel de campos de la tabla de base datos: DECLARACIONES.

Tabla 5 Diccionario de datos de declaraciones al Detalle

|  |  |  |
| --- | --- | --- |
| **Nombre de la variable** | **Nombre en la base de datos** | **Tipo** |
| Código | CODIGO | Numérico |
| Año | ANIO | Numérico |
| Mes | MES | Numérico |
| Codigo\_Sector\_N1 | CODIGO\_SECTOR\_N1 | Alfanumérico |
| Provincia | PROVINCIA | Alfanumérico |
| Cantón | CANTON | Alfanumérico |
| Ventas netas tarifa 12% | VENTAS\_NETAS\_12 | Numérico |
| Ventas netas tarifa 0% | VENTAS\_NETAS\_0 | Numérico |
| Exportaciones | EXPORTACIONES | Numérico |
| Compras netas tarifa 12% | COMPRAS\_NETAS\_12 | Numérico |
| Compras netas tarifa 0% | COMPRAS\_NETAS\_0 | Numérico |
| Importaciones | IMPORTACIONES | Numérico |
| Compras\_Rise | COMPRAS\_RISE | Numérico |
| Total\_Compras | TOTAL\_COMPRAS | Numérico |
| Total\_Ventas | TOTAL\_VENTAS | Numérico |

**Fuente:** Adaptado a partir de la tabla original publicada por el SRI

Para la generación de valores de detalle en lo que respecta a los campos de Provincia y Cantón es necesario contar con el catastro de ubicaciones geográficas del Ecuador, para ello se requiere de una estructura denominada GEOGRAFICA que almacenará la información descargada del portal del Instituto Nacional de Estadísticas y Censos (INEC). La Tabla 6 muestra el diccionario de datos de la estructura GEOGRAFICA:

Tabla 6 Diccionario de datos de declaraciones al Detalle

|  |  |  |
| --- | --- | --- |
| **CAMPO** | **DESCRIPCIÓN** | **TIPO** |
| PROVINCIA | Nombre de Provincia | Alfanumérico |
| CANTON | Nombre de Cantón | Alfanumérico |

### 4.2.2 Definición de estructuras en Oracle

Para la creación de la tabla que almacenará la información a detalle y el procedimiento encargado de generarla de forma ficticia y masiva, se requiere de un esquema/usuario de base datos. Toda la información sobre la creación del esquema, así como los privilegios requeridos para la creación de objetos de base de datos se detallan en el Anexo I de esta memoria.

Una vez configurado el esquema DECLARACIONES y permisos respectivos; se describe la creación de la tabla DECLARACIONES, junto a sus campos constituyentes:

create table declaraciones(

CODIGO number(15), ANIO number(4),

MES number(2), CODIGO\_SECTOR\_N1 varchar2(10),

PROVINCIA varchar2(100),CANTON varchar2(100),

VENTAS\_NETAS\_12 number(24,2), VENTAS\_NETAS\_0 number(12,2),

EXPORTACIONES number(24,2), COMPRAS\_NETAS\_12 number(12,2),

COMPRAS\_NETAS\_0 number(24,2), IMPORTACIONES number(12,2),

COMPRAS\_RISE number(24,2), TOTAL\_COMPRAS number(12,2),

TOTAL\_VENTAS number(12,2));

ALTER TABLE declaraciones ADD (CONSTRAINT declaraciones\_PK PRIMARY KEY (CODIGO));

El campo CODIGO, corresponde a un valor auto incremental, por consiguiente, se crea objetos tipo secuencia y *trigger* (dispararador) que generan automáticamente un valor de CODIGO cada que se requiere insertar un registro sobre la tabla.

CREATE SEQUENCE declaraciones\_sequence;

CREATE OR REPLACE TRIGGER declaraciones\_trigger

BEFORE INSERT ON declaraciones FOR EACH ROW

BEGIN

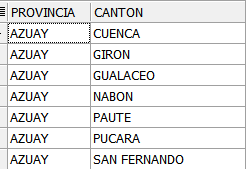
SELECT declaraciones\_sequence.nextval INTO :new.CODIGO FROM dual;

END;

La tabla GEOGRAFICA, encarga de almacenar los datos de provincias y cantones tiene la siguiente definición:

create table geografica (provincia varchar2(100), canton varchar2(150));

La Figura 14 detalla una muestra con la información de provincias y cantones, para un total de 221 registros.

Figura 14 Muestra de datos ubicaciones geográficas

Con la creación de la estructura donde se almacena la información a detalle de declaraciones, junto a la tabla catálogo con ubicaciones geográficas, es momento de un procedimiento almacenado GENERACION\_DECLARACIONES, responsable de poblar la tabla DECLARACIONES con data randómica al momento de ser invocado. Este procedimiento tiene un parámetro denominado nombre\_provincia el cual recibe el valor de una determinada provincia para la consecuente generación de datos de tal. Este parámetro admite valores nulos, en caso de ser así, el procedimiento generará información de cualquier provincia.

CREATE OR REPLACE PROCEDURE generacion\_declaraciones (

nombre\_provincia IN varchar2 DEFAULT NULL )

IS

var\_ANIO NUMBER := 2022;

var\_MES NUMBER;

var\_CODIGO\_SECTOR\_N1 VARCHAR2 (10);

var\_PROVINCIA VARCHAR2 (100);

var\_CANTON VARCHAR2 (100);

var\_VENTAS\_NETAS\_12 NUMBER (24,2);

var\_VENTAS\_NETAS\_0 NUMBER (24,2);

var\_EXPORTACIONES NUMBER (24,2);

var\_COMPRAS\_NETAS\_12 NUMBER (24,2);

var\_COMPRAS\_NETAS\_0 NUMBER (24,2);

var\_IMPORTACIONES NUMBER (24,2);

var\_COMPRAS\_RISE NUMBER (24,2);

var\_TOTAL\_COMPRAS NUMBER (24,2);

var\_TOTAL\_VENTAS NUMBER (24,2);

BEGIN

-- GENERACION DE UN VALOR RANDÓMICO PARA EL MES

var\_mes := ROUND (DBMS\_RANDOM.VALUE (1, 12));

/\*OBTENCION DE UN VALOR RANDÓMICO DE PROVINCIA REALIZANDO

PREVIAMENTE LA VALIDACION SI SE INVOCÓ AL PROCEDIMIENTO

CON VALOR DE PROVINCIA EN ESTADO NULL, SE OBTIENE PROVINCIA Y CANTÓN

DE FORMA RANDÓMICA\*/

if nombre\_provincia is null then

SELECT provincia, canton into var\_provincia, var\_canton

FROM (SELECT \* FROM geografica ORDER BY DBMS\_RANDOM.RANDOM)

WHERE rownum =1;

else --SI LA PROVINCIA NO ES NULO SE OBTIENE UN CANTÓN DE FORMA RANDÓMICA

SELECT canton into var\_canton

FROM ( SELECT \* FROM geografica where provincia=nombre\_provincia

ORDER BY DBMS\_RANDOM.RANDOM) WHERE rownum =1;

var\_provincia:=nombre\_provincia;

end if;

-- GENERACION DEL CÓDIGO DE SECTOR CON VALORES ENTRE A y X

var\_CODIGO\_SECTOR\_N1:=chr(DBMS\_RANDOM.VALUE (65, 88));

-- GENERACION DE VALORES RANDOMICOS PARA LOS CAMPOS TIPO NUMÉRICO

var\_VENTAS\_NETAS\_12 := ROUND (DBMS\_RANDOM.VALUE (1, 100000), 2);

var\_COMPRAS\_NETAS\_12 := ROUND (DBMS\_RANDOM.VALUE (1, 100000), 2);

var\_VENTAS\_NETAS\_0:=var\_VENTAS\_NETAS\_12\*1.1;

var\_EXPORTACIONES:=var\_VENTAS\_NETAS\_12\*1.2;

var\_COMPRAS\_NETAS\_0:=var\_COMPRAS\_NETAS\_12\*1.1;

var\_IMPORTACIONES:=var\_COMPRAS\_NETAS\_12\*1.5;

var\_COMPRAS\_RISE:=var\_COMPRAS\_NETAS\_12\*0.05;

--OBTENCIÓN DE LOS TOTALES DE COMPRAS Y VENTAS

var\_TOTAL\_COMPRAS :=var\_COMPRAS\_NETAS\_12 + var\_COMPRAS\_NETAS\_0 + var\_COMPRAS\_RISE;

var\_TOTAL\_VENTAS := var\_VENTAS\_NETAS\_12 + var\_VENTAS\_NETAS\_0;

FOR i IN 1 .. 500

LOOP

--INSERCIÓN EN LA TABLA DE DECLARACIONES

INSERT /\*+ append \*/ INTO declaraciones nologging (anio,

mes,provincia, canton, CODIGO\_SECTOR\_N1,VENTAS\_NETAS\_12,

VENTAS\_NETAS\_0, EXPORTACIONES,COMPRAS\_NETAS\_12,COMPRAS\_NETAS\_0,

IMPORTACIONES,COMPRAS\_RISE,TOTAL\_COMPRAS,TOTAL\_VENTAS)

VALUES (var\_anio,

var\_mes,var\_provincia, var\_canton, var\_CODIGO\_SECTOR\_N1,

var\_VENTAS\_NETAS\_12,var\_VENTAS\_NETAS\_0,var\_EXPORTACIONES,

var\_COMPRAS\_NETAS\_12,var\_COMPRAS\_NETAS\_0,var\_IMPORTACIONES,

var\_COMPRAS\_RISE,var\_TOTAL\_COMPRAS,var\_TOTAL\_VENTAS);

END LOOP;

COMMIT;

END;

### 4.2.3 Interfaz gráfica

Con el propósito de controlar la provincia que generen mayor cantidad de datos a fin de poder visualizar como los reportes en tiempo real cambian durante la ingesta de datos, se propone de una sencilla interfaz gráfica, en la Figura 15 se visualiza la estructura de la interfaz y sus respectivos componentes:

Figura 15 Interfaz gráfica declaraciones



Esta interfaz, creada con el entorno de desarrollo Netbeans para lenguaje Java, consta de los siguientes componentes:

* Radio botones con las 24 provincias del Ecuador, que invocan al procedimiento GENERACION\_DECLARACIONES enviando como parámetro el valor de una provincia; de esta forma la tabla de declaraciones será cargada con información exclusiva de la provincia seleccionada.
* Botón BALANCEAR: Al hacer clic sobre este componente gráfico, se invoca al mismo procedimiento de generación de declaraciones, con la única diferencia que los datos generados corresponderán a cualquier provincia.

La Tabla 7, realiza una descripción acerca de las clases constituyentes de la interfaz gráfica:

Tabla 7 Clases Java de interfaz gráfica

|  |  |
| --- | --- |
| **CLASE (.JAVA)** | **DESCRIPCIÓN** |
| Generación\_  Declaraciones | Interfaz gráfica con los componentes para seleccionar una provincia y llamar al procedimiento almacenado GENERACION\_DECLARACIONES |
| hilo\_provincia | Clase que crea hilos de ejecución, para que la interfaz gráfica se encuentre constantemente verificando el valor que toman los componentes gráficos. |

La clase Generacion\_Declaraciones, contiene distintos métodos para construir los componentes gráficos y las acciones a ejecutar cuando estos son seleccionados; así como la instanciación de 25 objetos de la clase hilo\_provincia para la permantente ingesta de datos. En la Tabla 8 se visualiza fragmentos importantes de código de esta clase y una descripción sobre su funcionalidad:

Tabla 8 Fragmentos de código clase Generacion\_Declaraciones

|  |  |
| --- | --- |
| **FRAGMENTO CODIGO** | **DESCRIPCIÓN** |
| hilo\_provincia hilo[]=new hilo\_provincia[25]; | Declaración de un vector de 25 hilos correspondientes a las 24 provincias del país y uno adicional para la generación de datos de todas las provincias |
| numero\_provincia=0;  if (jRadioAzuay.isSelected()){  //Activo el hilo de esta provincia  hilo[numero\_provincia].activa\_hilo();  //desactivo el hilo de generación de cualquier provincia  hilo[24].desactiva\_hilo();  }  else{ //desactivo el hilo de esta provincia hilo[numero\_provincia].desactiva\_hilo(); } | Activación y desactivación del hilo de ejecución relacionado a esta provincia en caso de (de)seleccionar este componente. En esta memoria se incluye el ejemplo de la provincia del Azuay, sin embargo, este código se repite tal cual para el resto de componentes radio botón. La diferencia es el cambio de la variable numero\_provincia cuyo valor es diferente para cada provincia. |
| numero\_provincia=24;  hilo[numero\_provincia].activa\_hilo();  for( int i=0; i<24; i++){  hilo[i].thread.suspend();  }  jbuttonGroup.clearSelection(); | En caso de dar clic sobre el BOTON\_BALANCEAR, se desactiva todos los hilos relacionados a los radio botones (provincias), para generar datos de todas las provincias |

La clase hilo\_provincia, contiene la definición de los hilos concurrentes de ejecución, con métodos parar iniciar, suspender o reiniciar la ejecución de un hilo y conexiones a la base de datos Oracle al procedimiento de generación de declaraciones. En la Tabla 9 se visualiza fragmentos importantes de código de esta clase y una descripción sobre su funcionalidad:

Tabla 9 Fragmentos de código clase hilo\_provincia

|  |  |
| --- | --- |
| **FRAGMENTO CODIGO** | **DESCRIPCIÓN** |
| public void activa\_hilo() {  decodifica\_provincia(numero\_hilo);  if (contador\_select == 0) {  this.thread.start();  } else {  this.thread.resume();  }  contador\_select = 1;  }  public void desactiva\_hilo() {  this.thread.suspend();  } | Métodos que permiten activar o desactivar el hilo de ejecución. Estos son llamados cuando se ha ejecutado alguna acción sobre los componentes gráficos: Al (de)seleccionar un radio botón, o dar clic sobre el botón BALANCEAR |
| Class.forName("oracle.jdbc.OracleDriver");  conn1 = DriverManager.getConnection(dbURL1);  sentencia = "CALL **generacion\_declaraciones**";  CallableStatement pstmt = conn1.prepareCall("{CALL generacion\_declaraciones(?)}");  pstmt.setString(1, **provincia**);  pstmt.executeUpdate();  conn1.close(); | Conexión a la base de datos Oracle, donde se ejecuta al procedimiento de generación de claves, teniendo como atributo el nombre de la provincia. Cuando se presiona el botón BALANCEAR, el valor enviado de provincia es nulo, por lo que el procedimiento generará data de cualquier provincia. |

### 4.2.3 Resultados de generación de datos de declaraciones a detalle

La Tabla 10, refleja los resultados agregados por provincia en distintos momentos de ejecución; a través de la sentencia:

select provincia, count(\*) TOTAL\_REGISTROS from declaraciones group by provincial

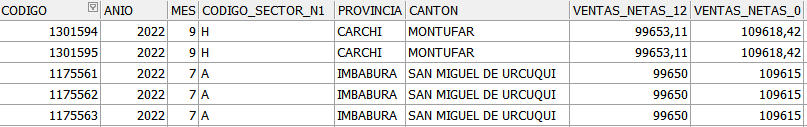
Cabe aclarar que los totales de los registros podrían variar, dependiendo su velocidad de ingesta en parámetros como consumo de RAM, procesamiento, tráfico de red o Disco Duro.

Tabla 10 Componentes gráficos interfaz declaraciones

|  |  |  |
| --- | --- | --- |
| **TIEMPO (s)** | **COMPONENTE SELECCIONADO** | **RESULTADO** |
| t=0 | Ningún Componente seleccionado | TABLA DECLARACIONES VACIA |
| t=10 |  |  |
| t=40 |  |  |
| t=105 |  |  |
| t=180 | Clic sobre | Se captura una muestra de resultados agregados generados. |

En la Figura 16 se visualiza una muestra de la información generada correspondiente de declaraciones a detalle en la base de datos transaccional.

Figura 16 Datos generados de declaraciones



# 4.3. Captura de información a detalle

### 4.3.1 Apache Kafka

Kafka es una plataforma desarrollada inicialmente por Linkedln para el manejo de eventos en tiempo real, basada en el paradigma publicador-suscriptor donde un conjunto de clientes escribe(publicadores) y leen (suscriptores) hacia y desde un bus de datos de naturaleza perdurable y tolerante a fallos.

**Arquitectura**

La Figura 17 despliega la arquitectura básica de Kafka y sus distintos componentes en un proceso de captura de datos en tiempo real.

Figura 17 Arquitectura Kafka

Diagrama

Descripción generada automáticamente

**connect**

**connect**

**connect**

**Nota:** Se modificó la figura original de https://www.educba.com/ incorporando los *connects*

Dentro de esta se describen los siguientes componentes:

* **Productor:** Responsable del envío de mensajes (tópicos) hacia un clúster de Kafka conformado por uno o varios *brokers* (servidores) que garantizan la tolerancia a fallos a través de la replicación.
* **Consumidor:**  Quien consume los tópicos almacenados en el servidor de Kafka, siendo este un mediador entre el consumidor y el productor.
* **Broker:**  Un servidor dentro de un clúster de Kafka.
* **Tópico (*Topic*):**  Es un tipo de mensaje en *stream,* único, que hace referencia a una estructura de datos generada por el productor y accesible por uno o varios consumidores.
* **Partición:**  En ambientes distribuidos como lo es Kafka, con varios servidores dentro de un clúster y ante la ingesta de grandes volúmenes de datos, una partición consiste en la segmentación de un tópico en distintos “pedazos” que son distribuidos entre los miembros del clúster.
* ***Offset*:**Consiste en un secuencial que se asigna sobre cada partición de un tópico, este número va en aumento a medida que existe nuevos mensajes entrantes al servidor.
* **Zookeeper:** Dentro de Kafka, Zookeeper cumple las funciones de coordinador, encargado de coordinar el trabajo de los distintos servidores del clúster, así como rastrear los tópicos y sus particiones.

Dentro de Kafka existe un elemento llamado conector (*Connect*), que actúa como mediador entre una fuente de datos (o destino) y el servidor de Kafka, de esta forma se puede realizar la transferencia de grandes volúmenes de datos en tiempo real hacia consumidores como *data lakes* o motores de procesamiento distribuido.

### 4.3.2 Configuración Apache Kafka- Confluent

Confluent es una plataforma desarrollada por los creadores de Kafka, que incorpora sobre la distribución original de Kafka mejoras adicionales para facilitar la conectividad entre esta y distintas fuentes de datos. Para el presente TFM se utiliza Confluent gracias a que implementa conectores sobre base de datos a través de controladores (jdbc), de esta forma resulta sencillo acceder a una BBDD con solo incluir parámetros de conexión, esquema, credenciales, e/o.

Para efectos de la configuración de Kafka, se tiene las siguientes variables, cuyos valores se detallan a continuación en la Tabla 11:

Tabla 11 Variables de configuración Kafka

|  |  |
| --- | --- |
| **VARIABLE** | **VALOR** |
| <PATH\_KAFKA> | /home/byron/confluent-4.1.1 |
| <KAKFA\_IP\_ADDRESS> | 192.168.1.43 |
| <ZOOKEEPER\_IP\_ADDRESS> | 192.168.1.43 |
| <ORACLE\_IP\_ADDRESS> | 192.168.1.56 |
| <PUERTO\_ORACLE> | 1521 |
| <SID> | XE |
| <USUARIO> | DECLARACIONES |
| <CONTRASEÑA> | DECLARACIONES |
| <TABLA\_BDD\_ORACLE> | DECLARACIONES |
| <CAMPO\_INCREMENTAL\_TABLA\_ORACLE> | CODIGO |

**Configuración:**

* Descargar la plataforma comprimida que contiene el servidor de kafka de la página oficial de Confluent, versión 4.1.1; y descomprimirlo en una ruta determinada, en adelante <PATH\_KAFKA>.
* Para la conexión del *content* con Oracle se deberá descargar el controlador ojdbc del sitio web de Oracle; para la versión de base Oracle 18c se utiliza el ojdbc versión 8, este será copiado en la ruta <PATH\_KAFKA>/share/java/kafka-connect-jdbc
* Abrir el archivo **zookeeper.properties**, ubicado en la ruta <PATH\_KAFKA>/etc/kafka y configurar el parámetro dataDir correspondiente al sitio donde se aloja los logs que el servidor de zookeeper genera, así como los registros de base de datos en memoria:

dataDir=<PATH\_KAFKA>/tmp/zookeeper

* Modificar el archivo **server.properties** de la ruta <PATH\_KAFKA>/etc/kafka los siguientes parámetros, necesarios para conectarse hacia el servidor de zookeeper y de los *listeners* que permitirán la creación de los servicios de *sockets* en las comunicaciones con los productores y consumidores:

broker.id=0

listeners=PLAINTEXT://<KAKFA\_IP\_ADDRESS>:9092

advertised.listeners=PLAINTEXT://<KAKFA\_IP\_ADDRESS>:9092

log.dirs=<PATH\_KAFKA>/tmp/kafka-logs

zookeeper.connect=<ZOOKEEPER\_IP\_ADDRESS>:2181

* Modificar el archivo **connect-standalone.properties**  dentro de <PATH\_KAFKA>/etc/kafka, parámetros para la conexión con el servidor de kafka, conversión de los datos almacenados en la tabla de la base de datos Oracle a formato json. Al parámetro plugin.path se deberá añadir a las rutas existentes [RUTAS\_CONFIGURADAS\_DEFAULT] el directorio que contiene el ojdbc de conexión a la base Oracle.

bootstrap.servers=<KAKFA\_IP\_ADDRESS>:9092

internal.key.converter=org.apache.kafka.connect.json.JsonConverter

internal.value.converter=org.apache.kafka.connect.json.JsonConverter

key.converter=org.apache.kafka.connect.json.JsonConverter

value.converter=org.apache.kafka.connect.json.JsonConverter

key.converter.schemas.enable=false

value.converter.schemas.enable=false

internal.key.converter.schemas.enable=false

internal.value.converter.schemas.enable=false

offset.storage.file.filename=<PATH\_KAFKA>/tmp/connect.offsets

plugin.path=[RUTAS\_CONFIGURADAS\_DEFAULT], <PATH\_KAFKA>/share/java/kafka- connect-jdbc

* Crear el fichero **fuente\_declaraciones.properties** en la ruta <PATH\_KAFKA>/etc/kafka, con parámetros para la conexión a la base de datos y objeto tabla con la información a capturar.

name=source-oracle-jdbc  
connector.class=io.confluent.connect.jdbc.JdbcSourceConnector  
tasks.max=1  
connection.url=jdbc:oracle:thin:<USUARIO>/<CONTRASEÑA>@<ORACLE\_IP\_ADDRESS>:<PUERTO\_ORACLE>/<SID>  
mode=incrementing  
incrementing.column.name=<CAMPO\_INCREMENTAL\_TABLA\_ORACLE>  
topic.prefix=topic\_  
table.whitelist=<TABLA\_BDD\_ORACLE>  
numeric.mapping=best\_fit

**Inicio de Servicios:** Ejecutar las siguientes sentencias en tres distintas terminales:

* Inicio del servidor de zookeper

sudo <PATH\_KAFKA>/bin/zookeeper-server-start <PATH\_KAFKA>/etc/kafka/zookeeper.properties

* Arranque del broker Kafka

sudo <PATH\_KAFKA>/bin/kafka-server-start<PATH\_KAFKA>/etc/kafka/server.properties

* Inicio del servicio Kafka-connect para la generación de un job para conexión a la base Oracle:

sudo <PATH\_KAFKA>bin/connect-standalone <PATH\_KAFKA>etc/kafka/connect standalone.properties etc/kafka/fuente\_declaraciones.properties

**Validación de Configuración**

Para validar que la configuración esté correcta, se puede crear un consumidor que vaya desplegando los resultados que Kafka captura a partir de su conexión a la base de datos, mediante la siguiente sentencia en una terminal:

sudo <PATH\_KAFKA>/bin/kafka-console-consumer --bootstrap-server <KAKFA\_IP\_ADDRESS>:9092 --topic topic\_<TABLA\_BDD\_ORACLE> --from-beginning

La Figura 18 muestra cómo al hacer una inserción de datos en la base de datos Oracle, se generan mensajes que son enviados a Kafka.

Figura 18 Muestra de datos capturados por Kafka

Una captura de pantalla de una computadora

Descripción generada automáticamente

# 4.4. Procesamiento y persistencia de información de declaraciones

En esta sección se hace una cobertura sobre la integración de Kafka con Apache Spark, siendo este último un motor de cálculo para el procesamiento de grandes volúmenes de datos de una forma distribuida y paralela; finalmente los datos que Spark capture y procese serán persistidos en ElasticSearch mediante la lectura y escritura de la tabla (índice) **declaraciones\_2022** que este almacena. Este tipo de índice tiene la misma estructura del índice **declaraciones,** utilizado para almacenar la *data* de declaraciones del periodo 2020-2021.

### 4.4.1 Apache Spark

Spark se define como una plataforma de código abierto para el procesamiento paralelo de datos en memoria, cuyo origen se remonta a la Universidad de California Berkeley, liberado tiempo después al *Apache Software Foundation*. Spark se encuentra desarrollado en Java y Scala, proveyendo APIs para la implementación de aplicaciones también en R y Python. Para este TFM se empleó Python 3.7 a través de la API denominada Pyspark sobre Escala 11 y JDK 8.

**Arquitectura:**

La Figura 19 grafica la arquitectura de Apache Spark, con tres elementos: el Controlador (*Driver),* que corresponde a la interfaz de usuario donde se crea una sesión de conexión y el programa el cual será ejecutado por los *Executors (nodos); y el Cluster Manager,* responsable de la administración de recursos para el balanceo y la distribución de los programas.

Figura 19 Arquitectura Apache Spark



**Fuente**: https://docs.microsoft.com

De acuerdo a la arquitectura detalla al inicio del capítulo, se cuenta con un solo equipo para la implementación del aplicativo en Apache Spark, siendo este sobre el que se implementará el desarrollo del programa sobre Spyder-Python (*Driver*), gestionará los recursos de memoria (*Cluster Manager*) y ejecutará dicho código (*Executor*).

**Componentes**

* **Spark Core:**  Contiene la estructura de datos fundamental de Spark: RDD *(Resilient Distributed Dataset),* que es una colección de datos en memoria, tolerante a fallos, particionada y distribuida.
* **Spark SQL y API Estructurada:**  Componente para el manejo de datos de forma estructurada a través de *Dataframes*, siendo este una interfaz sobre los RDD que permiten su manipulación de una forma más sencilla. Estos *Dataframes* pueden ser procesados a través de simples consultas SQL.
* **Spark Streaming:** Componente para el procesamiento de datos en formato RDD cuyo flujo es en tiempo real.
* **Spark Graphs:** Para el procesamiento de grafos.

Dentro de la implementación de la integración de Spark con Kafka y Elastic, documentado en este TFM, se hizo uso explícito de los componentes Spark SQL y API Estructura, así como Spark Structrured Streaming, que es una variante de Spark Streaming para el procesamiento de datos mediante *Dataframes*.

### 4.4.2 Integración Kafka - Spark - ElasticSearch

El desarrollo del código que integra estos tres componentes se describe a través de las siguientes actividades:

**Importación de Librerías y Creación de Sesión:** Importación de librerías del API Pyspark, con funciones para la manipulación de *Dataframes*. Para la creación de la sesión de Spark que incluye parámetros como el nombre de la sesión y librerías útiles para la integración con Elastic y Kafka.

Es muy importante recordar que, para conseguir una total integración entre los distintos componentes, se deberá considerar las versiones de cada uno de ellos y los lenguajes que la soportan: JDK8, Python 3.7, Scala 2.11; por ejemplo, en el siguiente código al incluir las librerías de integración, se resalta las versiones de los componentes, donde la librería para ElasticSearch corresponde para la versión de Elastic **8.2.0** y aquella para la conexión con Kafka es de la versión de Spark **2.4.8** sobre escala **2.11.**

import pyspark

from pyspark.sql import SparkSession

from pyspark.sql.types import StructType, StructField, StringType, DoubleType

from pyspark.sql import functions as F

spark = SparkSession.builder.appName("DECLARACIONES\_2022") \

.config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10\_**2.11:2.4.8**,org.elasticsearch:elasticsearch-hadoop:**8.2.0**") \

.config('spark.sql.debug.maxToStringFields', 2000) \

.config('spark.debug.maxToStringFields', 2000) \

.getOrCreate()

**Conexión a Kafka:** Se crea un *dataframe* para la lectura en *streaming* de la información proveniente de Kafka, para ello se define parámetros como la dirección IP del *broker* de Kafka y el tópico a consumir.

Declaraciones\_StreamingDF =spark.readStream\

.format("kafka")\

.option("kafka.bootstrap.servers", "192.168.1.43:9092")\

.option("subscribe", "topic\_DECLARACIONES")\

.load()

**Transformación de los *dataframes* entrandes**: El *dataframe* que lee los datos desde Kafka corresponde a un conjunto de bytes de información; dentro de estos se encuentra el campo “value”, que guarda en una cadena de texto con formato json los datos generados por el productor de Kafka, para su extracción se define un objeto tipo *StructType*  con la estructura de campos relativos al tópico y su mapeo dentro de un nuevo *dataframe*.

esquema=StructType([\

StructField("ANIO", StringType()),\

StructField("MES", StringType()),\

StructField("PROVINCIA", StringType()),\

StructField("CANTON", StringType()),\

StructField("CODIGO\_SECTOR\_N1", StringType()),\

StructField("VENTAS\_NETAS\_12", DoubleType()) ,\

StructField("VENTAS\_NETAS\_0", DoubleType()) ,\

StructField("EXPORTACIONES", DoubleType()) ,\

StructField("COMPRAS\_NETAS\_12", DoubleType()) ,\

StructField("COMPRAS\_NETAS\_0", DoubleType()) ,\

StructField("IMPORTACIONES", DoubleType()) ,\

StructField("COMPRAS\_RISE", DoubleType()) ,\

StructField("TOTAL\_COMPRAS", DoubleType()) ,\

StructField("TOTAL\_VENTAS", DoubleType())])

parsedDF = Declaraciones\_StreamingDF\

.select("value")\

.withColumn("value", F.col("value").cast(StringType()))\

.withColumn("parejas", F.from\_json(F.col("value"), esquema))\

.withColumn("ANIO", F.col("parejas.ANIO"))\

.withColumn("MES", F.col("parejas.MES"))\

.withColumn("PROVINCIA", F.col("parejas.PROVINCIA"))\

.withColumn("CANTON", F.col("parejas.CANTON"))\

.withColumn("CODIGO\_SECTOR\_N1", F.col("parejas.CODIGO\_SECTOR\_N1"))\

.withColumn("VENTAS\_NETAS\_12", F.col("parejas.VENTAS\_NETAS\_12"))\

.withColumn("VENTAS\_NETAS\_0", F.col("parejas.VENTAS\_NETAS\_0"))\

.withColumn("EXPORTACIONES", F.col("parejas.EXPORTACIONES"))\

.withColumn("COMPRAS\_NETAS\_12", F.col("parejas.COMPRAS\_NETAS\_12"))\

.withColumn("COMPRAS\_NETAS\_0", F.col("parejas.COMPRAS\_NETAS\_0"))\

.withColumn("IMPORTACIONES", F.col("parejas.IMPORTACIONES"))\

.withColumn("COMPRAS\_RISE", F.col("parejas.COMPRAS\_RISE"))\

.withColumn("TOTAL\_COMPRAS", F.col("parejas.TOTAL\_COMPRAS"))\

.withColumn("TOTAL\_VENTAS", F.col("parejas.TOTAL\_VENTAS"))

**Interacción con ElasticSearch:** Con el *dataframe*  resultante*,*  con una estructura definida de filas y columnas (campos), se aplica algunas operaciones de lectura y escritura sobre la base de datos de ElasticSearch, a fin de que los datos provenientes de Kafka sean consolidados con la información ya existente; para ello se define una función llamada **CONEXION\_ELASTIC** que se invoca a través de procesos en *microbatch* mediante la sentencia foreachbatch. Esta sentencia resulta de suma utilidad ya que permite aplicar cierta lógica sobre datos entrantes en modo *streaming;* para este proyecto se genera algunas transformaciones y operaciones dml sobre la base de datos sobre los datos entrantes en un periodo de 30 segundos.

escritura=parsedDF.writeStream.foreachBatch(CONEXION\_ELASTIC).trigger(processingTime='30 seconds').start()

La función **CONEXIÓN\_ELASTIC** ejecuta las siguientes instrucciones:

* **Agrupamiento del conjunto de datos ingestados** en los últimos 30 segundos por aquellas variables categóricas de los datos de declaraciones y la suma de aquellas variables cuantitativas.

datos\_entrantes=df.select("ANIO","MES","PROVINCIA","CANTON",\

"CODIGO\_SECTOR\_N1",'VENTAS\_NETAS\_12','VENTAS\_NETAS\_0',\ 'EXPORTACIONES','COMPRAS\_NETAS\_12','COMPRAS\_NETAS\_0',\ 'IMPORTACIONES','COMPRAS\_RISE','TOTAL\_COMPRAS','TOTAL\_VENTAS')\

.groupBy("ANIO","MES","PROVINCIA","CANTON","CODIGO\_SECTOR\_N1")\

.agg(F.sum("VENTAS\_NETAS\_12").alias("VENTAS\_NETAS\_12"),\

F.sum("VENTAS\_NETAS\_0").alias("VENTAS\_NETAS\_0"),\

F.sum("EXPORTACIONES").alias("EXPORTACIONES"),\

F.sum("COMPRAS\_NETAS\_12").alias("COMPRAS\_NETAS\_12"),\

F.sum("COMPRAS\_NETAS\_0").alias("COMPRAS\_NETAS\_0"),\

F.sum("IMPORTACIONES").alias("IMPORTACIONES"),\

F.sum("COMPRAS\_RISE").alias("COMPRAS\_RISE"),\

F.sum("TOTAL\_COMPRAS").alias("TOTAL\_COMPRAS"),\

F.sum("TOTAL\_VENTAS").alias("TOTAL\_VENTAS"))

* **Lectura de los datos almacenados en el índice declaraciones\_2022** de ElasticSearch , y su posterior transformación para la consolidación con la data entrante de Kafka.

es\_lectura = {"es.nodes" : "192.168.1.52","es.port" : "9200","es.resource" : "declaraciones\_2022","es.read.metadata": "true" }

declaraciones= spark.read.format("org.elasticsearch.spark.sql")\.

options(\*\*es\_lectura).load()

declaraciones=declaraciones.withColumn("ANIO1",F.col("ANIO"))\

.withColumn("MES1",F.col("MES"))\

.withColumn("PROVINCIA1",F.col("PROVINCIA"))\

.withColumn("CANTON1",F.col("CANTON"))\

.withColumn("CODIGO\_SECTOR\_N11",F.col("CODIGO\_SECTOR\_N1"))\

.withColumn("VENTAS\_NETAS\_121",F.col("VENTAS\_NETAS\_12")\

.cast(DoubleType()))\

.withColumn("VENTAS\_NETAS\_01",F.col("VENTAS\_NETAS\_0")\

.cast(DoubleType()))\

.withColumn("EXPORTACIONES1",F.col("EXPORTACIONES")\

.cast(DoubleType()))\ .withColumn("COMPRAS\_NETAS\_121",F.col("COMPRAS\_NETAS\_12")\

.cast(DoubleType()))\

.withColumn("COMPRAS\_NETAS\_01",F.col("COMPRAS\_NETAS\_0")\

.cast(DoubleType()))\

.withColumn("IMPORTACIONES1",F.col("IMPORTACIONES")\

.cast(DoubleType()))\

.withColumn("COMPRAS\_RISE1",F.col("COMPRAS\_RISE")\

.cast(DoubleType()))\

.withColumn("TOTAL\_COMPRAS1",F.col("TOTAL\_COMPRAS")\

.cast(DoubleType()))\

.withColumn("TOTAL\_VENTAS1",F.col("TOTAL\_VENTAS")\

.cast(DoubleType()))\ .select("ANIO1","MES1","PROVINCIA1","CANTON1","CODIGO\_SECTOR\_N11","VENTAS\_NETAS\_121","VENTAS\_NETAS\_01",\ "EXPORTACIONES1","COMPRAS\_NETAS\_121","COMPRAS\_NETAS\_01","IMPORTACIONES1","COMPRAS\_RISE1",\

"TOTAL\_COMPRAS1","TOTAL\_VENTAS1","\_metadata.\_id")

Con los datos entrantes de Kafka y la lectura del índice de declaraciones de ElasticSearch se genera una consolidación de ambas fuentes a través de dos operaciones de escritura sobre la base de datos: la operación *append* para la inserción de registros cuyas variables categóricas sean nuevas en la base de datos; y la operación *update,*  para en caso de que ya existan registros para un conjunto de variables categóricas, se proceda a sumar con los datos entrantes.

* **Escritura de nuevos registros:** Se realiza un *join* tipo *left\_anti* entre los datos ingestados por Kafka y la *data* de ElasticSearch, el resultado de esa juntura se define como datos no existentes en la base de datos y por lo tanto se insertan. La Figura 20 representa el tipo de juntura *left\_anti,* entendiéndose como aquellos registros que provienen de Kafka que no fueron encontrados en la base de datos a través de las variables de juntura: AÑO, MES, PROVINCIA, CANTON, CODIGO\_SECTOR\_N1.

Figura 20 Operación left\_anti join

datos\_nuevos=datos\_entrantes.join(declaraciones, (datos\_entrantes.ANIO==declaraciones.ANIO1)\

& (datos\_entrantes.MES==declaraciones.MES1)\ &(datos\_entrantes.PROVINCIA==declaraciones.PROVINCIA1)\ &(datos\_entrantes.CANTON==declaraciones.CANTON1)\ &(datos\_entrantes.CODIGO\_SECTOR\_N1==declaraciones.CODIGO\_SECTOR\_N11)\

, **'left\_anti'**)\ .select('ANIO','MES','PROVINCIA','CANTON','CODIGO\_SECTOR\_N1','VENTAS\_NETAS\_12','VENTAS\_NETAS\_0','EXPORTACIONES',\ 'COMPRAS\_NETAS\_12','COMPRAS\_NETAS\_0','IMPORTACIONES','COMPRAS\_RISE','TOTAL\_COMPRAS','TOTAL\_VENTAS')

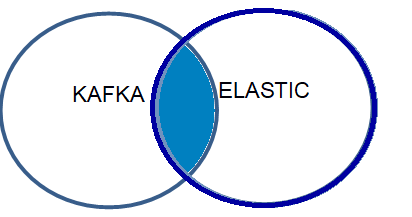
datos\_nuevos.write.format("org.elasticsearch.spark.sql").mode('**append**') \

.option("es.nodes", "http://192.168.1.52:9200") \

.save("declaraciones\_2022")

* **Escritura de registros existentes:**  Para ello se realiza un *inner join* entre los datos provenientes de Kafka y Elastic. La Figura 21 bosqueja una operación *inner,* siendo la intersección entre las dos fuentes de datos o aquellos datos cuyas variables de juntura (variables categóricas) coindicen. Si se encuentran coincidencias, se suman los valores numéricos de ambos *dataframes* y se actualiza a través de campo \_id (clave primaria o registro único) del índice.

Figura 21 Operación Inner Join

****

es\_modificacion = {"es.mapping.id": "\_id","es.mapping.exclude": "**\_id**",

"es.write.operation": "**update**","es.resource" : "declaraciones\_2022"}

datos\_modificables=datos\_entrantes.join(declaraciones, (datos\_entrantes.ANIO==declaraciones.ANIO1)\

& (datos\_entrantes.MES==declaraciones.MES1)\

& (datos\_entrantes.PROVINCIA==declaraciones.PROVINCIA1)\

& (datos\_entrantes.CANTON==declaraciones.CANTON1)\

& (datos\_entrantes.CODIGO\_SECTOR\_N1==declaraciones.CODIGO\_SECTOR\_N11)\

, '**inner**')

datos\_modificables=datos\_modificables.withColumn('VENTAS\_NETAS\_12',F.col('VENTAS\_NETAS\_12')+F.col('VENTAS\_NETAS\_121'))\

.withColumn('VENTAS\_NETAS\_0',F.col('VENTAS\_NETAS\_0')+F.col('VENTAS\_NETAS\_01'))\

.withColumn('EXPORTACIONES',F.col('EXPORTACIONES')+F.col('EXPORTACIONES1'))\

.withColumn('COMPRAS\_NETAS\_12',F.col('COMPRAS\_NETAS\_12')+F.col('COMPRAS\_NETAS\_121'))\ .withColumn('COMPRAS\_NETAS\_0',F.col('COMPRAS\_NETAS\_0')+F.col('COMPRAS\_NETAS\_01'))\

.withColumn('IMPORTACIONES',F.col('IMPORTACIONES')+F.col('IMPORTACIONES1'))\

.withColumn('COMPRAS\_RISE',F.col('COMPRAS\_RISE')+F.col('COMPRAS\_RISE1'))\ .withColumn('TOTAL\_COMPRAS',F.col('TOTAL\_COMPRAS')+F.col('TOTAL\_COMPRAS1'))\

.withColumn('TOTAL\_VENTAS',F.col('TOTAL\_VENTAS')+F.col\

('TOTAL\_VENTAS1'))\

.select('ANIO','MES','PROVINCIA','CANTON','CODIGO\_SECTOR\_N1'\

,'\_id','VENTAS\_NETAS\_12','VENTAS\_NETAS\_0','EXPORTACIONES','COMPRAS\_NETAS\_12','COMPRAS\_NETAS\_0',\ 'IMPORTACIONES','COMPRAS\_RISE','TOTAL\_COMPRAS','TOTAL\_VENTAS')

datos\_modificables.write.format("org.elasticsearch.spark.sql") \

.options(\*\*es\_modificacion) \

.mode('append') \

.save()

**Ejecución y pruebas**

Durante la ejecución del desarrollo para integrar los componentes se presentaron algunos errores al momento de integrar Kafka, Spark y ElasticSeach al realizar un procesamiento continuo de los datos capturados desde la primera e interactuar con la base de datos.

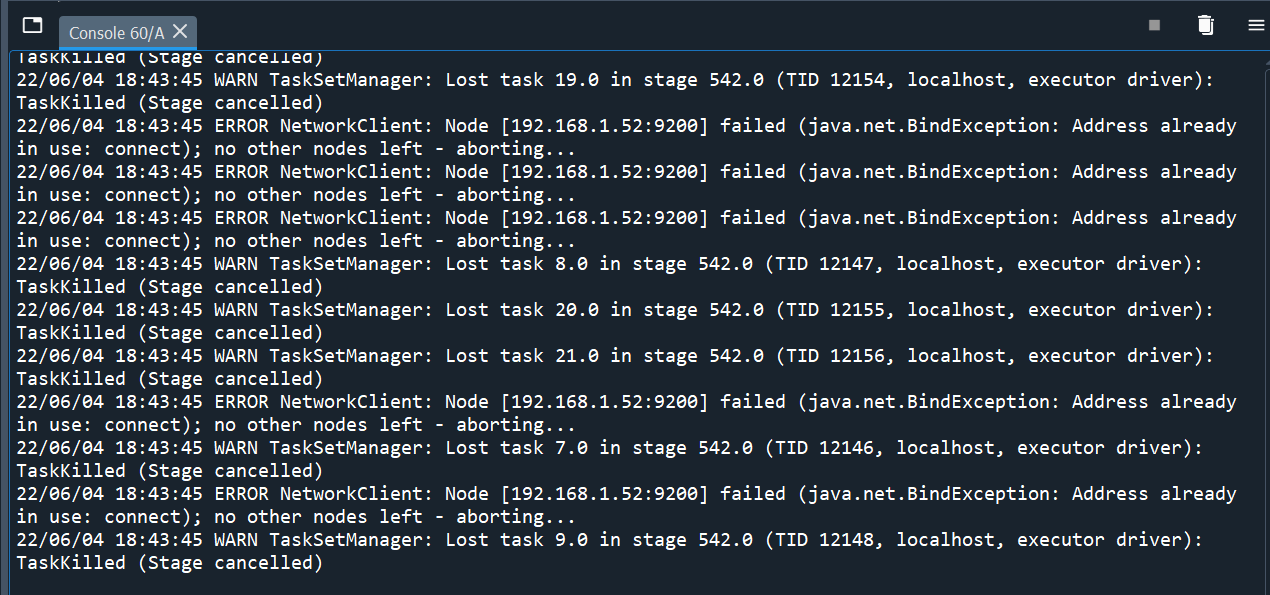
El código inicial se describe de la siguiente manera:

escritura=parsedDF.writeStream.foreachBatch(CONEXION\_ELASTIC).start()

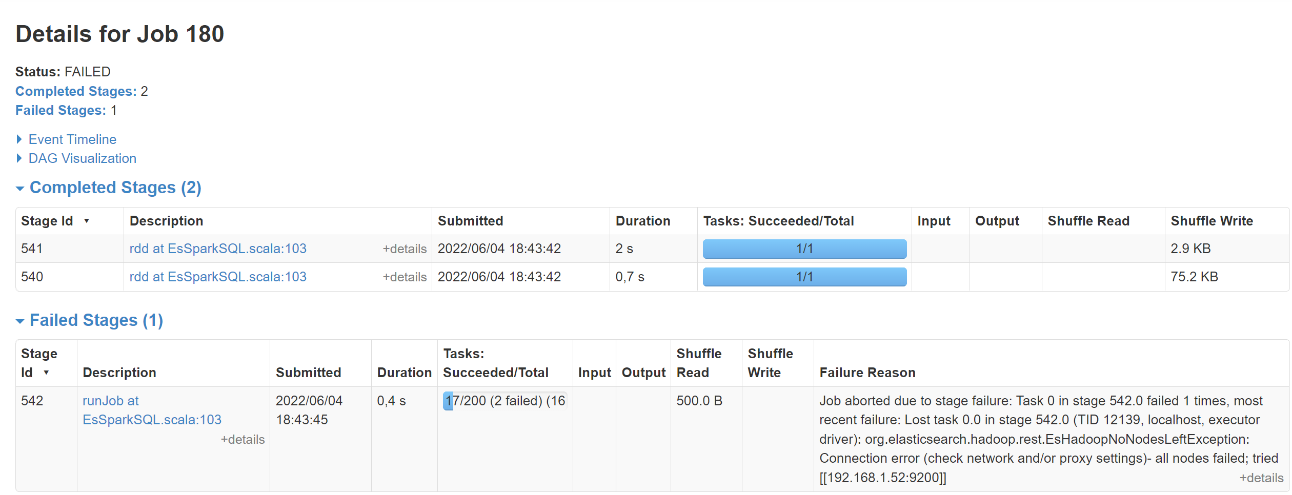
Este, a diferencia del código definitivo descrito anteriormente, no contiene la sentencia trigger(processingTime='30 seconds') por lo que el procesamiento será continuo y la conexión a la base de datos es ininterrumpida por cada registro que proviene desde Kafka. La Figura 22, despliega los errores desde la consola de Spyder/Python (a) y la interfaz web de Spark (b), donde se detalla que el nodo que corresponde al servidor de ElasticSearch (192.168.1.52:9200) se ha caído, provocando que la tarea ejecutada desde Spark falle.

Figura 22 Errores en ejecución continua de datos

(a)



(b)

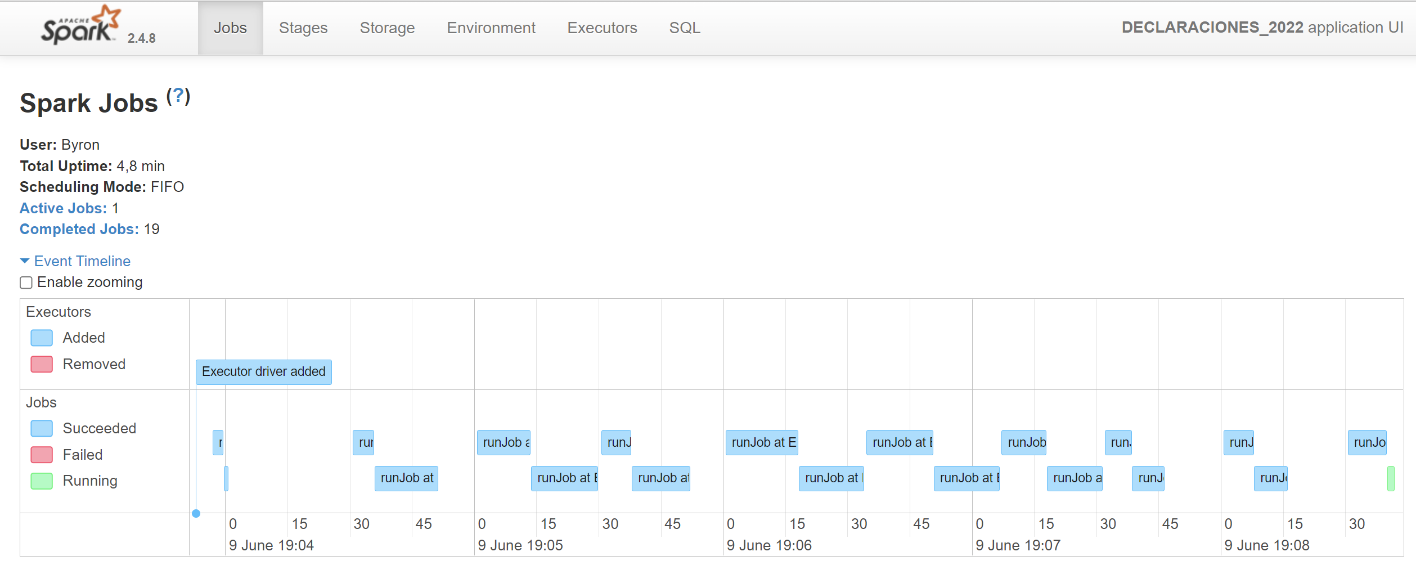


Este tipo de errores se producen debido a las limitaciones de red del nodo que aloja la base de datos; al tratarse de un computador, el número de sockets de conexión para dar respuesta a las solicitudes de lectura y escritura a la base de datos desde Spark es bajo, provocando la caída del *host*  por agotamiento TCP, lo que se traduce a la incapacidad de dar respuestas continuas a Spark por cada registro generado de declaraciones.

Es por ello, que para paliar las limitaciones de red de equipo con ElasticSearch, y el tiempo en que cada socket permanece ocupado aún después de finalizar cada operación de lectura y escritura sobre Elastic, se empleó la sentencia trigger(processingTime='30 seconds'), que implica el procesamiento en *micro batchs* de todos los registros de declaraciones que se generen en un periodo de 30 segundos, y su posterior interacción con Elastic. Aquí es importante resaltar que un tiempo mucho menor o incluso un procesamiento continuo provocará un agotamiento TCP.

En la Figura 23, se visualiza la ejecución de los *jobs*  lanzados desde Spark en distintos intervalos de tiempo con resultado exitoso.

Figura 23 Interfaz Spark - Jobs en Microbatch



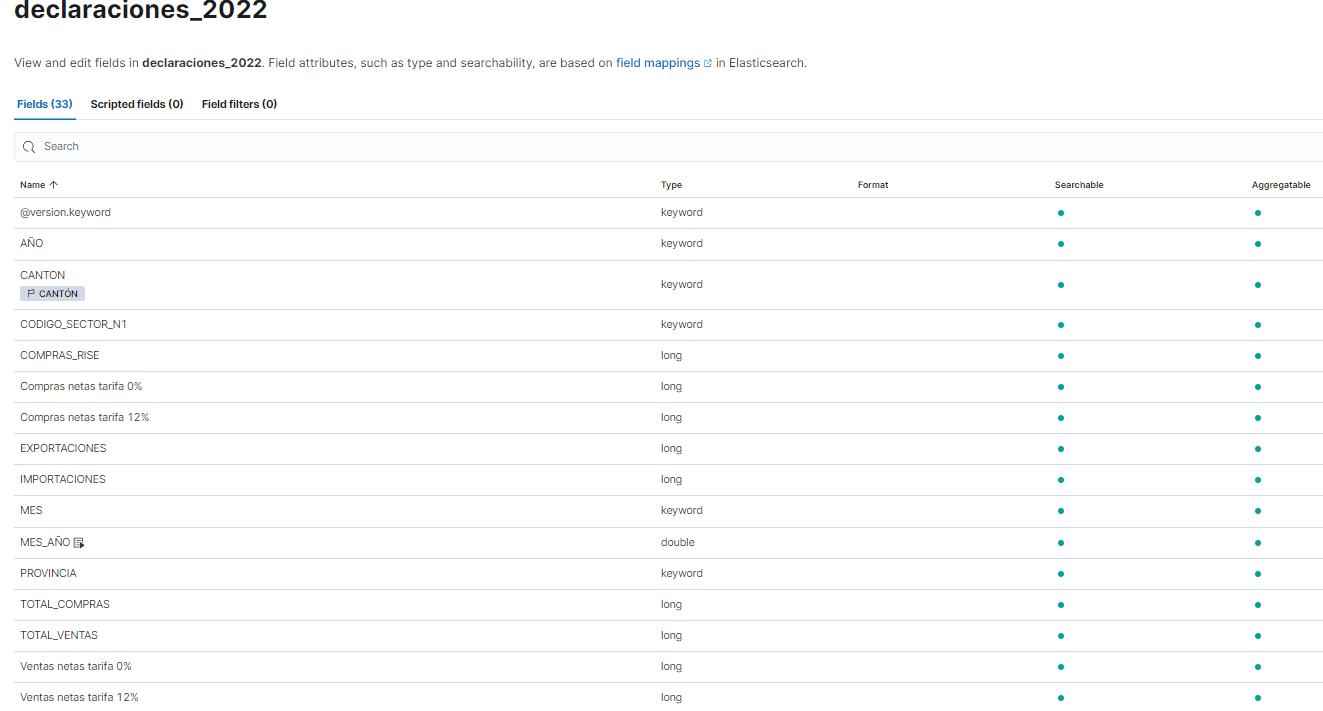
# 4.5. Cuadro de Mando en tiempo real

Para la construcción de reportes y cuadros de mandos se utilizó la herramienta Kibana, que se integra con la base de datos de ElasticSearch, la cual contiene funcionalidades para la creación y edición de *dashboards ,* manejo de filtros y navegación. Al igual que en la sección para la carga de la información histórica de declaraciones del periodo 2020-2021, se creó un índice (tabla) llamado declaraciones\_2022, que almacenará la *data* autogenerada del año 2022; este índice tiene la misma estructura a nivel de campos del índice declaraciones.

Para la construcción de los reportes, se definen las siguientes actividades:

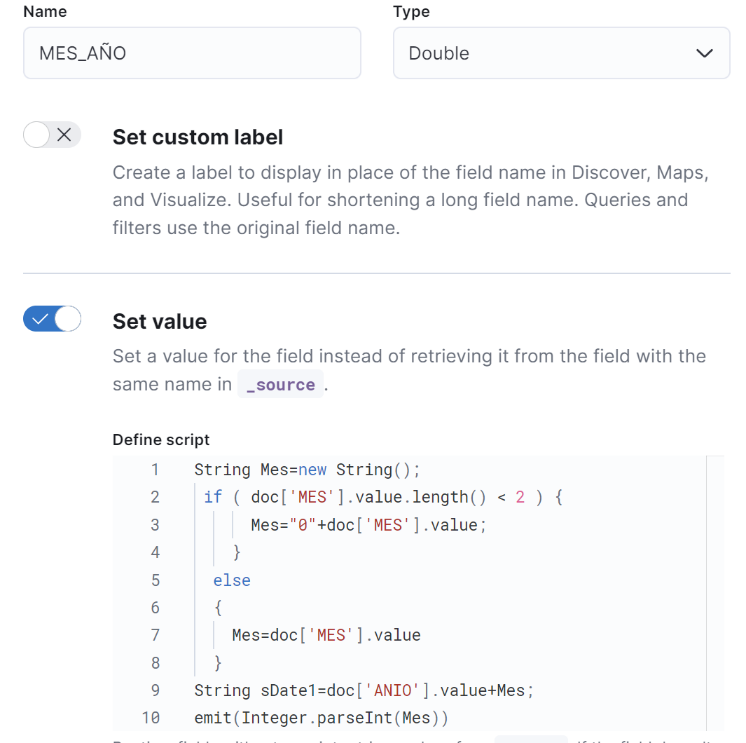
* **Creación de *View*:**  El *View* corresponde a una capa semántica con la definición a nivel de usuario de los campos constituyentes del índice, transformaciones y campos calculados. La Figura 24 muestra la vista declaraciones\_2022, con los campos proveniente del índice y campos calculados. En esta se creó la etiqueta CANTÓN para el campo CANTON.

Figura 24 Vista declaraciones\_2022



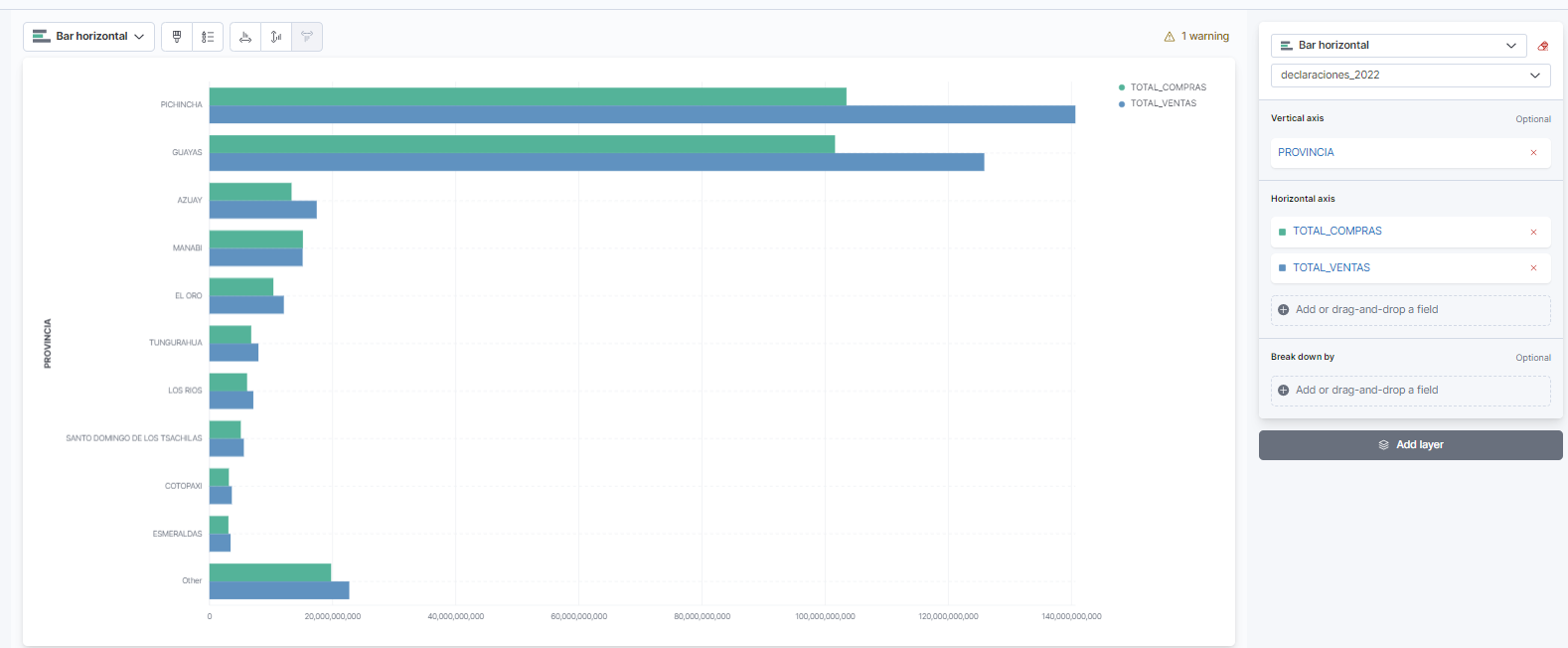
Adicional, se creó un campo calculado denominado MES\_AÑO, de tipo *double,*  que corresponde a la transformación a formato numérico del campo tipo *keyword* “MES”. Este nuevo campo permitirá el ordenamiento de los meses de forma numérica (1,2,3,…..12) y no en de tipo alfanumérica (1,11,12,2,3,4….) , de esta forma se podrá crear visualizaciones como gráfica de líneas para ver la evolución en el tiempo de determinadas métricas. La Figura 25 contiene la definición del campo calculado MES\_AÑO y el código Java para su creación.

Figura 25 Campo calculado MES\_AÑO



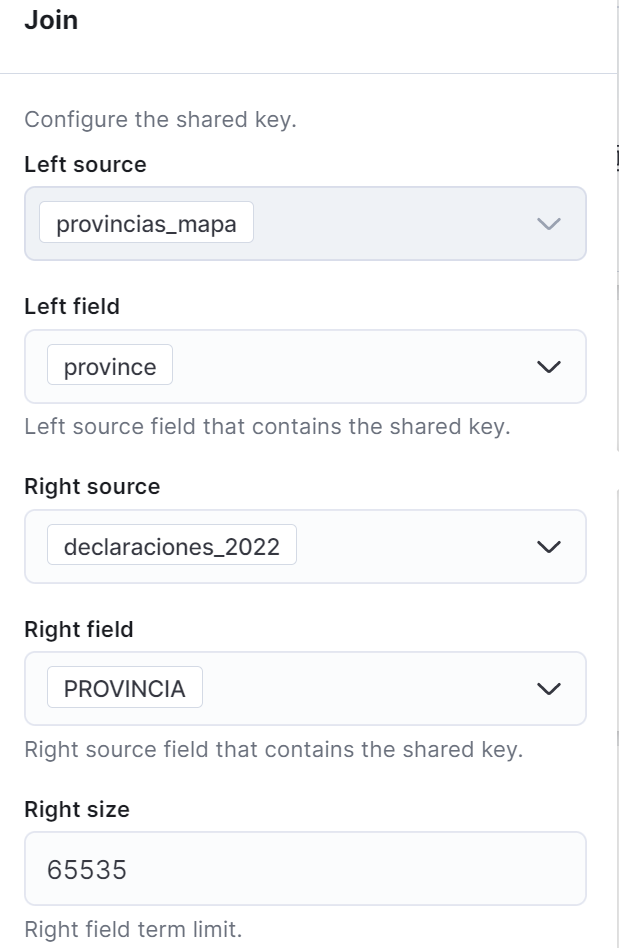
* **Creación de Visualizaciones:**  Kibana pone a disposición varios tipos de visualizaciones como gráficos de barras, pasteles (*donuts*), líneas, tablas o incluso mapas geográficos. Para el *dashboard*  en tiempo real, se crearon varias visualizaciones, cuya creación es similar. Para todos los gráficos, a excepción de los mapas, bastará con escoger el tipo de gráfico y los atributos (campos) que serán representados. La Figura 26, representa un gráfico de barras horizontales hecho en Kibana para la sumatoria de los campos TOTAL\_COMPRAS y TOTAL\_VENTAS por PROVINCIA.

Figura 26 Gráfico de barras en Kibana



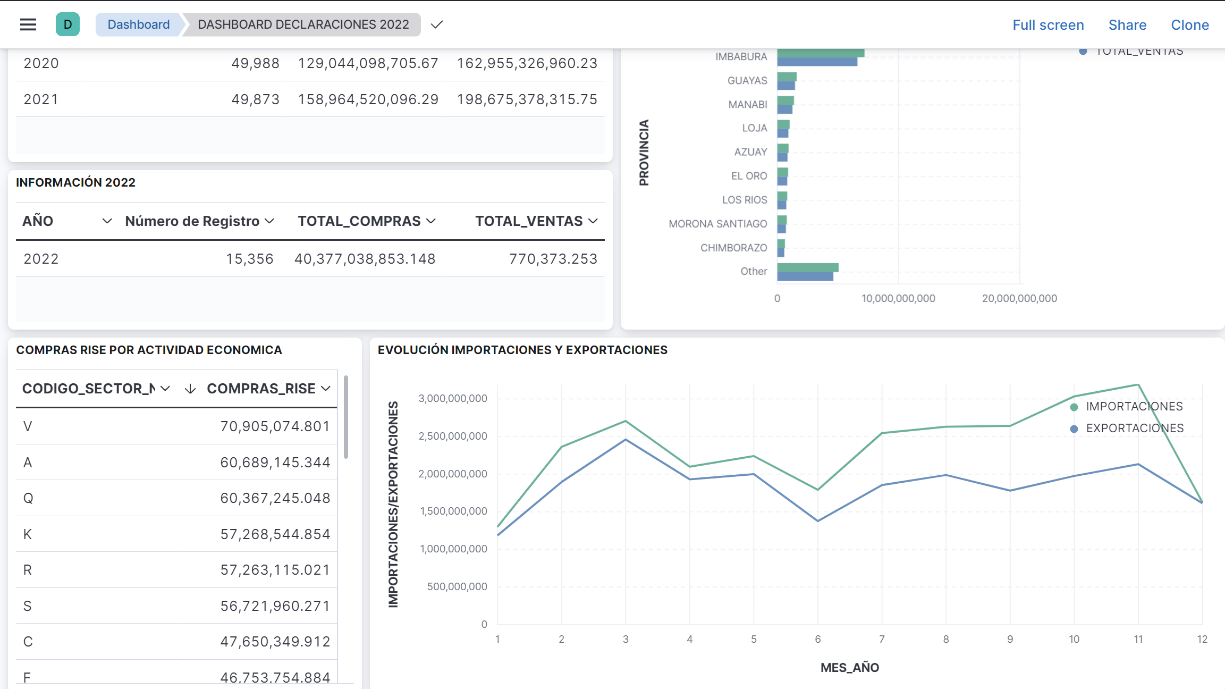
En el caso de los mapas, cuya ventaja radica en facilitar al usuario una mejor comprensión cognitiva de los datos a través de atributos geográficos; esto a partir de un fichero .geoson con la información de las localidades a visualizar. Este .geojson se importará en Elastic y generará un nuevo índice que permitirá a través de un *join,* cruzarse con la información del índice de declaraciones. La Figura 27 representa el *join*  entre los índices provincias\_mapa y declaraciones\_2022.

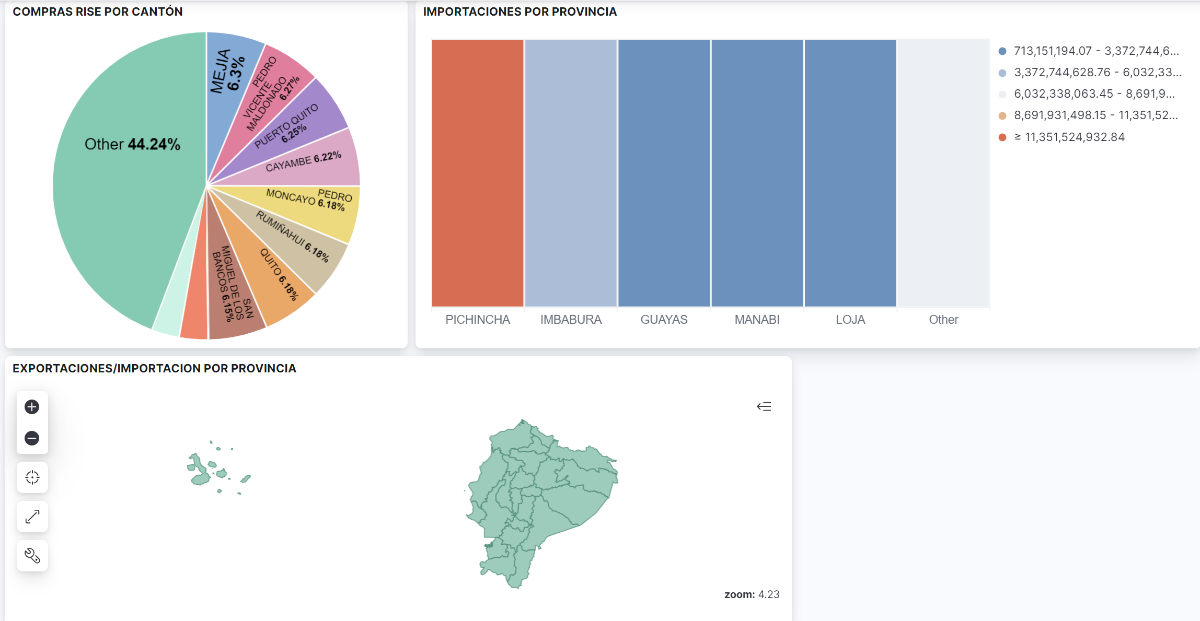
Figura 27 Join entre índice geográfico y declaraciones



* **Creación de *dashboard:*** Un *dashboard* constituye un conjunto de visualizaciones; en la Figura 28 representa el cuadro de mando construido para representar la información de declaraciones entre los que consta tablas, gráficas de barras y líneas, un pastel, un mapa de calor y un mapa geográfico.

Figura 28 Dashboard declaraciones 2022

****

****

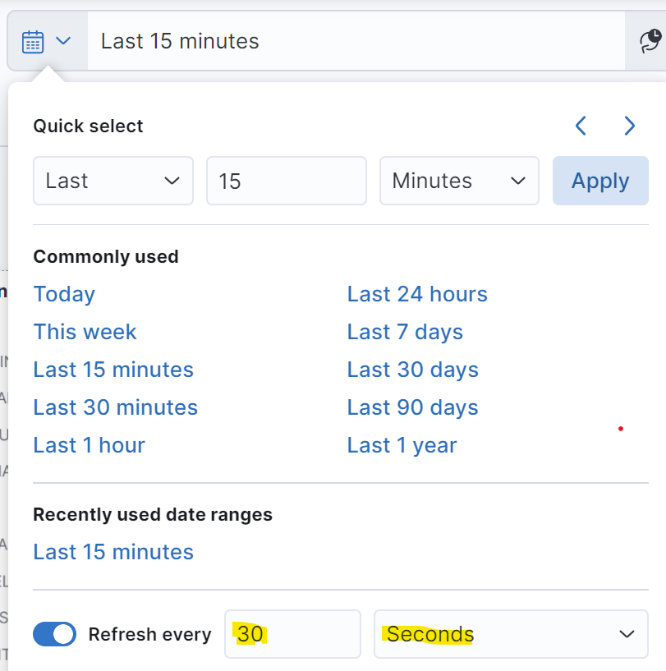
La Tabla 12 contiene la descripción de las visualizaciones que componen el *dashboard* de declaraciones, donde se define el tipo de gráfico y su descripción.

Tabla 12 Componentes del Dashboard de Declaraciones

|  |  |
| --- | --- |
| **TIPO DE GRÁFICO:NOMBRE** | **DESCRIPCIÓN** |
| **Tabla**: INFORMACIÓN HISTÓRICA 2020-2021 | Información histórica de declaraciones del periodo 2020-2021. Esta *data*  se extrae del índice declaraciones. |
| **Tabla**: INFORMACIÓN 2022 | Contiene información del número de registros agregados de declaraciones 2022 así como la sumatoria de los campos TOTAL\_COMPRAS y TOTAL\_VENTAS |
| **Tabla**: COMPRAS RISE POR ACTIVIDAD ECONÓMICA | Tabla con la sumatoria de COMPRAS\_RISE por el tipo de Actividad Económica |
| **Barras Horizontales**:Total Compras/Ventas por provincia en el 2022 | Información de Total Compras y Total Ventas por provincia. |
| **Líneas**: EVOLUCIÓN IMPORTACIONES Y EXPORTACIONES | Evolución de Importaciones y Exportaciones por cada mes del año 2022 |
| **Pastel**: COMPRAS RISE POR CANTÓN | Información de Compras Rise por cantón |
| **Mapa de Calor**: IMPORTACIONES POR PROVINCIA | Información del total de importaciones por provincia; el mapa de calor parte con una barra de color rojo para los montos más altos, hasta un color blanco para el monto menor. |
| **Mapa Geográfico**: EXPORTACIONES/IMPORTACIONES POR PROVINCIA | Un mapa geográfico del Ecuador dividido administrativamente por provincias creado a partir de un índice geográfico, el cuál despliega el monto de importaciones y exportaciones. |

Para que el *dashboard* construido represente la información de declaraciones que se carga en tiempo real, se deberá configurar para que su refrescamiento sea periódico. En la Figura 29 se visualiza la configuración para que el reporte se actualice cada 30 segundos; de esta forma se visualizará como los valores de las gráficas constantemente cambian a medida que se van ingestando los datos.

Figura 29 Refrescamiento de dashboard declaraciones

****

# 4.6. Clusterización

En esta sección se abordará el uso de técnicas de *Machine Learning* para la identificación de instancias (cantones) con atributos similares, para ello se emplea la librería de Python: Scikit-learn la cual cuenta con varios algoritmos no supervisados, entre ellos la clusterización.

Para la segmentación de localidades e identificación de sus realidades en el marco de declaraciones: compras, ventas, exportaciones, importaciones y demás, se ejecutaron varias fases sobre los datos agregados de declaraciones, los mismos que serán explicados a detalle más adelante; sin embargo, es imperativo abordar el uso de la técnica de Análisis de Componentes Principales, PCA (*Principal Components Analysis* ) por sus siglas en inglés, que aporta una mayor facilidad a la hora de trabajar con los atributos de un conjunto de datos. Se adjunta el Anexo V, que explica el funcionamiento del Análisis de Componentes Principales a través de un lenguaje de programación.

### 4.6.1 Análisis y procesamiento de la información

Para la construcción de procesos que involucren la incorporación de soluciones basadas en datos, entre ellas modelos de Inteligencia Artificial, se requieren de una serie de etapas que empiezan con un análisis exploratorio de los datos que permita comprender su estructura, variables y tipología, e identificar posibles errores de duplicación, inconsistencia o campos con valores en nulo.

A continuación, llega una etapa de preprocesamiento de los datos, cuya misión radica en tareas como limpieza de datos (*data cleansing*), consolidación o enriquecimiento del conjunto de datos con otras fuentes o catálogos de información que ayuden a tener un mayor espectro sobre la problemática a tratar o lo que se busca con la solución. Es significativo también mencionar procesos de normalización y estandarización para obtener datos consistentes y que permitan ajustar las escalas de las distintas variables en función de una escala/unidad común, así como técnicas que faciliten reducir la dimensionalidad de los datos.

Finalmente, con un *dataset*  limpio y consistente, se aplica una técnica de *Machine Learning* para identificar patrones y conductas que resultan de utilidad para la toma de decisiones, y la concerniente comunicación y discusión de los resultados. La Figura 30 subraya las etapas y tareas ejecutadas para el desarrollado de un modelo de clusterización de los datos de declaraciones tributarias agregados aplicando el lenguaje de programación Python.

Figura 30 Pasos para la aplicación de modelos de Clusterización

**ANALISIS EXPLORATORIO DE DATOS**

-Importación de set de datos

-Descripción de variables y estadísticas

**PREPROCESAMIENTO**

-Limpieza de datos

-Consolidación con otros catálogos de datos. - Escalamiento y PCA

**CLUSTERIZACION**

-Aplicación de algoritmos de clusterización

-Comparación de modelos

**COMUNICACIÓN DE RESULTADOS**

* Conclusiones y discusiones

Las distintas etapas constituyen un proceso de naturaleza cíclica, es decir que se puede llegar a resultados inesperados que obliguen a ir hacia una etapa anterior con el propósito de corregir errores u omisiones en esta.

Como se ha mencionado el lenguaje empleado para el desarrollo de este apartado fue Python y la plataforma de computación Jupyter, a través de la importación de distintas librerías. Dentro de esta memoria se adjunta un breve anexo para configurar el ambiente necesario para el uso de librerías como Geopandas.

* **ANÁLISIS EXPLORATORIO DE DATOS**

**Importación de datos:** Empleo de librerías como pandas, para importar el conjunto de datos:

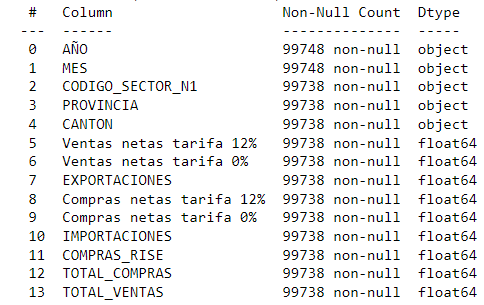
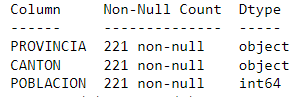
* Datos agregados de declaraciones agregados años 2020 y 2021: sri\_ventas\_2020L.csv y sri\_ventas\_2021L.csv desde la página del Servicio de Rentas Internas del Ecuador.
* Archivo con información sobre provincias y cantones del Ecuador, 221 registros en total y el número de habitantes estimado al año 2020: poblacion\_ecu\_2020.csv, provista por Instituto Nacional de Estadísticas y Censos.

**Exploración de datos:** Se obtiene información sobre el tipo de variables, que constituyen los datos importados, verificación de nulos y extracción de estadísticas: medias, desviación estándar, e/o.

La Figura 31 describe información sobre los campos y tipos de los datos de declaraciones de los años 2020-2021 (a) y de población (b), con 99748 y 221 registros respectivamente.

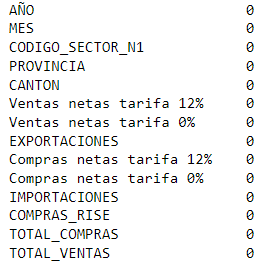
Figura 31 Descripción de variables de los datasets importados

1. (b)



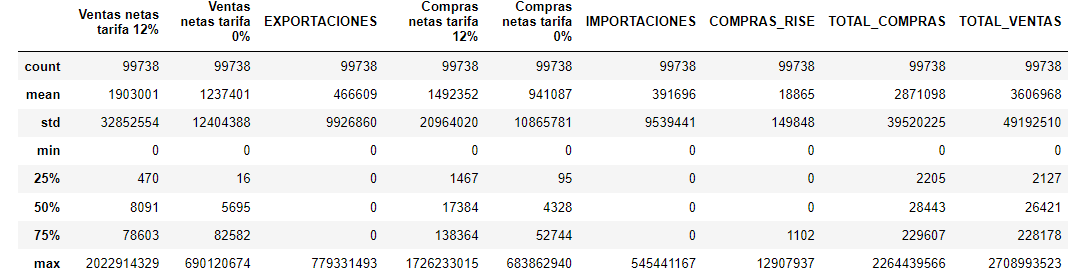
Para el caso de los datos de población no se encontraron nulos; sin embargo se verificó tal cual muestra la Figura 32, que los datos de declaraciones contenían 10 registros nulos; que posterioremente fueron eliminados al corroborarse que correspondían a saltos de línea.

Figura 32 Identificación y eliminación de datos nulos

Se constató también la no existencia de datos duplicados. Con la data depurada, se prosigue a obtener la estadísticas de la sobre los datos de declaraciones, en la Figura 33 se visualiza un detalle de los campos numéricos de las declaraciones, con medidas relevantes como la media, desviación estándar y cuartiles.

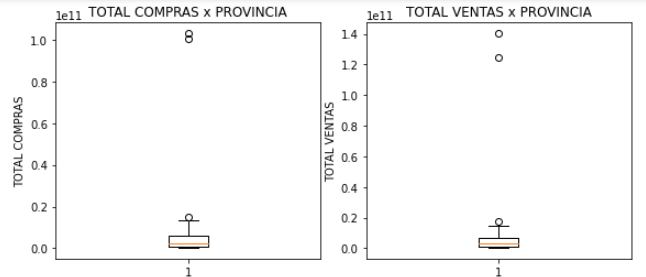
Figura 33 Estadísticas datos numéricos de Declaraciones



El despliegue de información de estadísticas es importante para realizar un análisis descriptivo, este se complementa con gráficos que facilitan su comprensión y comportamiento; la Figura 34, con las gráficas de cajas con información de las variables: Total Compras (a) y Total Ventas (b) por provincia se puede verificar 2 datos atípicos; sin embargo, estos representan información de Guayas y Pichincha, las provincias más grandes y pobladas del país. El resto de las provincias se encuentran dentro del primer y tercer cuartil.

Figura 34 Gráficas de cajas de los totales de compras y ventas por provincia

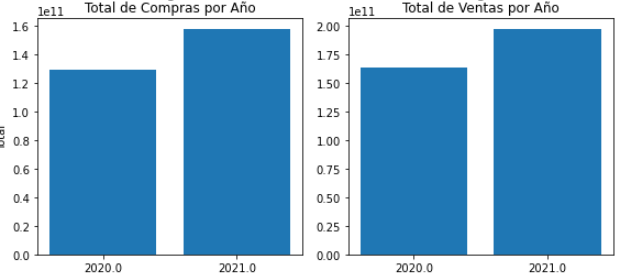
1. (b)



Se procede también con la obtención de los montos totales de compras y ventas por año, la Figura 35 evidencia que para ambas variables el año 2021 los montos son mayores; pudiendo atribuirse este crecimiento a la reapertura y flexibilización de medidas para contener la epidemia del COVID-19 que impulsaron una recuperación económica.

Figura 35 Total de compras y ventas por año

1. (b)



* **PREPROCESAMIENTO DE DATOS**

En esta etapa, se cumplieron tareas que involucra enriquecer, consolidar y estandarizar el conjunto de datos originales:

**Consolidación de datos:**  Unión (*Join*) entre las fuentes de declaraciones agregadas por **provincia y cantón** y la data de población. Esto enriquece los datos al contar con información sobre variables económicas como Total Compras, Total Ventas, e/o de una determinada localidad y la población de esta.

**Escalamiento:**  Con el propósito de obtener medidas relativas, que conlleven a un mayor análisis sobre la realidad de cada localidad ecuatoriana respecto a su número de habitantes, se crearon variables numéricas adicionales, mediante la siguiente fórmula:

CM\_VARIABLE=,

Donde la nueva variable calculada (ejemplo: CM\_TOTAL\_COMPRAS) representa la tasa por cada 10000 habitantes; con una variable original (ejemplo: TOTAL\_COMPRAS) sobre la población total de un cantón y multiplicado por 10000.

**Normalización:** Algoritmos como Análisis de Componente Principales y la mayoría de aquellos utilizados para lenguaje de máquina recomiendan la normalización de los datos para el tratamiento de las variables que puedan representar distintas unidades de medida.

**Análisis de Componentes Principales:**  Se aplicó un PCA para reducir la dimensionalidad de los datos, al contar 7 campos numéricos (tasas por cada 10000 habitantes) resulta complejo determinar si existen datos altamente correlacionados o redundantes; la “excesiva” cantidad de atributos dificulta al menos en el marco visual hacer un análisis más completo a través de algoritmos de clusterización. La Figura 36 indica que se tiene hasta el 83% de información (varianza) al obtener los dos primeros componentes principales.

Figura 36 Varianza acumulada por componentes



De esta forma se procede a realizar los cálculos para resumir el set de datos original de siete variables en un plano bidimensional.

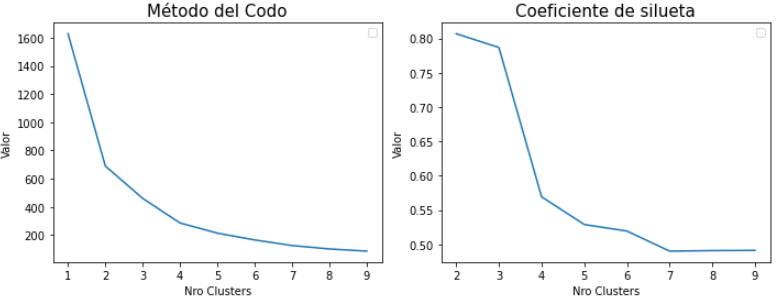
* **CLUSTERIZACIÓN**

**K-Means:**  La aplicación de modelos de clusterización tienen como hiperparámetro el número de clústeres; su determinación está basado en medidas como el coeficiente de silueta para medir el nivel de agrupamiento de los clústeres obtenidos; así como el método del codo para medir en qué punto (número de clúster) existe un cambio brusco de la evolución de una recta (inercia). Estas medidas al igual que la observación posterior de los clústeres identificados son un aporte para evitar resultados no deseados.

La Figura 37 describe el método del codo (a), observándose un cambio brusco a partir de contar con dos clústeres. Y la evolución del coeficiente de silueta (b) que muestran un valor mayor al 77% para agrupamientos de dos y tres clústeres, cayendo drásticamente su valor para un mayor número de clústeres.

Figura 37 Definición del número de clústeres mediante el método del codo y coeficiente de silueta

1. (b)



La Figura 38 despliega dos diagramas de dispersión de los dos componentes principales calculados anteriormente: (a) para dos y (b) tres clústeres. Se puede concluir que para el caso de tres clústeres se tiene una única instancia, la misma que podría ser tratadas como un atípico o una localidad donde los efectos de la pandemia fueron menores al resto de cantones. Para el clúster 0 (café) se tiene 207 cantones, mientras que para el clúster1(azul) 13 cantones.

Figura 38 Representación de las instancias para 2 y 3 clústeres calculados con K-Means

1. (b)

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Si se compara gráficamente los tres clústeres sobre la media de cada uno de los valores numéricos, la Figura 39 despliega valores completamente diferentes en cada uno de los clústeres. Para los atributos TOTAL\_COMPRAS\_CM y TOTAL\_VENTAS\_CM se observa al clúster 3(turquesa) con los valores más altos, mientras que el clúster 1(azul) que engloba el mayor número de cantones del país tiene las medias más bajas de compras y ventas.

Figura 39 Resumen gráfico de declaraciones por cada clúster

Gráfico, Gráfico de barras

Descripción generada automáticamente

**Aglomerativo:** La aplicación de un algoritmo de clusterización tipo aglomerativo requiere configurar el número de clústeres deseado, su correcta definición viene dada al igual que K-Means a través del análisis del coeficiente de silueta, la visualización de las gráficas de dispersión, y en particular el uso de un gráfico de dendograma para observar un conjunto de subcategorías que parten del mínimo detalle (instancia) y cómo se van agrupando en categorías más grandes.

La Figura 40 genera un dendograma (a) donde se observa las distintas categorías que van agrupando las instancias con dos rectas horizontales trazadas para identificar dos (recta roja) y tres clústeres (recta azul). La parte (b) despliega la evolución para distintos clústeres del coeficiente de silueta con valores sobre el 77% para un número de clústeres mayor a tres y una tendencia a la baja para valores superiores.

Figura 40 Dendongrama y evolución del coeficiente de silueta

1. (b)

Al observar la distribución de las instancias(cantones) de acuerdo con la Figura 41, (a) para dos y (b) tres clústeres, se puede identificar que para este último existe una única instancia perteneciente al tercer clúster (turquesa); interpretándose de forma similar a lo acontecido en K-Means como un posible valor atípico o un cantón que tuvo la menor afectación durante la pandemia. Para el clúster 0 (azul) se tiene 207 cantones, mientras que para el clúster1(café) 13 cantones.

Figura 41 Representación de las instancias para 2 y 3 clústeres calculados con Algoritmo Aglomerativo

1. (b)

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

Si se compara gráficamente los tres clústeres sobre la media de cada uno de los valores numéricos, la Figura 42 despliega valores completamente diferentes en cada uno de los clústeres. Para los atributos TOTAL\_COMPRAS\_CM y TOTAL\_VENTAS\_CM se observa al clúster 3(turquesa) con los valores más altos, mientras que el clúster 0 (azul) que engloba el mayor número de cantones del país tiene las medias más bajas de compras y ventas.

Figura 42 Resumen gráfico de declaraciones por cada clúster

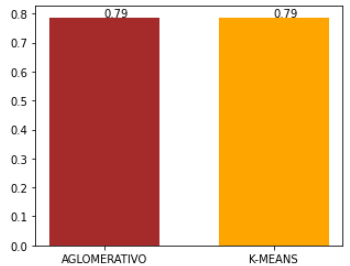
Gráfico

Descripción generada automáticamente

**Comparación de Modelos**

Al realizar una comparativa entre ambos algoritmos, K-Means y Aglomerativo, para un número de clústeres=3, la Figura 43 puede evidenciar que parámetros como el coeficiente de silueta, así como los diagramas de dispersión de las Figuras 38(b) y 41(b) resultan iguales. Por lo que para el conjunto de datos trabajado cualquiera de los modelos resulta idóneo para segmentar los cantones en base a sus características propias de declaraciones (compras, ventas, e/o) y su población.

Figura 43 Comparación de modelos: K-Means vs Aglomerativo



* **COMUNICACIÓN DE RESULTADOS**

La Figura 44, que despliega la media sobre cada una de las variables numéricas por cada clúster, indica que en todas las medidas exceptuando COMPRAS\_RISE\_CM (COMPRAS\_RISE por cada 10000 habitantes) el clúster 0 tiene valores menores a los clústeres 1 y 2 en ese orden.

Figura 44 Resumen de medias por métricas de declaraciones por cada clúster

Imagen que contiene Tabla

Descripción generada automáticamente

La Figura 45 permite ver un mapa administrativo del Ecuador, con los 221 cantones, lo cual permite visualizar con facilidad las instancias que conforman cada uno de los clústeres.

Figura 45 Mapa del Ecuador distribuido por clústeres

Imagen que contiene Mapa

Descripción generada automáticamenteImagen que contiene Mapa

Descripción generada automáticamente

Tal como se resaltó en el desarrollo del apartado la mayoría de los cantones tuvieron una mayor afectación por la pandemia del COVID-19, reflejándose tal en los menores valores de declaraciones con respecto a unas pocas localidades. Esto no solo se visualiza a nivel de la gran mayoría de cantones, sino también en el número de habitantes. Por ejemplo, la Figura 46 muestra que las ciudades con mayor población como Quito, Guayaquil o Cuenca se encuentran entre las localidades dentro del grupo con menores declaraciones. Esto presumiblemente se deba a que la pandemia como es conocido tuvo un mayor impacto social y sanitario en las grandes ciudades, que obligó a las autoridades a tomar medidas más radicales para su control.

Figura 46 Información de los 10 cantones más grandes del Ecuador

Tabla

Descripción generada automáticamente

### 4.6.3 Código Fuente Relevante

En la Tabla 13 se visualiza fragmentos importantes de código Python para el desarrollo de los modelos de clusterización:

Tabla 13 Código relevante Python para clusterización

|  |  |
| --- | --- |
| **FRAGMENTO CODIGO** | **DESCRIPCIÓN** |
| url\_datos\_2020= ' sri\_ventas\_2020L.csv'  data\_2020=pd.read\_csv(url\_datos\_2020, sep=";")  url\_datos\_2021= 'sri\_ventas\_2021L.csv'  data\_2021=pd.read\_csv(url\_datos\_2021 , sep=";")  url\_poblacion\_ecuador='poblacion\_ecu\_2020.csv'  data\_poblacion\_canton=pd.read\_csv(url\_poblacion\_ecuador, sep=";") | Uso de la librería pandas (pd) para la importación de archivos relaciones a la declaraciones tributarias y población; achivos con campos separados por “;” |
| data\_total.isna().sum()  data\_total.dropna(inplace=True)  data\_total.isna().sum()  data\_total[data\_total.duplicated(keep=False) | Sentencias para la identificación de registros en nulo y duplicados. La variable data\_total corresponde a los datos de declaraciones de los años 2020-2021 |
| data\_agregada\_canton=data\_total.groupby(["PROVINCIA",'CANTON'])[campos\_numericos].sum().reset\_index()  data\_agregada\_provincia2=data\_total.groupby(["PROVINCIA"])[campos\_numericos].sum().reset\_index() | Obtención de tablas resumen para la generación de un análisis descriptivo. |
| dataset\_consolidado\_provincia=pd.merge(data\_agregada\_provincia, data\_poblacion\_canton, on=['PROVINCIA','CANTON'], how='inner')  for campo in campos\_numericos\_10K:  dataset\_consolidado\_provincia[campo]=dataset\_consolidado\_provincia[campos\_numericos[contador\_campos\_numericos]]\*10000/dataset\_consolidado\_provincia['POBLACION'] | Consolidación de los datos de declaraciones con la población por cantones. Cálculo de nuevas variables numéricas por cada 10000 habitantes. |
| scaler=StandardScaler()  dataset\_normalizado=scaler.fit\_transform(dataset\_consolidado\_provincia\_Num)  pca=PCA(n\_components=2)  pca.fit(dataset\_normalizado)  pca.transform(dataset\_normalizado)  scores\_pca=pca.transform(dataset\_normalizado) | Normalización de los datos y aplicación de un algoritmo de Análisis de Componentes Principales |
| kmeans = KMeans(n\_clusters = nro\_clusters).fit(data\_for\_scatter) | Implementación de K-Means sobre los datos, para jerárquico aglomerativo se tiene una sentencia similar. |
| geo\_cantones = gpd.read\_file('cantones.geojson',encoding='utf-8')  datos\_final=gpd.GeoDataFrame(pd.merge(dataset\_consolidado\_provincia, geo\_cantones, on=['PROVINCIA','CANTON'], how='inner'))  datos\_final.plot(column = 'cluster1',figsize=(15, 10), categorical=True, legend=True) | Uso de la librería geopandas donde se importa un fichero tipo geojson que contiene la distribución geográfica de los cantones del Ecuador. Con esta data se consolida los datos resultantes una vez que se aplicaron los modelos de clusterización para su despliegue a través de un mapa geográfico que coloré los cantones en función de cada clúster identificado. |

# 

# 5. Conclusiones y trabajo futuro

## 5.1. Conclusiones

El presente Trabajo de Fin de Master consiste en el diseño e implementación de una solución tecnológica para la captura de información de declaraciones a detalle en tiempo real del año 2022, almacenados previamente en una base de datos transaccional Oracle, para su posterior procesamiento y persistencia mediante tecnologías Big Data como son: Kafka, un encolador de mensajes; el Framework de procesamiento y computación de datos Apache Spark; así como la base de datos NoSQL documental ElasticSearch. Además, se almacenó en esta última los datos públicos agregados de declaraciones del periodo 2020-2021; para ambos conjuntos de datos se crearon cuadros de mando utilizando Kibana. Finalmente, se construyó en base a los datos agregados del periodo 2020-2021 los modelos de clusterización K-Means y Jerárquico Aglomerativo para determinar grupos de cantones del Ecuador con similares características de declaraciones a fin de determinar el grado en que la pandemia del Covid-19 afectó a cada localidad en función de su realidad económica y poblacional.

Este proyecto permitió dilucidar las siguientes conclusiones:

* A pesar de concluir con éxito la implementación del TFM, se constató que la integración de las herramientas Spark y ElasticSearch es posible hasta la versión de Spark 2.4.X que soporta JDK 8; lo que sería una limitante en caso de querer emplear nuevas funcionalidades y mejoras disponibles en versiones de Spark más recientes (3.X) como la nueva interfaz de monitoreo de Spark Streaming o un Spark SQL más cercano a ANSI SQL.
* La lectura de *logs y consolas* que registran los errores al ejecutar una herramienta resultan de vital importancia a fin de resolverlos. Como se pudo evidenciar a lo largo de la implementación de este Trabajo Final de Master, la correcta lectura e interpretación de los registros de errores por cada una de las herramientas empleadas, permitió solventar inconvenientes como la incompatibilidad de versiones, agotamiento TCP o incluso la falta de conectividad entre los equipos que conformaron la arquitectura propuesta.
* Los algoritmos de clusterización K-Means y Jerárquico Aglomerativo brindaron similares resultados de rendimiento (coeficiente de Silueta) para los datos correspondientes a declaraciones 2020-2021; sin embargo, es importante recordar que esta similitud se da para estos datos en particular. Corresponderá realizar un análisis específico sobre qué algoritmo resulta más idóneo en caso de tener otras fuentes de datos.
* A través de la implementación de los algoritmos de clusterización, se puede concluir que la mayoría de cantones del Ecuador tuvieron una afectación afín debido a la pandemia del COVID-19, aunque es imperativo realizar un análisis basado en la realidad de cada localidad, considerando factores como la población o situación socioeconómica; esto permitirá la toma de mejores decisiones.

## 5.2. Líneas de trabajo futuro

* Para la implementación del TFM se utilizó dos computadores locales que tenían instaladas varias herramientas para la captura y procesamiento de la información de declaraciones, ocasionando que los recursos de *hardware* de cada equipo se vean limitados al tener que ser compartidos entre varios aplicativos. Es por ello que se presenta como una línea a trabajo futuro el uso de varios equipos computacionales (*clustering*) que cuenten con herramientas dedicadas por cada computador, así como la replicación y particionamiento de los datos generados en varios equipos.
* Se plantea como un posible trabajo a futuro, la implementación del presente TFM con tecnologías *cloud.* Es de sobra conocido que plataformas como Amazon AWS o Azure cuentan con herramientas que facilitan una rápida integración entre ellas y escalabilidad horizontal a través de la incorporación de nuevos nodos, esto daría paso a focalizarse únicamente en el desarrollo de la solución (código) en lugar de las tareas relativas a infraestructura y configuración de herramientas.
* Los datos generados de declaraciones 2022 fueron de tipo randómico, lo que imposibilita realizar un análisis profundo de esta información. Se abre la posibilidad de implementar un piloto que capture datos reales de declaraciones en *streaming,* así como la construcción de modelos de *Machine Learrning* para la detección de patrones o tendencias en tiempo real.
* Este proyecto se enfocó en información de declaraciones agregadas, donde se omite datos relativos de los contribuyentes; sin embargo, se plantea como un posible trabajo a futuro el trabajar con datos a detalle de declaraciones de impuestos a fin de que haciendo uso de técnicas de *Machine Learning* se permita identificar qué contribuyentes podrían incurrir en prácticas ilegales como la evasión y elusión fiscal; todo en marco de la protección y buen uso de los datos personales.

# 6. Bibliografía

Waehner, K. (25 de Febrero de 2022). *Kai Waehner*. Obtenido de https://www.kai-waehner.de/blog/tag/real-time/

Alvarez, J. (9 de Septiembre de 2018). *Blog de José Mariano Alvarez*. Obtenido de http://blog.josemarianoalvarez.com/2018/09/09/instalar-apache-spark-en-windows-10/

Apache Spark. (s.f.). *Spark*. Obtenido de https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

Banco Central del Ecuador. (30 de Noviembre de 2021). *https://www.bce.fin.ec/*. Obtenido de https://www.bce.fin.ec/index.php/boletines-de-prensa-archivo/item/1458-el-banco-central-actualiza-al-alza-su-prevision-de-crecimiento-para-2021-a-3-55

Bennett, E. (17 de Agosto de 2021). *Logit*. Obtenido de https://logit.io/blog/post/the-top-elasticsearch-use-cases

Burnay, C., Dargam, F., & Zarate, P. (2019). Special issue: Data visualization for decision‑making:. *Springer Nature*.

Castro, A., & Gonzalez, J. (2012). Utilidad y funcionamiento de las bases de datos NoSQL. *Red de Revistas Científicas de América Latina, el Caribe, España y Portugal*.

EDUCBA. (s.f.). *EDUCBA*. Obtenido de https://www.educba.com/

Elastic. (s.f.). *Elastic*. Obtenido de https://www.elastic.co/es/what-is/elk-stack

Enlyft. (s.f.). Obtenido de https://enlyft.com/tech/products/kibana

Feick, M., Kleer, N., & Kohn, M. (2018). Fundamentals of Real-Time Data Processing Architectures. *Lecture Notes in Informatics (LNI)*.

Fu, Y., & Chen, M. (20 de Diciembre de 2020). *Uber Engineering*. Obtenido de https://eng.uber.com/kafka/

Gupta, T. (06 de Diciembre de 2020). *medium*. Obtenido de https://medium.com/analytics-vidhya/fastest-way-to-install-geopandas-in-jupyter-notebook-on-windows-8f734e11fa2b

Hasani, Z. e. (2014). Lambda architecture for real time big data analytic. *ICT Innovations*.

Intellipaat. (17 de Octubre de 2020). *Intellipaat*. Obtenido de https://intellipaat.com/blog/a-brief-introduction-to-principal-component-analysis/

Kaloyanova, E. (12 de Diciembre de 2019). *365datascience*. Obtenido de https://365datascience.com/tutorials/python-tutorials/principal-components-analysis/

Kaloyanova, E. (10 de Marzo de 2020). *365DataScience*. Obtenido de https://365datascience.com/tutorials/python-tutorials/pca-k-means/

Li, K., Tiwari, A., Alcock, J., & Bermell-Garcia, P. (2016). Categorisation of visualization methods to support the design of Human-Computer Interaction Systems. . *Applied Ergonomics*, 55,85-107.

Microsoft. (02 de Mayo de 2022). *Microsoft*. Obtenido de https://docs.microsoft.com/es-es/dotnet/spark/what-is-spark

Moore, J. (2017). DATA VISUALIZATION IN SUPPORT OF EXECUTIVE DECISION MAKING. *Informing Science Institute*.

Nava, V. (10 de Marzo de 2016). *Qubole*. Obtenido de https://www.qubole.com/blog/apache-spark-use-cases/

Pedregosa, F., Varoquaux, G., & Gramfort, A. e. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning*.

programmerclick. (s.f.). *programmerclick*. Obtenido de https://programmerclick.com/article/4039326230/

ProjectPro. (31 de Marzo de 2022). *ProjectPro*. Obtenido de https://www.projectpro.io/article/top-5-apache-spark-use-cases/271

Santos, P. (22 de Marzo de 2019). *openwebinars*. Obtenido de https://openwebinars.net/blog/como-utilizar-spark-en-windows/

Sartorius. (18 de Agosto de 2020). Obtenido de https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186

Scikit-Learn. (s.f.). *Scikit-Learn*. Obtenido de https://scikit-learn.org/stable/testimonials/testimonials.html

Tuyishimire, E., Mabuto, W., & Gatabazi, P. (17 de Febrero de 2022). *MDPI*. Obtenido de https://www.mdpi.com/2078-2489/13/2/94/htm#B12-information-13-00094

Waehner, K. (23 de Septiembre de 2021). *https://www.kai-waehner.de/*. Obtenido de Kai Waehner: https://www.kai-waehner.de/

Xie, G., & Huang, Y. (24 de Julio de 2017). *Uber Engineering*. Obtenido de https://eng.uber.com/elk/

# Anexos

## Anexo I. Configuración Esquema BDD Oracle XE

* Descargar del sitio oficial de Oracle <https://www.oracle.com/mx/database/technologies/xe-downloads.html>, la versión Oracle Database Express Edition.
* Dentro de Windows la instalación de la base de datos toma algunos minutos, mediante la ejecución de simples clics en el botón Next y definiciones sencillas sobre la ruta donde se alojará la base de datos. Adicionalmente el instalador solicitará el ingreso de una clave de administrador, que es importante registrar para futuras configuraciones.
* Una vez que se instaló Oracle, se necesita de una interfaz gráfica (GUI) que permita loguearse a la base de datos y realizar la creación del esquema, permisos y objetos de base de datos; para ello se recomienda descargar del sitio oficial de Oracle la herramienta SQL Developer, la misma que viene en un comprimido conteniendo el ejecutor.
* Se procede a crear una conexión con el usuario system, este esquema es de tipo administrador, con privilegios para crear nuevos usuarios, roles, permisos, así como herramientas de monitoreo, optimización, e/o. Para ello, dentro de SQL Developer, se crea una nueva conexión mediante clic derecho sobre la opción Oracle Conexiones, seleccionando **Nueva Conexión,**  la Figura 47 detalla los datos de conexión hacia el esquema system, con el campo encriptado de Contraseña, que corresponde al valor de clave ingresado durante la instalación de Oracle.

Figura 47 Creación nueva conexión Oracle

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

* Una vez conectado en system, se debe definir un tablespace, que corresponde a un espacio de disco reservado, el cual almacenará toda la información de declaraciones que se genere.

CREATE TABLESPACE ts\_declara DATAFILE 'declarac.dat' SIZE 200M ONLINE;

* Se procede a crear el usuario de base de datos, junto a su contraseña de acceso, asociadas al tablespace creado.

alter session set "\_ORACLE\_SCRIPT"=true;

CREATE USER declaraciones IDENTIFIED BY declaraciones DEFAULT TABLESPACE ts\_declara;

* A continuación, se otorgan permisos al usuario: (1) creación de sesiones, (2) permiso para ampliar el tablespace en caso de que la información generada sobrepasa el valor de espacio físico inicial asignado, en adición a permisos para crear tablas (3) y procedimientos (4), secuencias(5) y disparadores (6)

GRANT CREATE SESSION to declaraciones; --(1)

GRANT UNLIMITED TABLESPACE TO declaraciones; --(2)

GRANT CREATE TABLE to declaraciones; --(3)

GRANT CREATE PROCEDURE to declaraciones; --(4)

GRANT CREATE sequence to declaraciones; --(5)

GRANT CREATE trigger to declaraciones; --(6)

## Anexo II. Instalación y configuración Elasticsearch, Logstash y Kibana (ELK)

Para la instalación y configuración de la plataforma ELK se deberá descargar del sitio web elastic.co/es/downloads las carpetas comprimidas para un determinado sistema operativo, para este TFM se optó por Windows, la versión más actual (a Mayo 2022) de ELK 8.2.0. La versión de JDK utilizado es la 18.

Los archivos deberán ser descomprimidos dentro del equipo local, en adelante las rutas serán definidas de la siguiente forma:

* + <RUTA\_ELASTIC>: Ruta donde se aloja el aplicativo ElasticSearch
  + <RUTA\_LOGSTASH>: Ruta donde se aloja LogStash
  + <RUTA\_KIBANA>: Ruta donde se aloja Kibana

Para LogStash, al tratarse de una herramienta que se empleó para la carga de los archivos .csv de declaraciones agregadas, no existe configuración adicional, salvo la creación de archivos .conf los cuales serán invocados para la ingesta de información que se explica en el capítulo de carga de información histórica.

**ElasticSearch**

* Ejecutar a través de una línea de comandos el archivo elasticsearch.bat de la ruta <RUTA\_ELASTIC>\bin , el cual ejecuta el servicio de ElasticSearch; en su primera ejecución se encargará de aplicar las configuraciones por defecto de la herramienta. La Figura 48 despliega información del usuario por defecto elastic y la contraseña; así como el token de enrolamiento. Estos datos deberán ser tomados en cuenta a la hora de instalar Kibana.

Figura 48 Ejecución Inicial de ElasticSearch

Texto

Descripción generada automáticamente

* Al entrar a un navegador web a la dirección localhost:9200, tal cual se indica en la Figura 49, se desplegará una respuesta de parte de ElasticSearch con información sobre el nombre del equipo y versión ElasticSearch.

Figura 49 Página Inicial ElasticSeach

Texto

Descripción generada automáticamente

**Kibana**

* Ejecutar a través de una línea de comandos el archivo kibana.bat de la ruta <RUTA\_KIBANA>\bin para iniciar el servicio de Kibana. La Figura 50 describe la ejecución del servicio de kibana, como se observa se indica la dirección a donde se accederá a kibana.

Figura 50 Inicio de Servicio de Kibana

Texto

Descripción generada automáticamente

* Dar clic sobre la URL desplegada en la línea de comandos, se desplegará la página de inicio de instalación de kibana. Tal como se visualiza en la Figura 51, Kibana solicitará ingresar el token de enrolamiento, aquí se deberá insertar el token desplegado al instalar Elastic, a continuación, dar clic en **Configure Elastic**.

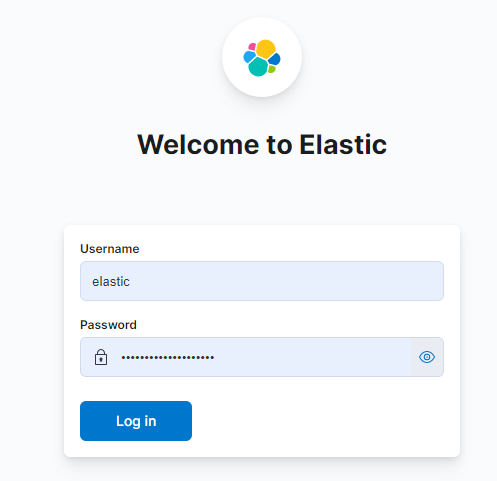
Figura 51 Configuración Kibana- Token de enrolamiento

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

* La Figura 52, corresponde a la pantalla que Kibana despliega para ingresar el usuario y la contraseña, estos datos corresponden a las credenciales generadas en la instalación de ElasticSearch.

Figura 52 Página de inicio de autenticación de Elastic



* Una vez ingresado las credenciales, tal como se describe en la Figura 53, se desplegará la página de Inicio de Kibana. Con ello se podrá crear los índices (tablas), visualizaciones y cuadros de mando.

Figura 53 Página de Inicio Kibana

Interfaz de usuario gráfica, Texto, Aplicación, Sitio web

Descripción generada automáticamente

**Deshabilitar HTTPS**

Hasta este punto se tiene implementado ElasticSearch y Kibana con conexión HTTPS; sin embargo, el certificado generado no es confiable, lo que no permitirá el acceso desde Apache Spark a menos de que se cuente con un certificado avalado por una autoridad de certificación. Para ambientes con Apache Spark en los cuales no se requiera de mecanismos de seguridad, se requiere deshabilitar el protocolo HTTPS, a fin de que las solicitudes de acceso fluyan únicamente por HTTP.

* Modificar el archivo elasticsearch.yml dentro de la ruta <RUTA\_ELASTIC>\config realizando los cambios sobre las siguientes etiquetas, las mismas que se encuentran inicialmente con valor en *True.*  Esto permitirá deshabilitar el protocolo https y acceder desde Kibana y Spark mediante http.

xpack.security.enabled: **false**

xpack.security.enrollment.enabled: **false**

xpack.security.http.ssl:

enabled: **false**

xpack.security.transport.ssl:

enabled: **false**

* Abrir el archivo kibana.yml dentro de la ruta <RUTA\_KIBANA>\config, dentro de las etiquetas **elasticsearch.hosts** y **xpack.fleet.outputs**  y modificar la dirección donde se encuentra el servidor de ElasticSearch con el puerto http.

elasticsearch.hosts: ['**http**://192.168.1.52:9200']

xpack.fleet.outputs: [{id: fleet-default-output, name: default, is\_default: true, is\_default\_monitoring: true, type: elasticsearch, hosts: ['**http**://192.168.1.52:9200'], ca\_trusted\_fingerprint: 615bebd8495982e091ae9c3c5f43c17feca2ac101444d0987bc657e777bd9b11}]

* Con estos cambios efectuados, se deberá iniciar nuevamente los servicios de ElasticSearch y Kibana.

## Anexo III. Instalación y configuración de Apache Spark

Para la instalación de Apache Spark descargar el archivo comprimido desde el sitio <https://spark.apache.org/downloads.html>; para que exista conectividad y compatibilidad con las versiones de Java, Python, Kafka y ElasticSearch se deberá obtener la versión Spark 2.4.0. Se optó por el Sistema Operativo Windows.

**Spark**

* Descomprimir el archivo resultante en un directorio local, en adelante la ruta será definida con el valor <RUTA\_SPARK>.
* Crear la variable de entorno SPARK\_HOME con el valor <RUTA\_SPARK> y añadir dentro de la variable path el valor <RUTA\_SPARK>/bin
* Crear el directorio HADOOP/bin y asociarlo a una variable de entorno <HADOOP\_HOME>.
* Dentro del directorio HADOOP/bin copiar los archivos winutil.exe y hadoop.dll, los mismos pueden ser obtenidos del sitio público <https://github.com/steveloughran/winutils/tree/master/hadoop-2.8.1>. Este paso es imperativo para que se permita correr el servicio de Spark sobre Windows.

**Anaconda, Python , JDK y FindSpark**

* Descargar e instalar Anaconda para el sistema Operativo Windows del sitio <https://www.anaconda.com/products/distribution>.
* Una vez que se ha instalado Anaconda, debido a que la más reciente versión (a Mayo del 2022) corre sobre Python 3.9, se deberá degradar el mismo hacia Python 3.7; este paso es necesario ya que la versión de Spark descargada se ejecuta correctamente sobre Python 3.7. Para ello dentro de una línea de comandos (CMD.exe Prompt) lanzada desde Anaconda se ejecuta el comando:

conda install python=3.7

* Crear la variable de entorno PYTHONPATH para poder acceder a la clase java py4j. El valor de la variable corresponde al valor %SPARK\_HOME%/python;%SPARK\_HOME%/python/lib/py4j-**X.Y.Z**-src.zip;%PYTHONPATH% . La Figura 54 permite visualizar la versión de py4j dentro del directorio <SPARK\_HOME>/Python/lib, para esta instalación se tiene la versión 0.10.7, por lo que ese valor deberá ser descrito dentro de la variable de entorno (**X.Y.Z**).

Figura 54 Versión py4j dentro de SPARK\_HOME

Texto

Descripción generada automáticamente

* De forma similar, la versión de Spark 2.4.0 soporta JDK 8, para ello dentro de una línea de comandos lanzada desde Anaconda se ejecuta el comando:

conda install openjdk=8

* Finalmente, para que cualquier *notebook* de Jupyter localice a nuestro servicio de Spark (pyspark) alojado en nuestro equipo local se instalará el paquete de python findspark, a través del comando

pip install findspark

## Anexo IV. Instalación de Geopandas en Anaconda-Jupyter

Para la instalación de Geopandas en Jupyter se ejecutará las siguientes actividades:

* Crear un entorno de desarrollo en una terminal lanzada en Anaconda con el comando:

conda create -n nombre\_entorno\_desarrollo

* Activar el entorno de desarrollo:

conda activate nombre\_entorno\_desarrollo

conda config --env --add channels conda-forge

conda config --env --set channel\_priority strict

* Instalar geopandas mediante el comando:

conda install geopandas

* Añadir el entorno de desarrollo Python recientemente creado a Jupyter, a través de la instrucción:

python -m ipykernel install --name geo\_env

## Anexo V. Análisis de Componentes Principales

El Análisis de Componentes Principales es una técnica estadística empleada para la reducción de dimensiones, la Figura 55, demuestra como a través de PCA, un conjunto de datos definidos en primera instancia con tres variables (3D) logran representarse mediante un plano bidimensional con el objetivo clave que es facilitar su análisis y visualización, así como también mejorar el rendimiento de un algoritmo de *Machine Learning*.

Figura 55 Reducción de dimensionalidad con PCA

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza media

**Fuente:** https://365datascience.com/

A través de esta técnica, se busca que, con una menor dimensionalidad, se represente la más importante información. Las variables resultantes, mejor conocidas como componentes principales, en sí no son una copia de las variables de entradas, sino que son un resumen de estas y cómo se relacionan.

La obtención de los componentes principales viene dada por los pasos descritos a continuación en el contexto de utilizar un lenguaje de programación para su cálculo:

* Normalizar los datos, de esta forma todas las variables que pueden estar en diferentes unidades de medida son transformadas en un conjunto de datos con el mismo peso e importancia.
* Calcular todos los componentes, cuya cantidad es similar al total de variables, que contienen en forma ordenada la distribución de la varianza. La sumatoria de la varianza es 1. La Figura 56 describe la varianza acumulativa para cada componente de un total de siete. Como se observa, la varianza es mayor en los primeros componentes. Por ejemplo, con los tres primeros se tiene una varianza acumulativa del 80%, lo que equivale a tener con únicamente tres dimensiones el 80% de la información más representativa del *set*  de datos en lugar de emplear siete variables.

Figura 56 Varianza acumulada por componentes

Gráfico, Gráfico de líneas

Descripción generada automáticamente

**Fuente:** https://365datascience.com/

* Obtener el set de datos resultante en función de un cálculo de PCA para un determinado número de componentes. Con estos nuevos datos se podrá realizar cualquier tipo de análisis, en el caso de este TFM la aplicación de algoritmos de clusterización.

## Anexo VI. Repositorio

Las fuentes de datos utilizadas y código fuente implementado en este TFM se encuentran en el sitio web <https://github.com/byronodg/REPOSITORIO_TFM> , cuyo acceso es de tipo público. La Figura 57 representa una captura con el directorio principal del Repositorio.

Figura 57 Repositorio Github



La Tabla 14 detalla cada uno de los componentes junto a su descripción:

Tabla 14 Descripción de componentes repositorio Github

|  |  |  |
| --- | --- | --- |
| **Archivo/Carpeta** | **Subcomponentes** | **Descripción** |
| Arquitectura.PNG | N/A | Representa a la arquitectura de la implementación, con las direcciones IPs, tecnologías utilizadas, versiones, e/o. |
| INTERFAZ\_GRAFICA | Generaciones\_Declaraciones.form  Generaciones\_Declaraciones.java  Hilo\_provincia.java | Clases java con código fuente de la interfaz gráfica para la generación de declaraciones al detalle del año 2022 y el hilo de ejecución. |
| ORACLE | declaraciones.sql  geográfica.sql  genera\_declaraciones.sql | Definición de tablas declaraciones para almacenar los datos autogenerados, y la tabla geográfica que contiene la información de provincias y cantones del Ecuador. También se define el procedimiento genera\_declaraciones que es invocado por la interfaz java para la generación de datos aleatorios. |
| KAFKA | zookeeper.properties  server.properties  connect\_standalone.properties  fuente\_declaraciones.properties | Archivos de configuración del servidor de Zookeeper, Kafka, y definición de la conexión hacia la base de datos Oracle. |
| ELK | Elasticsearch.yml  Kibana.yml | Archivos de configuración para los servidores de Elastic y Kibana |
| carga\_2020.conf  carga\_2020.conf  sri\_ventas\_2020\_1.csv  sri\_ventas\_2021\_2.csv | Archivos de configuración LogStash para generación de procesos de carga de archivos .csv con información de declaraciones 2020-2021 |
| Índices\_declaraciones.txt | Archivo con definición de las tablas (índices) de Elastic: declaraciones para datos del periodo 2020 y declaraciones\_2022 para data autogenerada del año 2022 |
| [dashboard\_declaraciones\_historicas.ndjson](https://github.com/byronodg/REPOSITORIO_TFM/blob/main/ELK/dashboard_declaraciones_historicas.ndjson)  [dashboard\_declaraciones\_2022.ndjson](https://github.com/byronodg/REPOSITORIO_TFM/blob/main/ELK/dashboard_declaraciones_2022.ndjson) | Archivos con definición de reportes en Kibana sobre declaraciones históricas (2020-2021) y cuadro de mando en tiempo real de la información del año 2022 |
| SPARK | FINAL\_TFM\_DECLARACIONES\_2.py | Código Python Pyspark de la integración KAFKA-SPARK-ELASTIC |
| CLUSTERIZACION | [cantones.geojson](https://github.com/byronodg/REPOSITORIO_TFM/blob/main/CLUSTERIZACION/cantones.geojson)  [poblacion\_ecu\_2020.csv](https://github.com/byronodg/REPOSITORIO_TFM/blob/main/CLUSTERIZACION/poblacion_ecu_2020.csv)  [sri\_ventas\_2020L.csv](https://github.com/byronodg/REPOSITORIO_TFM/blob/main/CLUSTERIZACION/sri_ventas_2020L.csv)  [sri\_ventas\_2021L.csv](https://github.com/byronodg/REPOSITORIO_TFM/blob/main/CLUSTERIZACION/sri_ventas_2020L.csv) | Información utilizada para construir los modelos de clusterización. Archivos con data de declaraciones 2020-2021, población del Ecuador al 2020 y archivo .geoson para la gráfica del mapa del Ecuador con Geopandas |
| [clusterizacion.ipynb](https://github.com/byronodg/REPOSITORIO_TFM/blob/main/CLUSTERIZACION/clusterizacion.ipynb) | Código Python sobre Jupyter con la implementación de los modelos K-Means y Jerárquico Aglomerativo. |

Adicionalmente se tiene cargado un video en YouTube donde se realiza una demostración del desarrollo implementado, con la ruta [TFM: Demostración Procesamiento Declaraciones 2022 - YouTube](https://www.youtube.com/watch?v=fPw1PyqrleM)

En este video se hace la integración de la interfaz gráfica en Java para la generación de declaraciones a detalle 2022 y su almacenamiento de Oracle, su captura a través de un conector de Kafka, el procesamiento mediante Apache Spark y la interacción con la base de datos de ElasticSearch, que realizará operaciones *upsert* de los datos ya agregados. Finalmente se dispondrá de un *dashboard*  cuyo refrescamiento periódico permitirá la visualización de los datos.