UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO
FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA
ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS

TESIS

Preprocesamiento de datos mediante herramientas de Big Data

PRESENTADO POR:
Br. CADY INDIRA LOAYZA BLANCO

PARA OPTAR AL TITULO PROFESIONAL DE INGENIERO INFORMÁTICO Y DE SISTEMAS

ASESOR:
M.Sc. YESHICA ISELA ORMEÑO AYALA

Financiado por:
CONCYTEC - UNSAAC - FONDECYT

CUSCO - PERÚ
2023

INFORME DE ORIGINALIDAD

(Aprobado por Resolución Nro.CU-303-2020-UNSAAC)

El que suscribe, **Asesor** del trabajo de investigación/tesis titulada: Preprocesamiento de datos mediante herramientas de Big Data

presentado por: Cady Indira Leayza Blanco con DNI Nro.: 43120865

presentado por: ..... con DNI Nro.: .....

para optar el título profesional/grado académico de Ingeniero Informático y de Sistemas

Informo que el trabajo de investigación ha sido sometido a revisión por 2 veces, mediante el Software Antiplagio, conforme al Art. 6° del ***Reglamento para Uso de Sistema Antiplagio de la UNSAAC*** y de la evaluación de originalidad se tiene un porcentaje de 9 %.

Evaluación y acciones del reporte de coincidencia para trabajos de investigación conducentes a grado académico o título profesional, tesis

Porcentaje	Evaluación y Acciones	Marque con una (X)
Del 1 al 10%	No se considera plagio.	X
Del 11 al 30 %	Devolver al usuario para las correcciones.	
Mayor a 31%	El responsable de la revisión del documento emite un informe al inmediato jerárquico, quien a su vez eleva el informe a la autoridad académica para que tome las acciones correspondientes. Sin perjuicio de las sanciones administrativas que correspondan de acuerdo a Ley.	

Por tanto, en mi condición de asesor, firmo el presente informe en señal de conformidad y **adjunto** la primera página del reporte del Sistema Antiplagio.

Cusco, 05 de octubre de 2023

.....

Firma

Post firma Jessica Isela Ormeño Ayala

Nro. de DNI 25002834

ORCID del Asesor 0000-0002-5497-6928

Se adjunta:

1. Reporte generado por el Sistema Antiplagio.
2. Enlace del Reporte Generado por el Sistema Antiplagio: https://unsaac.turnitin.com/viewer/submissions/oid:27259:272957580?locale=es-MX

turnitin

Identificación de reporte de similitud: oid:27259:272957580

NOMBRE DEL TRABAJO
tesis_Indira final051023.pdf

AUTOR
Cady Indira Loayza Blanco

RECUENTO DE PALABRAS
17016 Words

RECUENTO DE CARACTERES
94554 Characters

RECUENTO DE PÁGINAS
84 Pages

TAMAÑO DEL ARCHIVO
2.3MB

FECHA DE ENTREGA
Oct 5, 2023 11:38 AM GMT-5

FECHA DEL INFORME
Oct 5, 2023 11:39 AM GMT-5

9% de similitud general

El total combinado de todas las coincidencias, incluidas las fuentes superpuestas, para cada base c

- 9% Base de datos de Internet
- Base de datos de Crossref
- 1% Base de datos de publicaciones
- Base de datos de contenido publicado de Crossr

Excluir del Reporte de Similitud

- Base de datos de trabajos entregados
- Material citado
- Coincidencia baja (menos de 12 palabras)
- Material bibliográfico
- Material citado

Yesica Ormeño Ayala
DNI 25002834

Resumen

Agradecimientos

Este trabajo está dedicado a mi madre Flor, por su apoyo incondicional y la motivación que me brindo para mejorar cada vez más. A mis hermanas María y Danae por ser cómplices de muchas experiencias. A mis amigos porque han estado a mi lado apoyándome y compartiendo momentos de aprendizaje.

Agradezco a la Universidad Nacional de San Antonio Abad del Cusco y la escuela profesional de Ingeniería Informática y de Sistemas donde me formé para llegar a ser un buen profesional, además de los docentes que me guiaron en este ámbito profesional.

Agradezco de manera especial a mi asesora la profesora Yeshica Isela Ormeño Ayala y a Hans Harley Ccacyahuillca Bejar por su apoyo, recomendaciones y sugerencias durante el proceso de investigación y desarrollo de mi proyecto de tesis.

Agradezco también al Consejo Nacional de Ciencia, Tecnología e Innovación Tecnológica (CONCYTEC) y al Fondo Nacional de Desarrollo Científico, Tecnológico y de Innovación Tecnológica (FONDECYT), que han financiado el desarrollo de esta tesis.

Resumen

El preprocesamiento de datos en entornos de *Big Data* es una etapa crucial para garantizar la calidad y la utilidad de los datos antes de que sean utilizados en análisis o aplicaciones. En este proceso existen desafíos por resolver, como por ejemplo, campos sin formato, fechas con diferentes formatos, valores nulos, ruido, identificación de características relevantes. Superar estos problemas es fundamental para aprovechar al máximo el potencial de los datos y lograr análisis precisos y significativos.

El objetivo principal de este trabajo es obtener conjuntos de datos limpios, libre de ruido que puedan considerarse correctos y útiles para el procesamiento de datos. Dado un dataset y eligiendo una herramienta como Apache Spark para el preprocesamiento de Big Data en un caso de uso, mediante los algoritmos existentes en esta librería se procede a limpiar, transformar, seleccionar características, manejo de valores atípicos, manejo de valores faltantes, normalización y estandarización, conversión de tipos de datos, reducción de ruido, muestreo de datos, para finalmente obtener como resultado datos preprocesados. También fueron aplicadas pruebas unitarias e integrales para cuantificar la calidad de datos de forma automática, preprocesando los datos en cada etapa.

Se realizó casos de uso con datasets; COVID-19, sismos y diabetes, para demostrar la generación de datos de limpios mediante técnicas de preprocesamiento específicamente en Apache Spark. Como resultado ilustramos tareas de análisis y visualización.

Palabras Claves: *Big Data, Apache Spark, Preprocesamiento*

Abstract

Data preprocessing in Big Data environments is a crucial stage to ensure the quality and usefulness of data before it is used in analytics or applications. In this process there are challenges to solve, such as raw fields, dates with different formats, null values, noise, identification of relevant characteristics. Overcoming these issues is critical to fully harnessing the potential of data and achieving accurate and meaningful analytics.

The main objective of this work is to obtain Smart Data, which can be considered correct and useful for data processing. Given a dataset and choosing a tool like Apache Spark for Big Data preprocessing in a use case, using the existing algorithms in this library we proceed to clean, transform, select features, handle outliers, handle missing values, normalization and standardization, data type conversion, noise reduction, data sampling, to finally obtain Smart Data as a result. Unit and integral tests were also applied to quantify data quality automatically, preprocessing the data at each stage.

Use cases were carried out with the COVID-19, earthquake and diabetes dataset, to demonstrate the generation of Smart Data through preprocessing techniques. As a result we illustrate analysis and visualization tasks.

Keywords: *Big Data, Smart Data, Apache Spark, Pre-processing*

Índice

Índice de Figuras

Abreviaturas

ETL Extraccion, carga y transformación

FS Selección de características

IS Selección de instancias

IG Generación de instancias

API Application Programming Interface

DQR Data Quality Rules

PCA Pricipal Component Analysis

RDD Conjuntos de datos distribuidos resistentes

DM Data mining

DStream Data Stream

HDFS Hadoop Distributed File System

Capítulo I

Generalidades

1 Aspectos Generales

Las generalidades del proyecto de investigación son presentadas en el presente capítulo.

1.1 Problema de Investigación

1.1.1 Descripción del Problema

Las organizaciones y las personas generan grandes cantidades de datos a un ritmo muy elevado que llegan a una escala de exabyte (10^{18} Bytes), se pueden obtener nuevos conocimientos a partir de sus contenidos. Este último ayudará a las instituciones a obtener conocimientos más valiosos y mejorar su posición competitiva algunas de las aplicaciones se ven en Facebook, Instagram, Netflix, AliExpress y otros. Se sabe que Big Data representa grandes cantidades de datos que son casi imposibles de administrar y procesar utilizando herramientas tradicionales de administración de datos. Es decir grandes conjuntos de datos que requiere plataformas computacionales complejas para ser analizadas como información. Además originalmente los datos son incompletos lo que origina ciertas discrepancias e inconsistencias. Estas anomalías son causadas por algunos factores humanos o de transmisión, así como valores perdidos, campos con datos incoherentes, datos superfluos o un tamaño demasiado grande(número de atributos e instancias). Así como tal, estos datos no son aptos para ser procesados, se requiere realizar un tratamiento previo para obtener datos limpios, libres de ruido.

La propuesta de la presente tesis es obtener datos limpios con el uso de las técnicas de preprocesamiento por ejemplo, normalización de datos, imputación de valores perdidos, selección de atributos, selección de instancias, discretización, extracción, carga y tratamiento de datos (ETL). Pretendemos hacer un estudio de las técnicas de preprocesamiento con ejemplos en casos de uso y aplicados en datasets, para verificar la limpieza de los datos.

1.1.2 Formulación del Problema

- ¿Es posible obtener datos limpios mediante una herramienta que use técnicas de preprocesamiento de Big Data?

1.2 Antecedentes

El procesamiento de datos en big data, se ha contribuido en una variedad de experimentos donde se pretenden demostrar, analizar y mejorar el preprocesamiento

de datos y su rendimiento respecto al estado de arte. A continuación se presenta un conjunto de trabajos relacionados que describen algunos de estos casos:

1. En el trabajo de (García-Gil. et al., 2019) se utilizan las herramientas de pre-procesamiento de Big Data, tal que a partir de datos sin procesar se obtuvo Smart Data. Donde fueron utilizadas dos librerías BigDaPSpark (Apache Spark) e BigDaPLink (Apache Flink), ambas librerías contienen una serie de algoritmos muy populares y ampliamente utilizados, para el filtro de ruido, la discretización o la reducción de datos entre otros. Fueron seleccionados algoritmos de cada biblioteca para el filtrado de ruido y para la discretización del flujo de datos. Para el filtrado de ruido eligieron un conjunto de datos con 5×10^6 instancias y 18 atributos, y para la discretización utilizaron un conjunto de datos con 9.29×10^5 instancias y 11 atributos, SUSY dataset se encuentra en el repositorio UCI, en este trabajo se tiene un desafío que es la combinación y disposicion de los métodos.Con estos algoritmos lograron obtener Smart Data a partir de Big Data sin procesar.

2. Por otro lado, en el trabajo desarrollado por (Taleb and Serhani, 2017) la calidad de datos es considerada como un elemento clave en Big Data durante la fase de procesamiento en esta etapa, los datos de baja calidad no son introducidos en la cadena de valor de Big Data. De esta forma se abordan reglas para la obtención de datos de calidad (DQR - Data Quality Rules) después de la evaluación de calidad y antes del pre-procesamiento de Big Data. El trabajo propone un modelo para descubrir reglas (DQR) que mejoren y establezcan con precisión las actividades de preprocesamiento basadas en requisitos de calidad. Para lo cual definen un conjunto de actividades de pre-procesamiento para automatizar la generación de reglas. Los experimentos realizados mostraron un aumento de calidad en los puntajes tras aplicar las reglas descubiertas y optimizados en los datos.

3. El trabajo de (García et al., 2017) tiene como objetivo estudiar la creciente importancia del pre-procesamiento de datos en Big Data, así como enumerar y describir las tecnologías y herramientas de analítica de datos y las técnicas existentes. Para ello, el trabajo se organiza como sigue. Se describe en qué consiste el preprocesamiento de datos y las dos áreas en las que podemos clasificar las técnicas de preprocesamiento. Además se analiza las tecnologías y herramientas para el procesamiento de datos masivos. Se estudia las propuestas escalables para el procesamiento de datos masivos y presentan un caso de uso para selección de atributos. Este trabajo tiene como reto diseñar nuevos algoritmos de

preprocesamiento de datos masivos.

4. En este otro trabajo de (García-Gil et al., 2019), enfoca el filtrado de ruido para Big Data, siendo este un problema común que afecta la calidad de los datos, particularmente en los problemas de clasificación, donde el ruido de la etiqueta se refiere al etiquetado incorrecto de las instancias de entrenamiento, considerandose una característica muy perjudicial de los datos. Sin embargo, en esta era de Big Data, el crecimiento masivo en la escala de los datos plantea un desafío a las propuestas tradicionales creadas para abordar el ruido, ya que tienen dificultades para hacer frente a una cantidad de datos tan grande. Se deben proponer nuevos algoritmos para abordar el ruido en los problemas de Big Data, proporcionando datos limpios y de alta calidad, también conocidos como Smart Data. En este trabajo, se proponen dos enfoques de preprocesamiento de Big Data para eliminar ejemplos ruidosos: un conjunto homogéneo y un filtro de conjunto heterogéneo, con especial énfasis en su escalabilidad y rasgos de rendimiento. Los resultados obtenidos muestran que estas propuestas permiten obtener eficientemente un conjunto de datos inteligente de cualquier problema de clasificación con Big Data.

5. Esta tesis (Olarte C. and Casaverde L., 2020) trata del análisis de las opiniones de una gran cantidad de personas respecto a política, salud, educación, y eventos relevantes que conciernen a nuestro país, analizar la opinión de las personas es sumamente importa para la toma de decisiones y sobre todo para obtener de una forma más rápida un índice de aceptación con respecto a cierto tema pero con los métodos convencionales como encuestas o llamadas telefónicas consumen mucho tiempo y esfuerzo. Para obtener este tipo de información de forma fácil, accesible, automática y en tiempo real para ello se uso Aprendizaje Automático con esta herramienta se puede predecir los sentimientos ya sea positivo o negativo y resolver los problemas de impacto social en la toma de decisiones en las organizaciones públicas o privadas, también se aplica para medios publicitarios como ver la aceptación de un producto en el mercado.

1.3 Justificación

El preprocesamiento de datos es muy útil para la tarea de extracción de conocimiento sin embargo son latentes las dificultades presentes en ciertos tipos de datos los cuales requieren una exploración más rigurosa, considerando las cantidades excesivas de datos inconsistentes o valores nulos hacen necesaria la identificación y aplicación de técnicas de preprocesamiento de conjuntos de datos no estructurados (*datasets*) y de esta forma evitar una manipulación indiscriminada y precaria.

El uso de técnicas robustas de preprocesamiento para el análisis de los datos nos permite obtener resultados de mucho valor para la toma de decisiones y en tiempo real, un enfoque real en nuestra región seria por ejemplo en el ámbito de proyectos de investigación ya que existen *datasets* en las diferentes áreas de estudio donde es posible su aplicación, lo cual generaría *software* que ayude a obtener resultados científicos de mayor impacto.

Es importante fomentar la investigación en esta área, en este sentido el presente trabajo pretende contribuir con distintas instituciones y/o entidades estableciendo técnicas de preprocesamiento para diferentes datasets en el entorno de Big Data con el fin de obtener datos de calidad.

1.4 Objetivos

1.4.1 Objetivo General

Obtener datos limpios, libre de ruido a través de técnicas de preprocesamiento, en tres datasets, mediante el uso de herramientas de Big Data.

1.4.2 Objetivos Específicos

Dentro de los objetivos específicos se incluyen:

- Seleccionar datasets que no hayan sido preprocesados.
- Explorar o analizar las datasets.
- Aplicar técnicas de preprocesamiento en diferentes casos de estudio.
- Validar la propuesta de mejora de calidad de datos con el uso del test unitario integral.

1.5 Alcances y Limitaciones

1.5.1 Alcances

- Para efectos de validación del presente proyecto fueron utilizadas herramientas de gestión de Big Data (Apache Spark).
- Debido a que el trabajo de investigación se enfocará en la etapa preprocesamiento de datos, no se realizará el análisis de aprendizaje automático (Machine Learning), sin embargo su desempeño es consecuencia de la calidad de datos.
- Se utilizará dataset de COVID-19, sismos y diabetes para demostrar las técnicas de preprocesamiento.

- Para la verificación de la calidad de datos se utilizará pruebas unitarias e integrales, y en cada etapa se realizará el preprocesamiento.
  - En un primer momento se configurarán las pruebas unitarias e integrales hasta completar el preprocesamiento, futuramente el dataset es verificado automáticamente, por ejemplo, en el caso de un dataset que varia con el tiempo, como el caso de COVID-19.
- Se estudiaron, ejemplificaron y aplicaron varias técnicas de preprocesamiento, como por ejemplo, Extracción, Carga y Transformación de datos (ETL), técnicas de transformación para resolver el problema de categorización, normalización, estandarización y filtrado de datos.

1.5.2 Limitaciones

En el desarrollo de la investigación se presentan las siguientes limitaciones.

- No se hará comparaciones de rendimiento entre las técnicas de preprocesamiento.
- Únicamente se utilizará la herramienta Apache Spark para el preprocesamiento de datos.
- No se utilizarán datos de *streaming*.
- Solo se utilizará *google colab* para la realización de casos de uso.

1.6 Metodología

1.6.1 Tipo de Investigación

El tipo de investigación realizada es aplicada y experimental, porque se utilizará un método para medir la calidad de datos en diferentes casos de uso.

1.6.2 Diseño Metodológico

La metodología a seguir para el desarrollo de la presente investigación está sustentada en las siguientes fases (Figura 1.1):

- **Primero** En la primera fase se hace la revisión bibliográfica de artículos científicos relacionado a herramientas para preprocesamiento de datos en big data y se obtiene las datasets, estos datasets se descargan de un repositorio libre, como mínimo se tendrá dos datasets para el preprocesamiento.
- **Segundo** En la segunda fase se tiene la dataset, con el uso de las librerías de Apache Spark se hace la lectura del contenido de los datasets.

- **Tercero** En esta fase se realiza la limpieza de los datos aplicando las diferentes técnicas de preprocesamiento que se tiene.
- **Cuarto** En esta última fase se hace uso del test unitario integral para verificar la calidad de los datos, en caso de estar libre de ruido pasan al proceso de análisis de datos, caso contrario vuelven al preprocesamiento de datos este proceso se repite hasta que los datos queden limpios.
- **Quinto** En esta última fase se realiza el análisis de datos, esto quiere decir que se obtendrá datos limpios en formato csv o json.

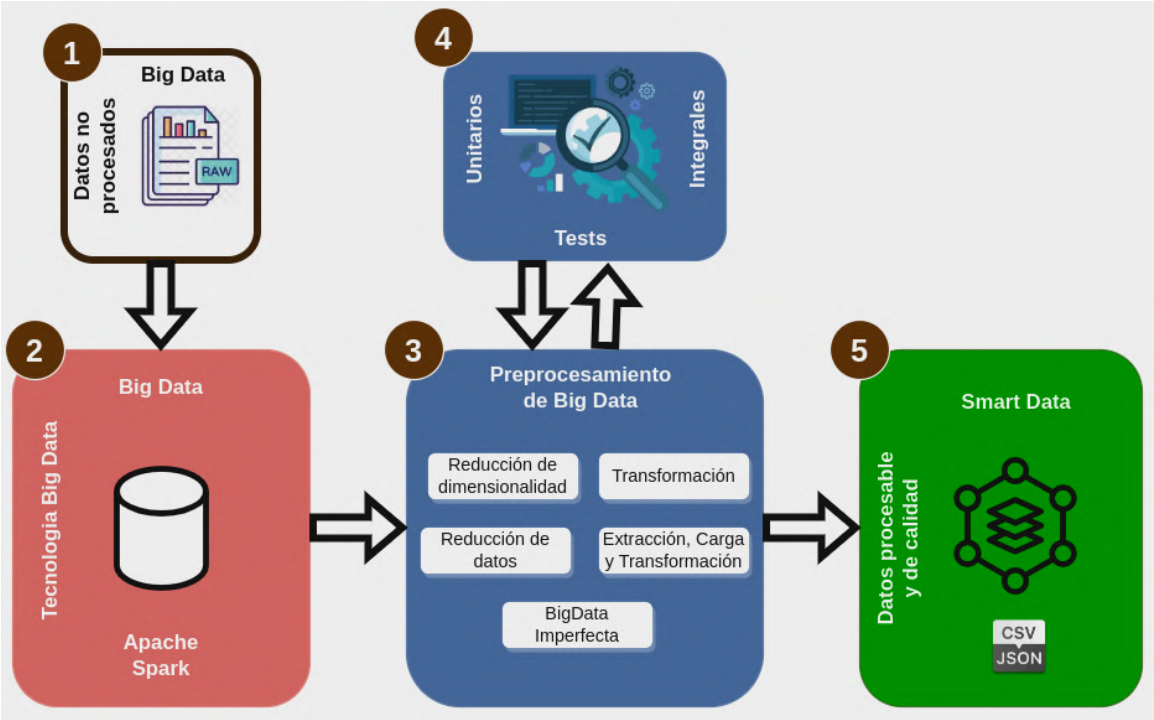


Figura 1.1: *Modelo a seguir: Flujo de preprocesamiento de Big Data para la obtención de datos de calidad (*Smart Data*).*
Fuente: *Propia*

1.7 Contribuciones

Las contribuciones de este trabajo son las siguientes:

- Preprocesamiento de los datos para la obtención de datos limpios en una dataset de COVID-19 para la región de Cusco.
- Aplicar en diferentes casos de uso donde se muestre los procesos que se siguen en el preprocesamiento y la obtención de 3 datasets libres de ruido.

1.8 Cronograma de actividades

El cronograma de actividades se muestra en la figura:

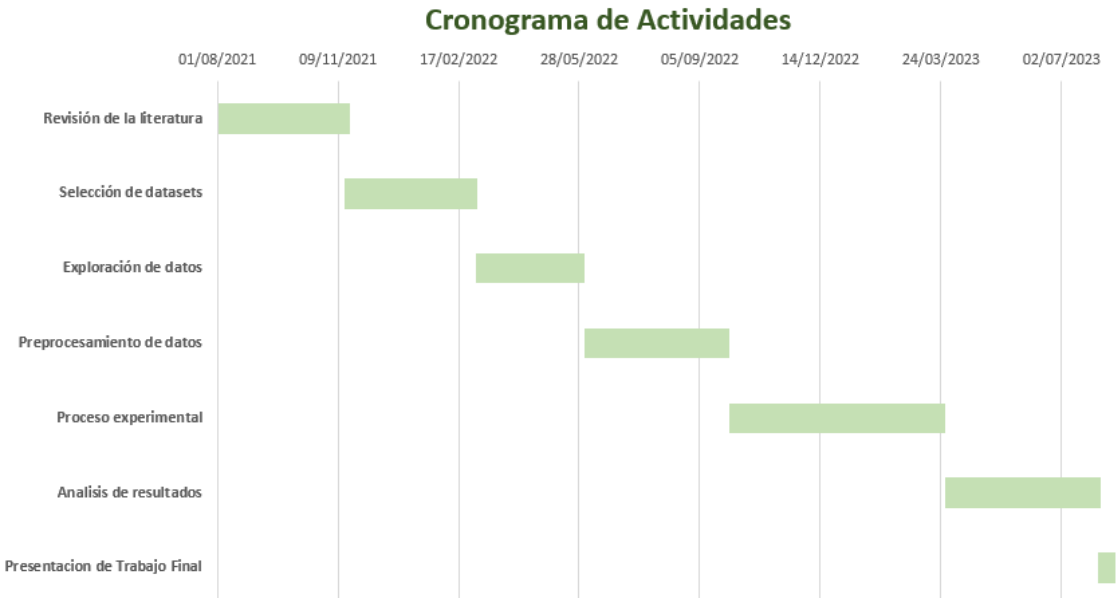


Figura 1.2: *Cronograma de actividades*
Fuente: *Propia*

Capítulo II

Marco Teórico

2 Marco Teórico

2.1 Big Data

2.1.1 Tipos de datos para Big Data

- Datos de sensor / generados por la máquina: incorpora Registros de detalles de llamadas (CDR), weblogs, medidores agudos, fabricación de sensores, registros de engranajes, intercambio de marcos información (Schell, 2013).
- Información social: incorpora corrientes de críticas del cliente publicadas por la población en general todo el mundo, sitios de blogs a pequeña escala como Twitter, redes sociales en línea como Facebook.
- Datos de negociación de acciones: la información de negociación de acciones contiene datos sobre **compra** y **dar** opciones sobre activos comunes, las ofertas de los clientes se supervisan desde el organizaciones.
- Gridding Data: La información de la matriz de potencia contiene datos devorados por un hub como para una estación base
- Datos de transporte: la información de transporte incorpora mostrar, limitar, separar y accesibilidad de un vehículo.
- Buscar datos del motor web: los motores de búsqueda recuperan gran cantidad de información (Michael and Miller, 2013).

2.1.2 Características de Big Data

- Volumen: Volumen alude a la medida de información. La información creada por la máquina se entrega en cantidades significativamente mayores que la información no convencional (Ji et al., 2012). Con más de 25.000 vuelos de aerolíneas por día, el volumen diario de esta única fuente de información sigue llegando a los petabytes. Medidores inteligentes y hardware mecánico sustancial como refinerías de petróleo y aparatos de penetración crean volúmenes de información comparativa, intensificando el problema.
- Velocidad: es la velocidad de preparación de la información. El ritmo al que la información se transmite desde fuentes, por ejemplo, teléfonos móviles, flujos de clics, alta recurrencia El intercambio de existencias y las formas de máquina a máquina es enorme y consistentemente rápido en movimiento (Muhtaroğlu et al., 2013).

- Variedad: alude al tipo de datos. La gran información llega más allá de lo organizado información, por ejemplo, números, fechas y cadenas para incorporar datos no estructurados información; video, sonido, transmisiones de clics, información 3D y registros de registro (Ji et al., 2012).

- Valor: La estimación monetaria de diversa información difiere fundamentalmente. Regularmente hay grandes datos cubiertos entre un conjunto más grande de información no convencional; la prueba es distinguir lo que es rentable y después que cambiar y separar esa información para su examen (Ji et al., 2012).

2.1.3 Tecnologías de Big Data

Big data es un término general que describe la acumulación de conjuntos de datos que no se pueden ser utilizados con técnicas de cálculo convencionales. Para poder manejar la enorme información se requieren diferentes dispositivos, sistemas y estructuras. Los grandes avances en la información son imperativos para procesar enormes volúmenes de información organizada y no estructurada en tiempo real y brindar una investigación más precisa, lo que puede impulsar a un liderazgo que genere beneficios transaccionales más notables, disminuciones de precios y menores peligros para el negocio. A continuación veremos los tipos de datos en big data con los que se tratan más.

2.1.4 Tipos Básicos de Big Data

- Datos estructurados: Los datos planificados son números y palabras que se pueden clasificar y analizar de forma eficaz. La información es producida por cosas como sensores del sistema implantados en dispositivos electrónicos, celdas avanzadas y dispositivos de marco de localización mundial (GPS). Los datos estructurados incorporan además cosas relacionadas con cifras, ajustes de cuenta e intercambio de información.

- Datos no estructurados: Los datos no estructurados incorporan más datos interesantes, por ejemplo, auditorías de clientes de sitios comerciales, fotografías y otros medios mixtos, y comentarios sobre lugares de comunicación informal de largo alcance. Este tipo de información no se puede aislar y convertir en clasificaciones o diseccionar numéricamente. El delicado desarrollo de Internet en los últimos tiempos implica que la variedad y la cantidad de información enorme siguen creciendo. Gran parte de ese desarrollo se origina en datos no estructurados. Hay muchos desafíos, de los cuales los principales son los siguientes:

  - Creación
  - Transferencia

- Presentación
- Lugar de almacenamiento
- Dar
- Captura de datos
- Buscando

2.2 Smart Data

Smart Data son datos de calidad, que están prestos para ser utilizados en la extracción de conocimiento y la toma de decisiones inteligente basada en datos. El preprocesamiento de datos es fundamental para convertir los datos almacenados en datos de calidad (Figura 2.1).
Hay 3 atributos claves que son necesarios para ser considerado *Smart Data* (García-Gil et al., 2019; Luengo et al., 2020a).

- Preciso: los datos deben ser lo que dicen que son con suficiente precisión para impulsar el valor. La calidad de los datos importa.
- Accionable: los datos deben impulsar una acción escalable inmediata de una manera que maximice un objetivo comercial como el alcance de los medios en todas las plataformas. La acción escalable es importante.
- Ágil: los datos deben estar disponibles en tiempo real y listos para adaptarse al entorno empresarial cambiante. La flexibilidad importa.

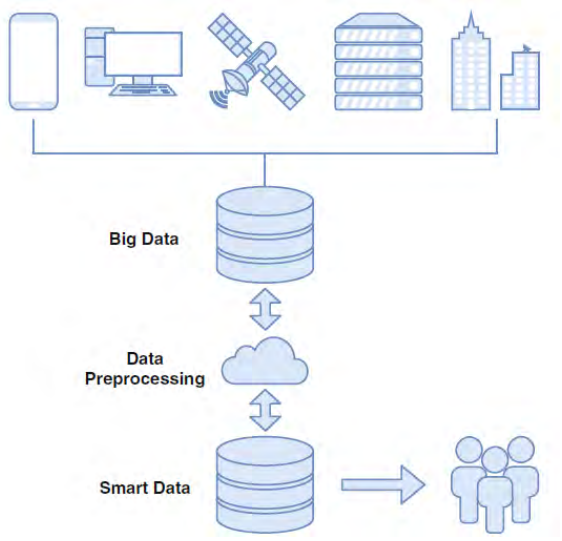


Figura 2.1: *Smart Data*
Fuente: *(Luengo et al., 2020a)*

2.3 Preprocesamiento de datos en la teoría de la información

Cuando los datos son imperfectos, contienen inconsistencias y tienen redundancias para ello es imperante realizar un preprocesamiento de datos con el objetivo de obtener datos de calidad. El preprocesamiento de datos provee al usuario una herramienta muy útil para tratar datos complejos que conlleva un largo periodo de tiempo para limpiar los datos. A continuación se presentan los principales aspectos y técnicas del preprocesamiento de datos (Luengo et al., 2020b).

2.3.1 Enfoque de reducción de datos

Las técnicas de reducción de datos surgen como algoritmos de preprocesamiento que tienen por objetivo simplificar y limpiar datos en bruto. Los métodos de reducción de datos se dividen en diferentes grupos dependiendo de la reducción, que son: Reducción de dimensionalidad, reducción de instancia y discretización.
(1) Desde el punto de vista de los atributos, el proceso más conocido de reducción de datos es el de selección de características (FS) y extracción de características. (2) Concerniente a la reducción de número de ejemplos, las técnicas de reducción de instancia pueden dividirse en métodos de selección de instancias (IS) y generación de instancias. (3) Por ultimo, cuando la reducción afecta los valores de los puntos de datos, nos referimos a técnicas de discretización (Figura 2.2).

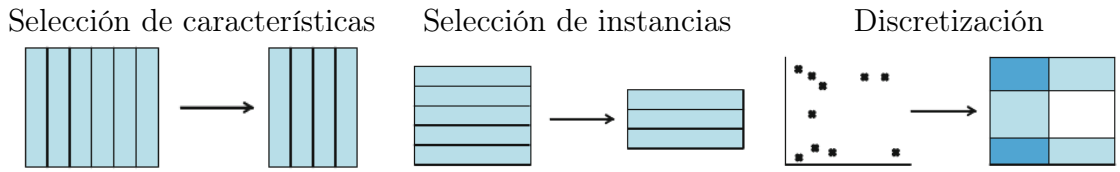


Figura 2.2: *Formas de reducción de datos*
Fuente: *(Luengo et al., 2020b)*

Reducción de dimensionalidad

Es cuando un *dataset* esta formado por un número grande de variables de predicción o número e instancias. A continuación se presentan los algoritmos de reducción de dimensionalidad más usados:

A) **Selección de características (FS):** El método de selección de características es el proceso de identificar y borrar lo más posible de información irrelavante y redundante Luengo et al.. Con el objeto de obtener un subconjunto de características que describa apropiadamente el problema original. FS elimina las características irrelevantes y redundantes que pueden inducir a correlaciones equivocadas en los algoritmos de aprendizaje de computacional, perjudicando su aspecto de generalización y evitando tambien el *overfitting*. FS tambien reduce el

espacio de busqueda de las características, lo que hace el proceso de aprendizaje computacional mucho más rapido y disminuye el consumo de memoria.
Modelos y visualizciones realizados con pocas características son faciles de entender e interpretar.

B) **Extracción de características:** FS no es el único método que lidia con la reducción de dimensionalidad. En vez de seleccionar características más resaltantes, las técnicas de extracción de características generan un nuevo conjunto de características combinando las originales deacuerdo a un criterio. Por ejemplo PCA (Principal Component Aalysis) Luengo et al., asi como tambien otros métodos lineales y no lineales.

C) **Reducción de instancia:** La reducción de instancia es el enfoque para minimizar el impacto de reducir un dataset muy grande. Se reduce el tamaño del *dataset* sin disminuir la calidad del conocimiento que puede ser extraido del mismo.
La reducción de instancia es un proceso complementario respecto de FS. Se reduce la cantidad de datos ya sea por eliminación de instancias o para generar otras nuevas.

D) **Selección de instancias (IS):** La selección de instancia es muy necesaria (Luengo et al., 2020b). El principal problema en IS es identificar ejemplo apropiados de una cantidad ingente de instancias y prepararlas para su procesamiento de análisis de datos. Entonces, IS esta conformada por una serie de técnicas que es capaz de escoger un subconjunto de datos que reemplace al *dataset* original y al mismo tiempo ser capaz de cumplir el objetivo de la aplicación.

E) **Generación de instancias (IG):** Los métodos IS concernientes a la identificación de un subconjunto óptimo de objetos representativos de los datos de entrenamiento (*data training*) original descartanto los ejemplos de ruido y redundancia. Los métodos e generación de instancias (IG) (Luengo et al., 2020b), por el contrario, en vez de seleccionar los datos, puede generar y reemplazar los datos originales con datos artificiales. Este proceso permite completar regiones en el dominio del problema, que no estan representados en los datos originales, o simplificar una cantidad ingente de instancias en menos ejemplos. Los métodos IG frecuentemente son llamados métodos de generación de prototipos (PG), los ejemplos artificiales tienden a comportarse como regiones o subconjuntos representativos de las instancias originales.
Los prototipos pueden generar diversos criterios. El enfoque más simple es volver a etiquetar algunos ejemplos, por ejemplo aquellos de los cuales se tiene sospecha de que pertenescan a una clase etiquetada erroneamente. Algunos métodos PG crean centroides juntando ejemplos similares, o mezclando el espacio de

características en varias regiones y despues crean un conjunto de prototipos para cada una.

2.3.2 Tratamiento de datos imperfectos

Es común que los datos en el mundo real presenten errores o estén incompletos (imperfectos). Por tal motivo es normal que se utilizen técnicas para limpiar el ruido en datos o atribuir datos faltantes (*missing values*). Las técnicas utilizadas son el filtro de ruido y la atribución a datos faltantes.

2.4 Apache Spark

Spark fue creado en 2009 como un proyecto dentro de AMPLab en la Universidad de California, Berkeley. Más específicamente, nació de la necesidad de probar el concepto de Mesos (Hindman et al., 2011), que también se creó en AMPLab. Spark se discutió por primera vez en el documento técnico de Mesos titulado "Mesos: una plataforma para el intercambio de recursos de grano fino en el centro de datos", escrito principalmente por Benjamin Hindman y Matei Zaharia.

Desde el principio, Spark se optimizó para ejecutar en memoria, lo que ayudó a procesar los datos mucho más rápidamente que los enfoques alternativos como MapReduce de Hadoop, que tiende a escribir datos desde y hacia los discos duros de las computadoras entre cada etapa de procesamiento. Sus defensores afirman que Spark que se ejecuta en la memoria puede ser 100 veces más rápido que Hadoop MapReduce, pero también 10 veces más rápido cuando se procesan datos basados en disco de manera similar a Hadoop MapReduce. Esta comparación no es del todo justa, sobre todo porque la velocidad bruta tiende a ser más importante para los casos de uso típicos de Spark que para el procesamiento por lotes, en el que las soluciones similares a MapReduce aún se destacan (Scott, 2015).

Spark se convirtió en un proyecto incubado de Apache Software Foundation en 2013 y, a principios de 2014, Apache Spark fue promovido para convertirse en uno de los proyectos de primer nivel de la Fundación. Spark es actualmente uno de los proyectos más activos gestionados por la Fundación, y la comunidad que ha crecido en torno al proyecto incluye tanto contribuyentes individuales prolíficos como patrocinadores corporativos bien financiados como Databricks, IBM y Huawei de China

Spark es un motor de procesamiento de datos de uso general, adecuado para su uso en una amplia gama de circunstancias. Las consultas interactivas en grandes conjuntos de datos (*datasets*), el procesamiento de transmisión de datos desde sensores o sistemas financieros y las tareas de aprendizaje automático tienden a asociarse con mayor frecuencia con Spark. Los desarrolladores también pueden usarlo para

respaldar otras tareas de procesamiento de datos, beneficiándose del amplio conjunto de bibliotecas y API para desarrolladores, y su soporte integral para lenguajes como Java, Python, R y Scala. Spark se utiliza a menudo junto con el módulo de almacenamiento de datos de Hadoop, HDFS, pero también puede integrarse igualmente bien con otros subsistemas de almacenamiento de datos populares como HBase, Cassandra, MapR-DB, MongoDB y S3 de Amazon. Hay muchas razones para elegir Spark, pero tres son clave:

- **Simplicidad:** se puede acceder a las capacidades de Spark a través de un conjunto de API enriquecidas, todas diseñadas específicamente para interactuar rápida y fácilmente con datos a escala. Estas API están bien documentadas y estructuradas de manera que sea sencillo para los científicos de datos y los desarrolladores de aplicaciones poner rápidamente a Spark en funcionamiento;

- **Velocidad:** Spark está diseñado para la velocidad, operando tanto en memoria como en disco. En 2014, Spark se utilizó para ganar el desafío de evaluación comparativa de Daytona Gray Sort, procesando 100 terabytes de datos almacenados en unidades de estado sólido en solo 23 minutos. El ganador anterior usó Hadoop y una configuración de clúster diferente, pero tomó 72 minutos. Esta victoria fue el resultado de procesar un conjunto de datos estáticos. El rendimiento de Spark puede ser incluso mayor cuando se admiten consultas interactivas de datos almacenados en la memoria, con afirmaciones de que Spark puede ser 100 veces más rápido que MapReduce de Hadoop en estas situaciones;

- **Soporte:** Spark admite una variedad de lenguajes de programación, incluidos Java, Python, R y Scala. Aunque a menudo está estrechamente asociado con el sistema de almacenamiento subyacente de Hadoop, HDFS, Spark incluye soporte nativo para una integración estrecha con una serie de soluciones de almacenamiento líderes en el ecosistema de Hadoop y más allá. Además, la comunidad de Apache Spark es grande, activa e internacional. Un conjunto creciente de proveedores comerciales, incluidos Databricks, IBM y todos los principales proveedores de Hadoop, brindan soporte integral para soluciones basadas en Spark.

2.4.1 Librerías

El componente final de Spark son sus bibliotecas, que se basan en su diseño como motor unificado para proporcionar una API unificada para tareas comunes de análisis de datos. Spark admite tanto bibliotecas estándar que se envían con el motor como una amplia gama de bibliotecas externas publicadas como de terceros paquetes de las comunidades de código abierto. Hoy en día, las bibliotecas estándar de Spark son en realidad la mayor parte del proyecto de código abierto: el motor central de Spark en sí ha cambiado poco desde que se lanzó por primera vez, pero las bibliotecas

han crecido para proporcionar más y más tipos de funcionalidad. Spark incluye bibliotecas para SQL y datos estructurados (Spark SQL), aprendizaje automático (MLlib), procesamiento de transmisión (Spark Streaming y la transmisión estructurada más nueva) y análisis de gráficos (GraphX). Más allá de estas bibliotecas, hay cientos de fuentes externas bibliotecas que van desde conectores para varios sistemas de almacenamiento hasta algoritmos de aprendizaje automático.

2.4.2 ¿Para qué se utiliza Spark?

Spark es un motor de procesamiento de datos de propósito general, un conjunto de herramientas impulsado por API que los científicos de datos y los desarrolladores de aplicaciones incorporan en sus aplicaciones para consultar, analizar y transformar datos rápidamente a escala. La flexibilidad de Spark lo hace muy adecuado para abordar una variedad de casos de uso, y es capaz de manejar varios petabytes de datos a la vez, distribuidos en un clúster de miles de servidores físicos o virtuales que cooperan. Los casos de uso típicos incluyen (Luu, 2018):

- **Procesamiento de flujos:** Desde archivos de registro hasta datos de sensores, los desarrolladores de aplicaciones tienen que lidiar cada vez más con "flujos" de datos. Estos datos llegan en un flujo constante, a menudo de múltiples fuentes simultáneamente. Es factible permitir que estos flujos de datos se almacenen en disco y se analicen retrospectivamente, a veces puede ser sensato o importante procesar y actuar sobre los datos a medida que llegan. Los flujos de datos relacionados con transacciones financieras, por ejemplo, se pueden procesar en tiempo real para identificar y rechazar transacciones potencialmente fraudulentas.

- **Aprendizaje automático:** A medida que aumentan los volúmenes de datos, los enfoques de aprendizaje automático se vuelven más factibles y cada vez más precisos. El software se puede entrenar para identificar y actuar sobre los factores desencadenantes dentro de conjuntos de datos bien entendidos antes de aplicar las mismas soluciones a datos nuevos y desconocidos. La capacidad de Spark para almacenar datos en la memoria y ejecutar consultas repetidas rápidamente lo hace adecuado para entrenar algoritmos de aprendizaje automático. La ejecución de consultas muy similares una y otra vez, a escala, reduce significativamente el tiempo necesario para iterar a través de un conjunto de posibles soluciones a fin de encontrar los algoritmos más eficientes.

- **Análisis interactivo:** En lugar de ejecutar consultas predefinidas para crear cuadros de mando estáticos de ventas o productividad de la línea de producción o precios de las acciones, los analistas comerciales y los científicos de datos desean cada vez más explorar sus datos haciendo una pregunta, viendo el resultado y luego alterando el pregunta inicial ligeramente o profundizando en los resultados. Este proceso de consulta interactivo requiere sistemas como Spark que sean

capaces de responder y adaptarse rápidamente.

- **Integración de datos:** los datos producidos por diferentes sistemas en una empresa rara vez son lo suficientemente limpios o consistentes como para combinarlos de manera simple y sencilla para informes o análisis. Los procesos de extracción, transformación y carga (ETL) se utilizan a menudo para extraer datos de diferentes sistemas, limpiarlos y estandarizarlos y luego cargarlos en un sistema separado para su análisis. Spark (y Hadoop) se utilizan cada vez más para reducir el costo y el tiempo necesarios para este proceso ETL.

2.4.3 Conjuntos de datos distribuidos resistentes (RDD)

El conjunto de datos distribuido resistente es un concepto en el corazón de Spark (Figura 2.3). Está diseñado para admitir el almacenamiento de datos en memoria, distribuidos en un clúster de una manera demostrable tolerante a fallas y eficiente. La tolerancia a fallas se logra, en parte, mediante el seguimiento del linaje de transformaciones aplicadas a conjuntos de datos de grano grueso. La eficiencia se logra mediante la paralelización del procesamiento en varios nodos del clúster y la minimización de la replicación de datos entre esos nodos. Una vez que los datos se cargan en un RDD, se pueden realizar dos tipos básicos de operación (Luu, 2018):

- **Transformaciones:** Se crean un nuevo RDD cambiando el original a través de procesos como mapeo, filtrado y más.

- **Acciones:** Como un contador, que miden pero no modifican los datos originales.

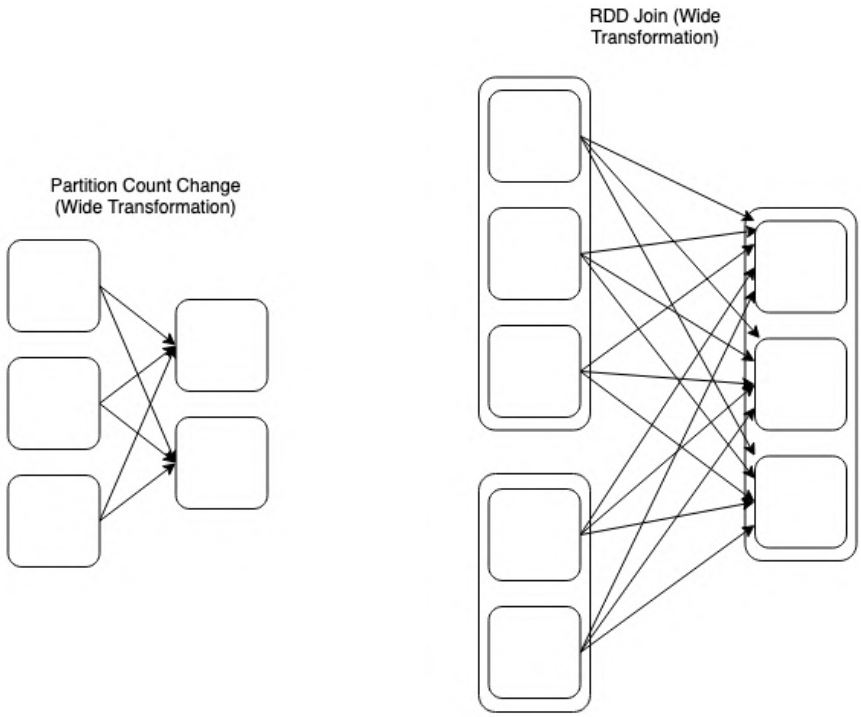


Figura 2.3: *Tipos de Transformación*
Fuente: *(Luu, 2018)*

El RDD original permanece sin cambios en todo momento. La cadena de transformaciones de RDD1 a RDDn se registra y se puede repetir en caso de pérdida de datos o falla de un nodo del clúster. Se dice que las transformaciones se evalúan de forma perezosa, lo que significa que no se ejecutan hasta que una acción posterior necesita el resultado. Esto normalmente mejorará el rendimiento, ya que puede evitar la necesidad de procesar datos innecesariamente. También puede, en determinadas circunstancias, introducir cuellos de botella en el procesamiento que hagan que las aplicaciones se detengan mientras se espera que concluya una acción de procesamiento. Cuando es posible, estos RDD permanecen en la memoria, lo que aumenta en gran medida el rendimiento del clúster, particularmente en casos de uso con un requisito de consultas o procesos iterativos.

2.4.4 SparkSession

Las aplicaciones de PySpark comienzan con la inicialización de SparkSession, que es el punto de entrada de PySpark como se muestra a continuación.

```
from pyspark.sql import SparkSession
from pyspark import SparkConf,SparkContext
spark=SparkSession.builder.appName('data_processing').getOrCreate()
import pyspark.sql.functions as F
from pyspark.sql.types import *
```

2.4.5 SparkContext

Es un punto de entrada a las funcionalidades de Spark desde python. SparkContext representa la conexión a un cluster de spark y se usar todos los métodos para crear y manipular RDDs (Luu, 2018), acumuladores y variables de difusión en ese cluster.

```
from pyspark import SparkConf, SparkContext
conf = SparkConf().setMaster("local").setAppName("Conteo de palabras
    ")
# Inicializo el Spark Context
sc = SparkContext(conf = conf)
```

2.5 Técnicas de preprocesamiento para Big Data

El preprocesamiento es un paso extremadamente importante en el Aprendizaje Automático (*Machine Learning*) y en el Aprendizaje Profundo (*Deep Learning*). No podemos simplemente volcar los datos sin procesar en un modelo y esperar que funcione bien. Incluso si construimos un modelo complejo y bien estructurado, su rendimiento es tan bueno como los datos que le proporcionamos. Por lo tanto, es necesario para procesar los datos en bruto para aumentar el rendimiento de los modelos.
En las siguientes secciones se mostrara los diferentes método y técnicas de preprocesamiento de *Big Data*:

2.5.1 Estandarización

La estandarización es una técnica de escalado donde los valores se centran alrededor de la media con una desviación estándar unitaria (García et al., 2015). Esto significa que la media del atributo se vuelve cero y la distribución resultante tiene una desviación estándar unitaria, a continuación se muestra la formula de estandarización (Ecuación 1):

$$X' = \frac{X - \mu}{\sigma} \tag{1}$$

Tal que, μ es la media de los valores de la característica y σ es la desviación estándar de los valores de la característica, resaltamos que en la estandarización los valores no restringidos a un rango de valores específico, diferentemente de la normalización.
La estandarización permite mejorar la tasa convergencia en el proceso de optimización, y previene características que ejercen una alta variación en la etapa del entrenamiento del aprendizaje de automático.

Por ejemplo, cuando los conjuntos de datos (datasets) que son proveídos a los modelos de aprendizaje automático suelen tener muchas características. Generalmente los valores de diferentes características están en una escala diferente. Consideremos un modelo que intenta predecir los precios de un inmueble. El área de una casa es de unos 200 metros cuadrados, mientras que la antigüedad suele ser inferior a 20. El número de dormitorios puede ser 1, 2 ó 3 en la mayoría de los casos. Todas estas características son importantes para determinar el precio de una casa.
Sin embargo, si los usamos sin ningún tipo de escala, los modelos de aprendizaje automático podrían dar más importancia a las características con valores más altos. Los modelos tienden a funcionar mejor y convergen más rápido cuando las características están en una escala relativamente similar.

Una solución a este problema es la **estandarización**. Primero consideramos las columnas como variables. Si una columna está estandarizada, el valor medio de la columna se resta de cada valor y luego los valores se dividen por la desviación estándar

de la columna. Las columnas resultantes tienen una desviación estándar de 1 y una media muy cercana a cero. Así, terminamos teniendo variables (columnas) que tienen una distribución casi normal. La estandarización se puede lograr con **StandardScaler**, función que esta presente en Apache Spark [1].

Estandarización en Apache Spark

1. Importar librerías: Como paso inicial siempre se importan las librerías que se van utilizar:

```
import pandas as pd
import numpy as np
from pyspark.sql import SparkSession
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StandardScaler
```

2. Inicializar Apache Spark: En este paso se configura el Apache Spark utilizando las funciones *SparkSession* y *sparkContext*:

```
spark = SparkSession.builder \
    .master("local") \
    .appName("standard_scaler") \
    .config("spark.executor.memory", "1gb") \
    .getOrCreate()
sc = spark.sparkContext
```

3. Generar datos: En este paso son generados datos aleatorios utilizando la librería *numpy*, para simular datos con 3 características (columnas):

```
a = np.random.randint(10, size=(10,1))
b = np.random.randint(50, 100, size=(10,1))
c = np.random.randint(500, 700, size=(10,1))
X = np.concatenate((a,b,c), axis=1)
X
```

4. Vectorizar datos: VectorAssembler es un transformador (transformer) que combina una lista dada de columnas en una sola columna vectorial. Es útil para combinar características sin procesar y características generadas por diferentes transformadores de características en un solo vector de características.

   En este caso con 3 elementos cada uno, para este paso usamos la libreria *MLlib* con la función *VectorAssembler*:

```
df = pd.DataFrame(X, columns = ['a','b','c'])
mydf = spark.createDataFrame(df)
```

[1] How to Standardize or Normalize Data with PySpark: https://www.youtube.com/watch?v=ucOin2KFt9E

```
feature_columns = mydf.columns
vector_assembler = VectorAssembler(inputCols = feature_columns,
    outputCol = 'features')
features_df = vector_assembler.transform(mydf).select(['features
    '])
features_df.show()

+----------------+
|        features|
+----------------+
|[9.0,61.0,556.0]|
|[4.0,73.0,549.0]|
|[8.0,85.0,624.0]|
|[3.0,84.0,658.0]|
|[3.0,67.0,618.0]|
|[3.0,95.0,660.0]|
|[3.0,92.0,655.0]|
|[4.0,71.0,616.0]|
|[3.0,50.0,543.0]|
|[4.0,66.0,596.0]|
+----------------+
```

5. Estandarización: En este último paso los valores de las características (*features*) son mapeadas para valores proporcionales, por ejemplo entre $[2, -2]$ o $[0, 1]$:

```
standardizer = StandardScaler(withMean=True, withStd=True,
    inputCol='features', outputCol='std_features')
standardizer_model = standardizer.fit(features_df)
standardized_features_df = standardizer_model.transform(
    features_df)
standardized_features_df.show()

+----------------+--------------------+
|        features|        std_features|
+----------------+--------------------+
|[9.0,61.0,556.0]|[2.07103577689744...|
|[4.0,73.0,549.0]|[-0.1800900675562...|
|[8.0,85.0,624.0]|[1.62081060800669...|
|[3.0,84.0,658.0]|[-0.6303152364470...|
|[3.0,67.0,618.0]|[-0.6303152364470...|
|[3.0,95.0,660.0]|[-0.6303152364470...|
|[3.0,92.0,655.0]|[-0.6303152364470...|
|[4.0,71.0,616.0]|[-0.1800900675562...|
|[3.0,50.0,543.0]|[-0.6303152364470...|
|[4.0,66.0,596.0]|[-0.1800900675562...|
+----------------+--------------------+
```

2.5.2 Transformación de datos

OneHotencoder: Es una técnica para convertir atributos categóricos en un vector binario. OneHotencoder asigna una columna de índices de categoría a una columna de vectores binarios, con como máximo un único valor por fila que indica el índice de categoría de entrada.
Por ejemplo, con 5 categorías, un valor de entrada de 2 se asignaría a un vector de salida de $[0, 0, 1, 0]$.

2.5.3 Big Data Imperfecta

En el preprocesamiento de datos, es común emplear técnicas para eliminar los datos con ruido o para imputar los datos faltantes.

Se tiene dos formas:

- Imputación de valores perdidos.
- Identificación de ruido.

2.5.4 Discretización de Big Data

La discretización se centra en la transformación de valores continuos con un orden entre los valores nominales o categóricos sin ordenar (García et al., 2015). También es una cuantificación de atributos numéricos. Los valores nominales están dentro de un dominio finito, por lo que también se consideran una técnica de reducción de datos (Chan et al., 2019).
Algunas ventajas que se tiene al usar esta técnica:

- Divida el rango de atributos numéricos (continuos o no) en intervalos.
- Almacena las etiquetas de los intervalos.
- Es crucial para las reglas de asociación y algunos algoritmos de clasificación, que solo aceptan datos discretos.

El proceso de la discretización tienen las siguientes etapas (Figura 2.4).
Descripción de las etapas en el proceso de discretización:

- En esta etapa el uso del algoritmo de clasificación(Quick Sort) es fundamental, esta clasificación se debe hacer solo un vez y al empezar el clasificador. Si la discretización esta dentro del proceso de otro algoritmo así como inducción de un árbol de decisión el tratamiento es local y caso contrario es una región.
- En esta etapa se hace la selección del mejor punto de corte o el mejor par de intervalos adyacentes se debe encontrar en el rango de atributos para dividir o

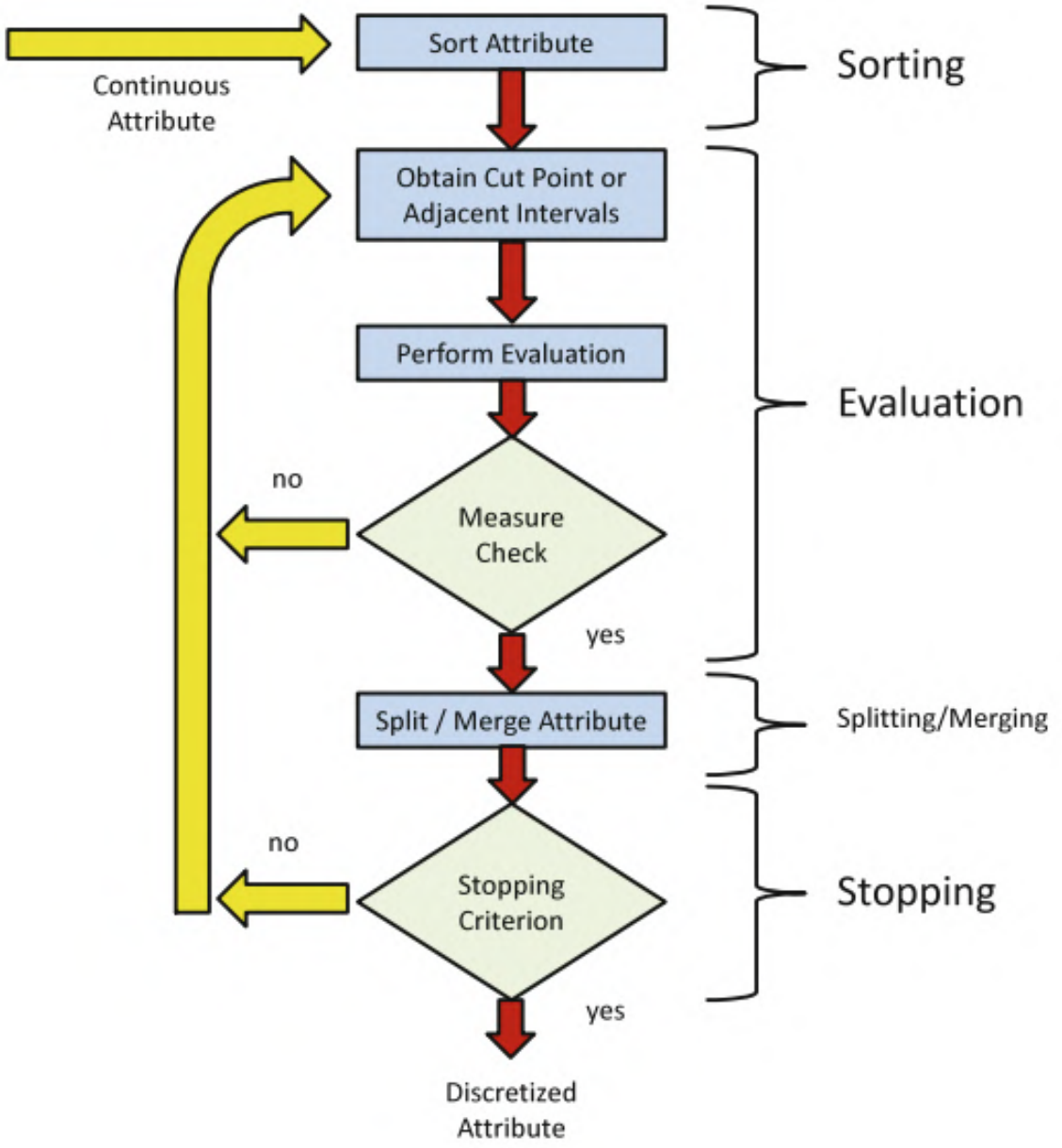


Figura 2.4: *Proceso de la discretización*
Fuente: *(Chan et al., 2019)*

fusionar en el siguiente paso requerido. Una medida de evaluación o función se usa para determinar la correlación, ganancia, mejora en el desempeño y cualquier otro beneficio de acuerdo con la clase.

- Los intervalos pueden dividirse o fusionarse. Para dividir todos los puntos de corte posibles dentro de un atributo, se debe evaluar. La discretización continúa con cada parte hasta que se satisface un criterio de parada. De manera similar, para la fusión, en lugar de encontrar el mejor punto de corte, el discretizador tiene como objetivo encontrar los mejores intervalos adyacentes para fusionar en cada iteración. La discretización continúa con el número reducido de intervalos hasta que se satisface el criterio.

- En esta estapa se detiene el proceso de discretización teniendo una simplicidad con alta precisión.

Discretización en Apache Spark

Spark DStream (Discretized Stream)

Es un concepto básico de *streaming* (datos continuos) en Spark (Chan et al., 2019). El streming de datos pueden provenir de muchas fuentes, por ejemplos de otras interfaces o de sockets TCP, por otro lado DStream es también un *streaming* continuo de RDD (Resilient Distribuited Datasets). Cada RDD en un DStream (*Data Stream*) contiene datos de un cierto intervalo de tiempo.

Caso de uso: Contador de palabras

En un escenario donde a cada segundo 9×10^3 tweets son enviados, 1000 fotos son subidas a Instagram, más de 2×10^6 de correos electrónicos son enviados y cerca de 80×10^3 búsquedas son realizadas según el Internet Live Stats.

De ahí la importancia del procesamiento y analisis de grandes volúmenes de datos continuos generados a cada segundo (Streaming).

Veamos a continuación el ejemplo de discretización en un contador de palabras usando DStream:

1. **Configurar e inicializar librerias:** En está etapa se configura el intervalo de tiempo en que se leerán los datos, por ejemplo a cada 5 segundos:

```
from pyspark.context import SparkContext
from pyspark.streaming import StreamingContext
from time import sleepsc=SparkContext("local[*]","
   StreamingExample")
ssc=StreamingContext(sc,5)
```

2. **Leer datos:** En este caso nuestra fuente es un archivo de texto:

```
lines=ssc.textFileStream(r'home/data')
```

3. **Sumar palabras:** Aquí se utiliza la función Map-Reduce para sumar las palabras:

```
words=lines.flatMap(lambda x:x.split(" "))
mapped_words=words.map(lambda x:(x,1))
reduced_words=mapped_words.reduceByKey(lambda x,y:x+y)
```

4. **Ordenar atributos:**

```
sorted_words=reduced_words.map(lambda x:(x[1],x[0])).transform(
    lambda x:x.sortByKey(False))
```

5. **Simulación del *streaming*:**

```
sorted_words.pprint()ssc.start()
sleep(20)
ssc.stop(stopSparkContext=True, stopGraceFully=True)
```

Capítulo III

Desarrollo de la Metodología

3 Metodología Propuesta para Verificar la Calidad de Datos

Proponemos utilizar test unitarios e integrales (Wang et al., 2021) para verificar y cuantificar la calidad de datos (*smart data*), en cada etapa se utilizan técnicas de preprocesamiento, como resultado se obtienen datos limpios con todos los test subsanados satisfactoriamente.

En ese sentido la metodología propuesta garantiza la integridad de la calidad de datos (*Smart Data*) para cualquier conjunto de datos (*dataset*) a través de diferentes técnicas de preprocesamiento de datos utilizando la herramienta *Apache Spark*.

3.1 Manipulación de datos

Mientras Apache Spark es una buena herramienta para transformación de datos, nuestro objetivo es asegurar que los datos de salida o resultados cumplan con estándar de calidad esperado, y usualmente se evalúa:

- Si cada columna es obligatoria.
- Si los valores son validos.
- Si el formato esta correcto (por ejemplo, fechas).
- Si la columna es numérica, debe estar en un cierto rango.

Todas estas cuestiones forman reglas para validación.

3.2 Modelo a seguir

Se aplicarán varias transformaciones (preprocesamientos) al dataset para obtener data de calidad, por ejemplo, se reducirán datos con ruido (valores nulos o repetidos) para lo cual utilizamos diferentes técnicas de preprocesamiento. (Figura 3.1). En este caso de uso pretendemos ejemplificar el uso de *tests* unitarios para garantizar calidad de datos con herramientas de *Big Data* (Apache Spark), con técnicas de preprocesamiento en cada etapa. Los test unitarios o integrales se elaboran para verificar la validez de las funcionalidades en una plataforma de software, se mide en términos de porcentaje según se van cumpliendo las pruebas satisfactoriamente.
Cuando un conjunto de datos (dataset) es tratado, se espera que contengan las

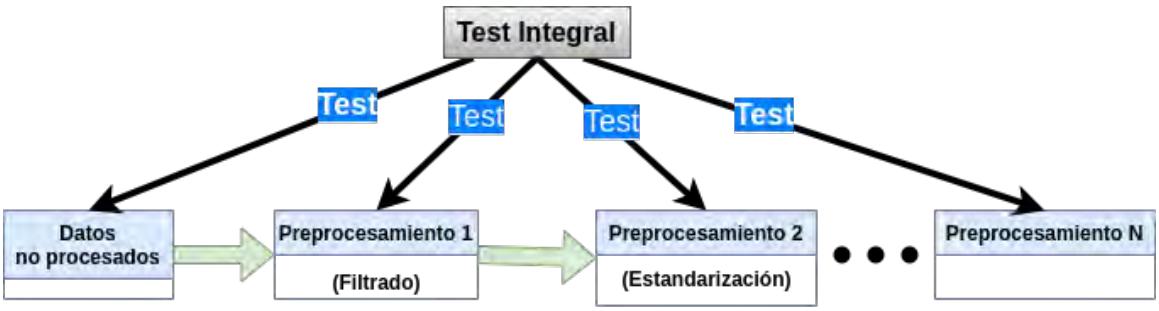


Figura 3.1: *Modelo de transformación de datos con test unitarios e integral con preprocesamiento en cada etapa.*
Fuente: *Propia*

columnas solicitadas, que los valores sean validos, que estén en un formato adecuado, por ejemplo los campos o atributos relacionados a fechas, todo esto para que cumplan con las reglas de negocio (Figura 3.2).

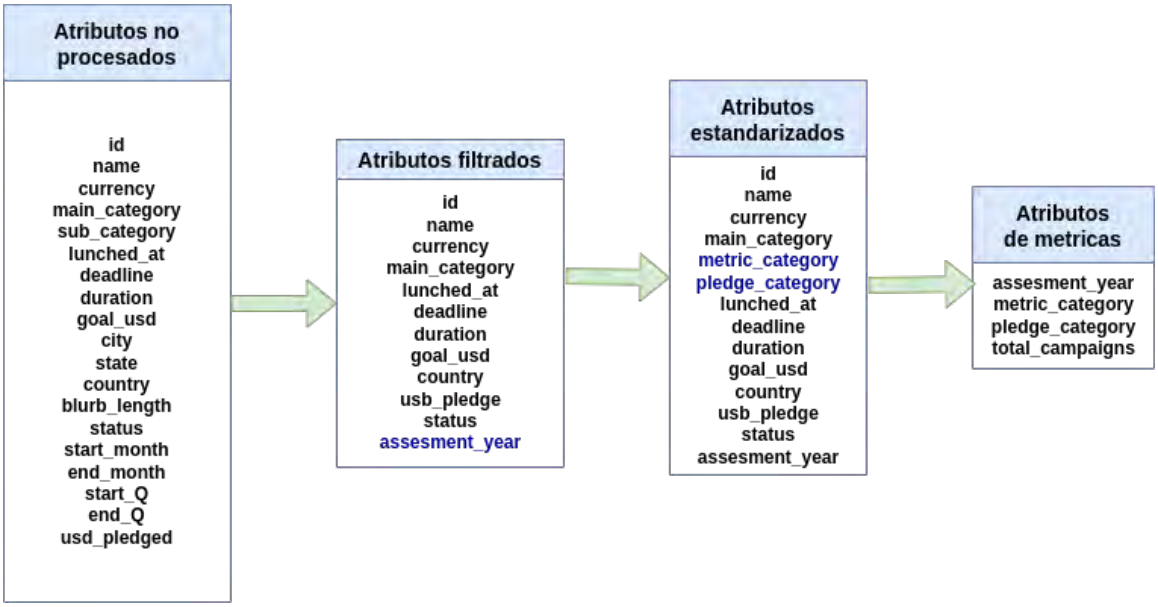


Figura 3.2: *Ejemplo de transformación de datos en cada etapa de los test unitarios*
Fuente: *Propia*

- **Pruebas unitarias e integrales para datos de calidad:**

PySpark posee herramientas para realizar este tipo de validaciones de datos con pruebas unitarias e integrales, por ejemplo la libreria *Great Expectations*[1].

Para ejemplificar el presente modelo metodológico de cuantificación de calidad de datos (*Smart Data*), utilizamos un dataset de proyectos financiados en diferentes

[1]Great Expectations: https://docs.greatexpectations.io/docs/

ciudades del mundo (**city**), de varios rubros (**main_category**) entre el periodo de 2014 al 2019[2], con 192548 registros y 20 columnas.

3.2.1 Exploración de datos sin procesar

En esta primera etapa se realizan las siguientes tareas:

- Importar las librerías de Apache Spark y *Great Expectations* para realizar las pruebas, por ejemplo Spark Session:

```
from great_expectations.dataset import SparkDFDataset
from pyspark.sql import SparkSession
import pandas as pd
spark = SparkSession.builder.master("local[*]").appName("Great
    Expectations with Pandas DataFrame").getOrCreate()
```

- Cargar los datos y crear una vista del dataset:

```
raw_df = spark.read.option("header", True).csv(folder+"/
    Kickstarter_projects_Feb19.csv")
raw_df.createOrReplaceTempView("CAMPAIGNS")
```

- Explorar el dataset:

```
raw_df.toPandas()
```

	id	name	currency	main_category	sub_category	launched_at	deadline	duration	goal_usd	city	state	country	blurb_length	name_length	status	start_month	end_month	start_Q
153		Socks of Speed and Socks of Elvenkind	USD	games	Tabletop Games	2018-10-30 20:00:02	2018-11-15 17:59:00	16.0	2000.0	Menasha	WI	US	14	7	successful	10	11	Q4
657		Power Punch Boot Camp: An All-Ages Graphic Novel	GBP	comics	Comic Books	2018-08-06 10:00:43	2018-09-05 10:00:43	30.0	3870.99771	Shepperton	England	GB	24	8	successful	8	9	Q3
436		"Live Printing with SX8: ""Squeegee Pulp Up"""	USD	fashion	Apparel	2017-06-09 15:41:03	2017-07-09 15:41:03	30.0	1100.0	Manhattan	NY	US	21	7	successful	6	7	Q2
071		Lost Dog Street Band's Next Album	USD	music	Country & Folk	2014-09-25 18:46:01	2014-11-10 06:00:00	45.0	3500.0	Nashville	TN	US	15	6	successful	9	11	Q3

- Crear una instancia de los datos no procesados utilizando la libreria *Great Expectations*:

```
raw_test_df = SparkDFDataset(raw_df)
```

La instancia del dataset para realizar los test es raw_test_df.

3.2.2 Test unitario en los datos no procesados

- Comprobar si las columnas obligatorias existen:

[2]Kickstarter krowfunding: `https://www.kaggle.com/code/kerneler/starter-kickstarter-campaigns-9f7d98d1-9/data`

```
MANDATORY_COLUMNS = [
  "id",
  "currency",
  "main_category",
  "launched_at",
  "deadline",
  "country",
  "status",
  "usd_pledged"
]

for column in MANDATORY_COLUMNS:
    try:
        assert raw_test_df.expect_column_to_exist(column).
    success, f"Uh oh! Mandatory column {column} does not exist:
    FAILED"
        print(f"Column {column} exists : PASSED")
    except AssertionError as e:
        print(e)
==============output==============================
Column id exists : PASSED
Column currency exists : PASSED
Column main_category exists : PASSED
Column launched_at exists : PASSED
Column deadline exists : PASSED
Column country exists : PASSED
Column status exists : PASSED
Column usd_pledged exists : PASSED
```

- Las columnas obligatorias no deben ser nulas:

```
for column in MANDATORY_COLUMNS:
    try:
        test_result = raw_test_df.
    expect_column_values_to_not_be_null(column)
        assert test_result.success, \
            f"Uh oh! {test_result.result['unexpected_count']} of
    {test_result.result['element_count']} items in column {
    column} are null: FAILED"
        print(f"All items in column {column} are not null:
    PASSED")
    except AssertionError as e:
        print(e)
    except AnalysisException as e:
        print(e)
==============output==============================
All items in column id are not null: PASSED
All items in column currency are not null: PASSED
Uh oh! 1 of 192548 items in column main_category are null:
```

```
   FAILED
Uh oh! 1 of 192548 items in column launched_at are null: FAILED
Uh oh! 1 of 192548 items in column deadline are null: FAILED
Uh oh! 1 of 192548 items in column country are null: FAILED
Uh oh! 1 of 192548 items in column status are null: FAILED
Uh oh! 1 of 192548 items in column usd_pledged are null: FAILED
```

```
--+------------------+------------------+------------+-----------+----------+--------+--------+--------+
id|              name|          currency|main_category|sub_category|launched_at|deadline|duration|goal_usd|
--+------------------+------------------+------------+-----------+----------+--------+--------+--------+
63|"In The Jungle 12...|,GBP,music,Electr...|        null|       null|      null|    null|    null|    null|
--+------------------+------------------+------------+-----------+----------+--------+--------+--------+
```

El test encontro varios valores nulos.

- Comprobar el formato de fecha válido de los campos launched_at y deadline:

```
test_result =  raw_test_df.
   expect_column_values_to_match_strftime_format('launched_at','
   %Y-%m-%d %H:%M:%S')
f"""{round(test_result.result['unexpected_percent'], 2)}% is not
    a valid date time format"""
====================output============================
0.29% is not a valid date time format
```

```
test_result =  raw_test_df.
   expect_column_values_to_match_strftime_format('deadline','%Y
   -%m-%d %H:%M:%S')
f"""{round(test_result.result['unexpected_percent'], 2)}% is not
    a valid date time format"""
====================output============================
0.04% is not a valid date time format
```

- Comprobar si los valores son únicos:

```
test_result = raw_test_df.expect_column_values_to_be_unique("id"
   )
failed_msg = " ".join([f"""Uh oh!""",
              f"""{test_result.result['unexpected_count']} of {
   test_result.result['element_count']} items""",
              f"""or {round(test_result.result['
   unexpected_percent'],2)}% are not unique: FAILED"""])
print(f"""{'Column id is unique: PASSED' if test_result.success
   else failed_msg}""")
====================output============================
Uh oh! 48162 of 192548 items or 25.01% are not unique: FAILED
```

	id	count
0	721760621	2
1	176761740	2
2	1071363810	2
3	1601388640	2
4	1353689425	2
...	...	...
24076	145298230	2
24077	492658907	2
24078	655103249	2
24079	894641055	2
24080	1369268884	2

24081 rows × 2 columns

En este test se observa que el dataset tiene mucha información duplicada.

Hasta este punto puede que los datos estén duplicados entonces será necesario realizar una segunda etapa de filtrado de datos.

3.2.3 Filtrado de datos

- El filtrado se debe realizar de acuerdo a las reglas de negocio, osea filtrar datos que estén relacionados a lo que queremos analizar o visualizar. Por ejemplo, solo utilizaremos valores de **US** del campo **country** para los periodos de 2017 a 2018, solo para ciertas categorías especificadas, campañas exitosas, con tipo de moneda **USD**:

```
MAIN_CATEGORIES = [
    'art',
    'publishing',
    'film & video',
    'technology',
    'journalism',
    'food',
    'dance',
    'photography',
    'games',
    'crafts',
    'music',
    'comics',
    'theater',
    'design'
]
ASSESSMENT_YEAR = ['2017','2018']
COUNTRY = 'US'
```

```
CURRENCY = 'USD'
```

- Definir periodos de evaluación (rangos de fechas de los periodos 2017 y 2018) de los datos:

```
assessment_year_reference = {
    'assessment_year': ['2017', '2018'],
    'period_start_dt': ['2016-07-01', '2017-07-01'],
    'period_end_dt': ['2017-06-30', '2018-06-30'],
}
ay_df = pd.DataFrame(data=assessment_year_reference)
spark_ay_df = spark.createDataFrame(ay_df)
spark_ay_df.createOrReplaceTempView("assessment_year_ref")
```

- Aplicar transformaciones:

```
filtered_df = spark.sql(f"""
    SELECT id,
           name,
           currency,
           main_category,
           launched_at,
           deadline,
           goal_usd,
           country,
           usd_pledged,
           status,
           assessment_year
    FROM (SELECT t.*,
                 ay.assessment_year,
                 row_number() OVER (
                     PARTITION BY t.id
                     ORDER BY t.launched_at,
                              ay.assessment_year DESC) row_no
          FROM CAMPAIGNS t
          INNER JOIN assessment_year_ref ay
              ON TO_DATE(t.launched_at) <= ay.period_end_dt
              AND t.deadline > ay.period_start_dt
          WHERE country = '{COUNTRY}'
          AND status = 'successful'
          AND main_category IN ('{"','".join(MAIN_CATEGORIES)}')
          AND ay.assessment_year IN ('{"','".join(
   ASSESSMENT_YEAR)}')
          AND currency = '{CURRENCY}'
   ) WHERE row_no = 1
    """)
filtered_df.createOrReplaceTempView("FILTERED_CAMPAIGNS")
```

- Explorar los datos filtrados:

```
filtered_df.toPandas()
```

id	name	currency	main_category	launched_at	deadline	goal_usd	country	usd_pledged	status	assessment_year
193	Euphorium	USD	film & video	2018-03-16 21:07:47	2018-04-10 21:07:47	1500.0	US	1500.0	successful	2018
129	Adult Colour Debut Album	USD	music	2017-05-25 20:27:33	2017-06-24 20:27:33	2200.0	US	2415.0	successful	2017
176	The Book of Maxwell	USD	journalism	2017-07-12 21:47:37	2017-08-07 02:59:00	1500.0	US	1804.0	successful	2018
887	Peak View Brewing Company	USD	food	2016-09-10 12:25:04	2016-10-10 12:25:04	15000.0	US	15018.0	successful	2017
505	Fidget Hoop: IT'S NOT WHAT IT LOOKS LIKE	USD	games	2018-03-07 14:33:20	2018-04-21 13:33:20	900.0	US	1051.0	successful	2018

- Crear una instancia de los datos filtrados:

```
filtered_test_df = SparkDFDataset(filtered_df)
```

3.2.4 Test Unitario de los datos filtrados

- Comprobar la existencia de las categorías especificadas:

```
test_result = filtered_test_df.expect_column_values_to_be_in_set
    ("main_category", MAIN_CATEGORIES)
print(f"""Categories are within scope: {'PASSED' if test_result.
    success else 'FAILED'}""")
========================output==========================
Categories are within scope: PASSED
```

- Verificar la presencia de los campos country, status y currency:

```
test_result = filtered_test_df.expect_column_values_to_be_in_set
    ("country", ["US"])
print(f"""All campaigns are done in the country of USA: {'PASSED
    ' if test_result.success else 'FAILED'}""")
========================output==========================
All campaigns are done in the country of USA: PASSED
```

```
test_result = filtered_test_df.expect_column_values_to_be_in_set
    ("status", ["successful"])
print(f"""All campaigns are successful: {'PASSED' if test_result
    .success else 'FAILED'}""")
========================output==========================
All campaigns are successful: PASSED
```

```
test_result = filtered_test_df.expect_column_values_to_be_in_set
    ("currency", ["USD"])
print(f"""All campaigns are successful: {'PASSED' if test_result
    .success else 'FAILED'}""")
========================output==========================
All campaigns are successful: PASSED
```

- Verificar la presencia de los campos necesarios (columnas) y que no sean nulos:

```
for column in MANDATORY_COLUMNS:
    try:
```

```
        test_result = filtered_test_df.
    expect_column_values_to_not_be_null(column)
        assert test_result.success, f"Uh oh! {test_result.result
    ['unexpected_count']} of {test_result.result['element_count
    ']} items in column {column} are null: FAILED"
        print(f"All items in column {column} are not null:
    PASSED")
     except AssertionError as e:
        print(e)
========================output=========================
All items in column id are not null: PASSED
All items in column currency are not null: PASSED
All items in column main_category are not null: PASSED
All items in column launched_at are not null: PASSED
All items in column deadline are not null: PASSED
All items in column country are not null: PASSED
All items in column status are not null: PASSED
All items in column usd_pledged are not null: PASSED
```

- Verificar que no haya duplicados:

```
test_result = filtered_test_df.
    expect_compound_columns_to_be_unique(["id", "assessment_year"
    ])
print(f"""id column is unique for each assessment year: {'PASSED
    ' if test_result.success else 'FAILED'}""")
========================output=========================
id column is unique for each assessment year: PASSED
```

- Comprobar el formato de fecha válido de los campos launched_at y deadline:

```
test_result =  filtered_test_df.
    expect_column_values_to_match_strftime_format('launched_at','
    %Y-%m-%d %H:%M:%S')
f"""launched_at column values are compliant to datetime format:
    {'PASSED' if test_result.success else 'FAILED'}"""
========================output=========================
launched_at column values are compliant to datetime format:
    PASSED
```

```
test_result =  filtered_test_df.
    expect_column_values_to_match_strftime_format('deadline','%Y
    -%m-%d %H:%M:%S')
f"""deadline column values are compliant to datetime format: {'
    PASSED' if test_result.success else 'FAILED'}"""
========================output=========================
deadline column values are compliant to datetime format: PASSED
```

En la siguiente etapa se aplica el preprocesamiento de estandarización.

3.2.5 Estandarización de los datos

- Crear reglas para la estandarización (reducir a 6 categorías):

```
METRIC_CATEGORIES = [
    'art, crafts, photography & design',
    'dance & theater',
    'music, film & video',
    'comics, publishing & journalism',
    'games & technology',
    'food'
]
```

```
PLEDGE_CATEGORIES = [
    '100 Thousand and under',
    'Between 100 Thousand and 500 Thousand',
    'Between 500 Thousand and 1 Million',
    'Between 1 Million and 5 Million',
    '5 Million and over'
]
```

- Transformación de datos y creación de la vista STANDARDISED_CAMPAIGNS:

```
standardised_df = spark.sql(f"""
    SELECT t.*,
         CASE
             WHEN main_category IN ('art','crafts','photography','design') THEN 'art, crafts, photography & design'
             WHEN main_category IN ('dance','theater') THEN 'dance & theater'
             WHEN main_category IN ('music','film & video') THEN 'music, film & video'
             WHEN main_category IN ('comics','publishing','journalism') THEN 'comics, publishing & journalism'
             WHEN main_category IN ('games', 'technology') THEN 'games & technology'
             WHEN main_category IN ('food') THEN 'food'
            END metric_category,
         CASE WHEN usd_pledged <= 100000 THEN '100 Thousand and under'
             WHEN usd_pledged > 100000 AND usd_pledged < 500000 THEN 'Between 100 Thousand and 500 Thousand'
             WHEN usd_pledged > 500000 AND usd_pledged < 1000000 THEN 'Between 500 Thousand and 1 Million'
             WHEN usd_pledged > 1000000 AND usd_pledged < 5000000 THEN 'Between 1 Million and 5 Million'
            ELSE '5 Million and over'
            END pledge_category
    FROM FILTERED_CAMPAIGNS t
""")
```

```
standardised_df.createOrReplaceTempView("STANDARDISED_CAMPAIGNS")
```

- Exploración de datos:

```
standardised_df.toPandas()
```

- Crear una instancia de los datos estandarizados:

```
standardised_test_df = SparkDFDataset(standardised_df)
```

3.2.6 Test Unitario de los datos estandarizados

- Comprobar si las campos de categorías que definimos anteriormente están presentes:

```
test_result = standardised_test_df.
    expect_column_values_to_be_in_set("metric_category",
    METRIC_CATEGORIES)

print(f"""Categories are within scope: {'PASSED' if test_result.
    success else 'FAILED'}""")
========================output============================
Categories are within scope: PASSED
```

```
test_result = standardised_test_df.
    expect_column_values_to_be_in_set("pledge_category",
    PLEDGE_CATEGORIES)

print(f"""Categories are within scope: {'PASSED' if test_result.
    success else 'FAILED'}""")
========================output============================
Categories are within scope: PASSED
```

- Comprobar si la población es igual al conjunto de datos anterior (los datos filtrados con los estandarizados):

```
filtered_total_rows = filtered_test_df.get_row_count()
test_result = standardised_test_df.
    expect_table_row_count_to_equal(filtered_total_rows)

print(f"""Total row count of standardised_df ({test_result.
    result['observed_value']}) \
is equal to total row count of filtered_df ({filtered_total_rows
    }): \
{'PASSED' if test_result.result['observed_value'] ==
    filtered_total_rows else 'FAILED'}""")
========================output============================
Total row count of standardised_df (17856) is equal to total row
     count of filtered_df (17856): PASSED
```

Se espera que tengan el mismo número de columnas.

3.2.7 Producir las métricas finales

En esta última etapa son verificados campos o atributos precalculados (métricas) para análisis de datos.

- Definir las métricas para cada restricción establecida, por ejemplo para las categorías definidas y por periodo (año):
- Aplicar transformaciones y crear una vista con el nombre SUCCESSFUL_CAMPAIGNS:

```
successful_campaigns_df = spark.sql(f"""
    SELECT assessment_year,
           metric_category,
           pledge_category,
           count(id) total_successful_campaigns
    FROM STANDARDISED_CAMPAIGNS
    GROUP BY assessment_year,
            metric_category,
            pledge_category
    ORDER BY assessment_year,
            metric_category,
            pledge_category
    """)
    successful_campaigns_df.createOrReplaceTempView("
   SUCCESSFUL_CAMPAIGNS")
```

- Explorar datos:

```
successful_campaigns_df.toPandas()
```

id	name	currency	main_category	launched_at	deadline	goal_usd	country	usd_pledged	status	assessment_year	metric_category	pledge_category
783	WarmUp: A High-Protein Coffee	USD	food	2016-11-14 17:55:34	2016-12-14 17:55:34	8000.0	US	8599.0	successful	2017	food	100 Thousand and under
089	Silver Linings by J.R. Mounts	USD	comics	2018-01-10 02:10:01	2018-02-16 05:00:00	500.0	US	2914.5	successful	2018	comics, publishing & journalism	100 Thousand and under
344	Princes of the Universe: a Nino Malong Art Book	USD	publishing	2016-07-31 19:03:04	2016-09-05 01:00:00	1200.0	US	1670.0	successful	2017	comics, publishing & journalism	100 Thousand and under
445	Art of Cardistry - Geometry Playing Cards Vers...	USD	games	2018-03-03 01:54:57	2018-03-13 00:54:57	1000.0	US	1630.0	successful	2018	games & technology	100 Thousand and under
598	Dead Duck & Zombie Chick Radio Show (vinyl pre...	USD	comics	2018-03-27 11:53:02	2018-04-26 11:53:02	7000.0	US	7656.0	successful	2018	comics, publishing & journalism	100 Thousand and under

- Crear una instancia de los datos evaluados satisfactoriamente:

```
successful_campaigns_df_test_df = SparkDFDataset(
   successful_campaigns_df)
```

3.2.8 Test unitario sobre las métricas finales

- Comprobar que los valores sean únicos, para las categorías creadas por periodo de tiempo:

```
test_result = successful_campaigns_df_test_df.
   expect_compound_columns_to_be_unique(["assessment_year","
   metric_category","pledge_category"])
```

```
2
3 print(f"""metric_category column is unique for each assessment
    year: {'PASSED' if test_result.success else 'FAILED'}""")
4 ==========================ouput=========================
5 metric_category column is unique for each assessment year:
    PASSED
```

- Comprobar la suma total de registros de las métricas es igual al dataset estandarizado:

```
1 standardised_total_rows = standardised_test_df.get_row_count()
2 test_result = successful_campaigns_df_test_df.
    expect_column_sum_to_be_between('total_successful_campaigns',
3
                    standardised_total_rows,
    standardised_total_rows)
4
5 print(f"""Total sum of campaigns in metrics dataset ({
    test_result.result['observed_value']}) \
6 is equal to total rows in standardised dataset ({
    standardised_total_rows}): \
7 {'PASSED' if test_result.result['observed_value'] ==
    standardised_total_rows else 'FAILED'}""")
8 ==========================output============================
9 Total sum of campaigns in metrics dataset (17856) is equal to
    total rows in standardised dataset (17856): PASSED
```

3.2.9 Resumen de todas las validaciones

Juntar todas las validaciones por etapas.

- Test integral de todos los datasets:

```
1 raw_test_df_validation = raw_test_df.validate()
2 print(f"""1. Raw dataset validations: {raw_test_df_validation.
    success}; {raw_test_df_validation.statistics['success_percent
    ']} successful""")
3 filtered_test_df_validation = filtered_test_df.validate()
4 print(f"""2. Filtered dataset validations: {
    filtered_test_df_validation.success}; {
    filtered_test_df_validation.statistics['success_percent']}
    successful""")
5 standardised_test_df_validation = standardised_test_df.validate
    ()
6 print(f"""3. Standardised dataset validations: {
    standardised_test_df_validation.success}; {
    standardised_test_df_validation.statistics['success_percent
    ']} successful""")
7 successful_campaigns_df_test_df_validation =
    successful_campaigns_df_test_df.validate()
```

```
print(f"""4. Metrics dataset validations: {
    successful_campaigns_df_test_df_validation.success}; {
    successful_campaigns_df_test_df_validation.statistics['
    success_percent']} successful""")
```

```
====================output================================
1. Raw dataset validations: False; 52.63157894736842 successful
2. Filtered dataset validations: True; 100.0 successful
3. Standardised dataset validations: True; 100.0 successful
4. Metrics dataset validations: True; 100.0 successful
```

Capítulo IV

Proceso Experimental

4 Proceso Experimental

En este capítulo describimos y mostramos pruebas unitarias para verificar la limpieza de los datos porcentualmente, en cada etapa de los test unitarios se utilizan técnicas de preprocesamiento, se sigue el modelo metodológico del Capítulo 3, luego se utiliza Smart Data en análisis y visualización de datos, finalmente se muestra un ejemplo de Smart Data en aprendizaje automático (*Machine Learning*).

A continuación se presentan pruebas unitarias en diferentes casos de uso:

4.1 Caso de Uso 1: Dataset COVID-19

En este caso de uso se muestra el análisis de datos en Big Data tomando como ejemplo los casos positivos de COVID-19 en Perú. En esta dataset también se aplican pruebas unitarias e integrales con las herramientas mostradas en el capítulo 3.

4.1.1 Dataset COVID-19 Perú

Este conjunto de datos (dataset) al 22 de julio de 2020, tiene 395005 registros con 8 columnas (UUID, DEPARTAMENTO, PROVINCIA, DISTRITO, METODODX, EDAD, SEXO, FECHA_RESULTADO).

Estos datos se encuentran disponibles en el ministerio de salud del gobierno peruano en la página de datos abiertos[1]. Este trabajo se centra en realizar un preprocesamiento de datos para ello se usan ciertas técnicas; limpiar valores nulos, estandarizar, dar formato a los valores de acuerdo al tipo de dato, entre otros.

4.1.2 Exploración de datos sin procesar

En esta etapa se realizan las siguientes tareas:

- Importar las librerías de Apache Spark y *Great Expectations* para realizar las pruebas, por ejemplo Spark Session: En esta etapa se inicializan las librerías y se configura la conexión.

```
from great_expectations.dataset import SparkDFDataset
from pyspark.sql import SparkSession
import pandas as pd
spark = SparkSession.builder.master("local[*]").appName("Casos
    COVID19").getOrCreate()
```

[1]Casos positivos por COVID-19 - [Ministerio de Salud - MINSA], url: https://www.datosabiertos.gob.pe/dataset/casos-positivos-por-covid-19-ministerio-de-salud-minsa

- Cargar los datos y crear una vista del dataset:

```
raw_df = spark.read.option("header", True).csv(folder+"/
    positivos_covid31072020.csv")
raw_df.createOrReplaceTempView("CASOS")
```

- Explorar el dataset:

```
raw_df.toPandas()
```

UUID	DEPARTAMENTO	PROVINCIA	DISTRITO	METODODX	EDAD	SEXO	FECHA_RESULTADO
;134e6	CAJAMARCA	JAEN	JAEN	PCR	64	MASCULINO	20200715
:24c65	LIMA	EN INVESTIGACIÓN	EN INVESTIGACIÓN	PCR	68	MASCULINO	20200716
320a6f	LIMA	LIMA	LOS OLIVOS	PCR	52	MASCULINO	20200716
)5e979	LIMA	EN INVESTIGACIÓN	EN INVESTIGACIÓN	PCR	42	MASCULINO	20200716
5396c3	CALLAO	CALLAO	BELLAVISTA	PCR	34	MASCULINO	20200716

- Crear una instancia de los datos no procesados utilizando la libreria *Great Expectations*:

```
raw_test_df = SparkDFDataset(raw_df)
```

La instancia del dataset para realizar los test es raw_test_df.

4.1.3 Test unitario en los datos no procesados

- Comprobar si las columnas obligatorias existen:

```
MANDATORY_COLUMNS = [
  "UUID",
  "DEPARTAMENTO",
  "PROVINCIA",
  "DISTRITO",
  "METODODX",
  "EDAD",
  "SEXO",
  "FECHA_RESULTADO"
]

for column in MANDATORY_COLUMNS:
    try:
        assert raw_test_df.expect_column_to_exist(column).
    success, f"Uh oh! Mandatory column {column} does not exist:
    FAILED"
        print(f"Column {column} exists : PASSED")
    except AssertionError as e:
        print(e)
==============output==============================
Column UUID exists : PASSED
Column DEPARTAMENTO exists : PASSED
```

```
Column PROVINCIA exists : PASSED
Column DISTRITO exists : PASSED
Column METODODX exists : PASSED
Column EDAD exists : PASSED
Column SEXO exists : PASSED
Column FECHA_RESULTADO exists : PASSED
```

- Las columnas obligatorias no deben ser nulas:

```
for column in MANDATORY_COLUMNS:
    try:
        test_result = raw_test_df.
    expect_column_values_to_not_be_null(column)
        assert test_result.success, \
            f"Uh oh! {test_result.result['unexpected_count']} of
     {test_result.result['element_count']} items in column {
    column} are null: FAILED"
        print(f"All items in column {column} are not null:
    PASSED")
    except AssertionError as e:
        print(e)
    except AnalysisException as e:
        print(e)
==============output================================
All items in column UUID are not null: PASSED
All items in column DEPARTAMENTO are not null: PASSED
All items in column PROVINCIA are not null: PASSED
All items in column DISTRITO are not null: PASSED
All items in column METODODX are not null: PASSED
Uh oh! 20 of 422183 items in column EDAD are null: FAILED
All items in column SEXO are not null: PASSED
All items in column FECHA_RESULTADO are not null: PASSED
```

El test encontró valores nulos en el campo EDAD.

- Comprobar el formato de fecha válido en el campo FECHA_RESULTADO:

```
test_result =  raw_test_df.
    expect_column_values_to_match_strftime_format('
    FECHA_RESULTADO','%Y-%m-%d %H:%M:%S')
f"""{round(test_result.result['unexpected_percent'], 2)}% is not
     a valid date time format"""
====================output============================
100.0% is not a valid date time format
```

- Comprobar si los valores son únicos:

```
test_result = raw_test_df.expect_column_values_to_be_unique("
    UUID")
failed_msg = " ".join([f"""Uh oh!""",
```

```
            f"""{test_result.result['unexpected_count']} of {test_result.result['element_count']} items""",
            f"""or {round(test_result.result['unexpected_percent'],2)}% are not unique: FAILED"""])
print(f"""{'Column UUID is unique: PASSED' if test_result.success else failed_msg}""")
====================output============================
Column UUID is unique: PASSED
```

Hasta este punto puede que los datos incluso tengan algunas inconsistencias por lo tanto será necesario realizar un filtrado de datos.

4.1.4 Filtrado de datos

- El filtrado se debe realizar de acuerdo a las reglas de negocio, osea filtrar datos que estén relacionados a lo que queremos analizar o visualizar. Por ejemplo, solo utilizaremos los de el departamento del Cusco:

```
PRINC_DEPARTAMENTOS = ['CUSCO']
SIN_PROVINCIA_CUSCO=['EN INVESTIGACIO'N']
```

- Aplicar transformaciones:

```
filtered_df = spark.sql(f"""
    SELECT *
    FROM Casos
          WHERE DEPARTAMENTO IN ('{"','".join(PRINC_DEPARTAMENTOS)}')
          AND PROVINCIA NOT IN ('{"','".join(SIN_PROVINCIA_CUSCO)}')
    """)
```

- Crear vista de los datos filtrados:

```
filtered_df.createOrReplaceTempView("FILTERED_CASOS")
```

- Explorar los datos filtrados:

```
filtered_df.toPandas()
```

UUID	DEPARTAMENTO	PROVINCIA	DISTRITO	METODODX	EDAD	SEXO	FECHA_RESULTADO
2e0ce	CUSCO	LA CONVENCION	PICHARI	PCR	24	MASCULINO	20200716
2a3f3	CUSCO	CUSCO	CUSCO	PCR	49	MASCULINO	20200705
1c39d	CUSCO	CUSCO	CUSCO	PCR	65	FEMENINO	20200705
6227d	CUSCO	CUSCO	SANTIAGO	PCR	54	MASCULINO	20200705
decdd	CUSCO	CUSCO	CUSCO	PCR	33	MASCULINO	20200705

- Crear una instancia de los datos filtrados: Utilizar el dataset filtrado para test unitario con SparkDFDataset

```
filtered_test_df = SparkDFDataset(filtered_df)
```

4.1.5 Test Unitario de los datos filtrados

- Verificar que solo existe datos de Cusco:

```
test_result = filtered_test_df.expect_column_values_to_be_in_set
    ("DEPARTAMENTO", PRINC_DEPARTAMENTOS)
print(f"""DEPARTAMENTO are within scope: {'PASSED' if
    test_result.success else 'FAILED'}""")
========================output=========================
DEPARTAMENTO are within scope: PASSED
```

- Verificar la presencia de los campos necesarios (columnas) y que no sean nulos:

```
for column in MANDATORY_COLUMNS:
    try:
        test_result = filtered_test_df.
    expect_column_values_to_not_be_null(column)
        assert test_result.success, f"Uh oh! {test_result.result
    ['unexpected_count']} of {test_result.result['element_count
    ']} items in column {column} are null: FAILED"
        print(f"All items in column {column} are not null:
    PASSED")
    except AssertionError as e:
        print(e)
========================output=========================
All items in column UUID are not null: PASSED
All items in column DEPARTAMENTO are not null: PASSED
All items in column PROVINCIA are not null: PASSED
All items in column DISTRITO are not null: PASSED
All items in column METODODX are not null: PASSED
All items in column EDAD are not null: PASSED
All items in column SEXO are not null: PASSED
All items in column FECHA_RESULTADO are not null: PASSED
```

En este punto a diferencia del anterior test unitario los datos deben pasar cada una de las pruebas.

4.1.6 Estandarización de los datos

- Crear reglas para la estandarización (estandarizar las provincias del Cusco):

```
PROVINCIAS_CUSCO = ['LA CONVENCION',
 'CUSCO',
 'ACOMAYO',
 'QUISPICANCHI',
```

```
 'URUBAMBA',
 'ANTA',
 'CALCA',
 'PARURO',
 'CANCHIS',
 'CHUMBIVILCAS',
 'PAUCARTAMBO',
 'ESPINAR',
 'CANAS']
```

- Transformación de datos y creación de la vista STANDARDISED_CASOS:

```
standardised_df = spark.sql(f"""
    SELECT t.*,
         CASE
             WHEN PROVINCIA IN ('LA CONVENCION','LA CONVENCIO'N
   ') THEN 'LA CONVENCION'
              WHEN PROVINCIA IN ('CUSCO','CUZCO') THEN 'CUSCO'
              WHEN PROVINCIA IN ('URUBAMBA') THEN 'URUBAMBA'
              WHEN PROVINCIA IN ('QUISPICANCHI') THEN '
   QUISPICANCHI'
              WHEN PROVINCIA IN ('PAUCARTAMBO') THEN '
   PAUCARTAMBO'
              WHEN PROVINCIA IN ('PARURO') THEN 'PARURO'
              WHEN PROVINCIA IN ('ESPINAR') THEN 'ESPINAR'
              WHEN PROVINCIA IN ('CANCHIS') THEN 'CANCHIS'
              WHEN PROVINCIA IN ('CANAS') THEN 'CANAS'
              WHEN PROVINCIA IN ('CALCA') THEN 'CALCA'
              WHEN PROVINCIA IN ('ANTA') THEN 'ANTA'
              WHEN PROVINCIA IN ('ACOMAYO') THEN 'ACOMAYO'
             END PROVINCIAS
    FROM FILTERED_CASOS t
""")
```

```
standardised_df.createOrReplaceTempView("STANDARDISED_CASOS")
```

- Exploración de datos:

```
standardised_df.toPandas()
```

UUID	DEPARTAMENTO	PROVINCIA	DISTRITO	METODODX	EDAD	SEXO	FECHA_RESULTADO	PROVINCIAS
2e0ce	CUSCO	LA CONVENCION	PICHARI	PCR	24	MASCULINO	20200716	LA CONVENCION
32a3f3	CUSCO	CUSCO	CUSCO	PCR	49	MASCULINO	20200705	CUSCO
1c39d	CUSCO	CUSCO	CUSCO	PCR	65	FEMENINO	20200705	CUSCO
6227d	CUSCO	CUSCO	SANTIAGO	PCR	54	MASCULINO	20200705	CUSCO
decdd	CUSCO	CUSCO	CUSCO	PCR	33	MASCULINO	20200705	CUSCO
...	...	...	...	...	...	...	...	...
2614c	CUSCO	CUSCO	CUSCO	PR	44	FEMENINO	20200729	CUSCO
b2956	CUSCO	LA CONVENCION	SANTA ANA	PR	48	FEMENINO	20200717	LA CONVENCION
24b4c	CUSCO	CUSCO	WANCHAQ	PR	36	FEMENINO	20200724	CUSCO
e9da4	CUSCO	LA CONVENCION	MEGANTONI	PR	21	FEMENINO	20200725	LA CONVENCION
156c9	CUSCO	CUSCO	WANCHAQ	PR	26	FEMENINO	20200727	CUSCO

- Crear una instancia de los datos estandarizados:

```
standardised_test_df = SparkDFDataset(standardised_df)
```

4.1.7 Test Unitario de los datos estandarizados

- Verificar si existen todas las provincias del Cusco:

```
test_result = standardised_test_df.
    expect_column_values_to_be_in_set("PROVINCIA",
    PROVINCIAS_CUSCO)

print(f"""PROVINCIA are within scope: {'PASSED' if test_result.
    success else 'FAILED'}""")
========================output============================
PROVINCIA are within scope: PASSED
```

- Comprobar si la población es igual al conjunto de datos anterior (los datos filtrados con los estandarizados):

```
filtered_total_rows = filtered_test_df.get_row_count()
test_result = standardised_test_df.
    expect_table_row_count_to_equal(filtered_total_rows)

print(f"""Total row count of standardised_df ({test_result.
    result['observed_value']}) \
is equal to total row count of filtered_df ({filtered_total_rows
    }): \
{'PASSED' if test_result.result['observed_value'] ==
    filtered_total_rows else 'FAILED'}""")
========================output============================
Total row count of standardised_df (4781) is equal to total row
    count of filtered_df (4781): PASSED
```

Se espera que tengan el mismo número de columnas

4.1.8 Producir las métricas finales

- Definir las métricas para cada restricción establecida, por ejemplo para agrupar por departamento, fecha, provincia y contar:
- Aplicar transformaciones y crear una vista con el nombre SUCCESSFUL_CASOS:

```
successful_casos_df = spark.sql(f"""
    SELECT
           FECHA_RESULTADO,
           PROVINCIA,
           count(PROVINCIA) total_successful_casos
    FROM STANDARDISED_CASOS
    GROUP BY DEPARTAMENTO,
           FECHA_RESULTADO,
           PROVINCIA
    ORDER BY FECHA_RESULTADO,
           PROVINCIA
    """)
    successful_campaigns_df.createOrReplaceTempView("
    SUCCESSFUL_CASOS")
```

- Explorar datos:

```
successful_casos_df.toPandas()
```

- Crear una instancia de los datos evaluados satisfactoriamente:

```
successful_casos_df_test_df = SparkDFDataset(successful_casos_df
    )
```

4.1.9 Test unitario sobre las métricas finales

- Comprobar que los valores sean únicos, para una fecha y una provincia:

```
test_result = successful_casos_df_test_df.
    expect_compound_columns_to_be_unique(["FECHA_RESULTADO","
    PROVINCIA"])

print(f"""FECHA_RESULTADO column is unique for each PROVINCIA:
    {'PASSED' if test_result.success else 'FAILED'}""")
==========================ouput=========================
FECHA_RESULTADO column is unique for each PROVINCIA: PASSED
```

- Comprobar la suma total de registros de las métricas es igual al dataset estandarizado:

```
standardised_total_rows = standardised_test_df.get_row_count()
test_result = successful_casos_df_test_df.
    expect_column_sum_to_be_between('total_successful_casos',
```

```
                    standardised_total_rows,
    standardised_total_rows)

print(f"""Total sum of casos in metrics dataset ({test_result.
    result['observed_value']}) \
is equal to total rows in standardised dataset ({
    standardised_total_rows}): \
{'PASSED' if test_result.result['observed_value'] ==
    standardised_total_rows else 'FAILED'}""")
==========================output=============================
Total sum of casos in metrics dataset (4781) is equal to total
    rows in standardised dataset (4781): PASSED
```

4.1.10 Resumen de todas las validaciones

Juntar todas las validaciones por etapas.

- Test integral de todos los datasets de cada etapa:

```
raw_test_df_validation = raw_test_df.validate()
print(f"""1. Raw dataset validations: {raw_test_df_validation.
    success}; {raw_test_df_validation.statistics['success_percent
    ']} successful""")
filtered_test_df_validation = filtered_test_df.validate()
print(f"""2. Filtered dataset validations: {
    filtered_test_df_validation.success}; {
    filtered_test_df_validation.statistics['success_percent']}
    successful""")
standardised_test_df_validation = standardised_test_df.validate
    ()
print(f"""3. Standardised dataset validations: {
    standardised_test_df_validation.success}; {
    standardised_test_df_validation.statistics['success_percent
    ']} successful""")
successful_casos_df_test_df_validation =
    successful_casos_df_test_df.validate()
print(f"""4. Metrics dataset validations: {
    successful_casos_df_test_df_validation.success}; {
    successful_casos_df_test_df_validation.statistics['
    success_percent']} successful""")
```

```
=====================output================================
1. Raw dataset validations: False; 88.88888888888889 successful
2. Filtered dataset validations: True; 100.0 successful
3. Standardised dataset validations: True; 100.0 successful
4. Metrics dataset validations: True; 100.0 successful
```

4.1.11 Analisis y visualización del dataset COVID-19

1. **Visualizar los datos:** En esta ultima etapa son generados diversos gráficos con los datos analizados y procesados.

```
plt.figure(figsize = (30, 8))
def provincia_plot(name):
    data = df_group_count.loc[name]
    plt.plot(data.index, data.values)
names = [x.PROVINCIA for x in filtered.select("PROVINCIA").
   distinct().collect()]

for name in names:
    provincia_plot(name)
plt.legend(names)
plt.title('Provincias Cusco infectados por fecha')
plt.grid(True)
```

A continuación se muestra el análisis de los datos correspondientes a casos positivos de COVID-19 en el departamento del Cusco en 31 de julio de 2020 (Figuras 4.1 y 4.2).

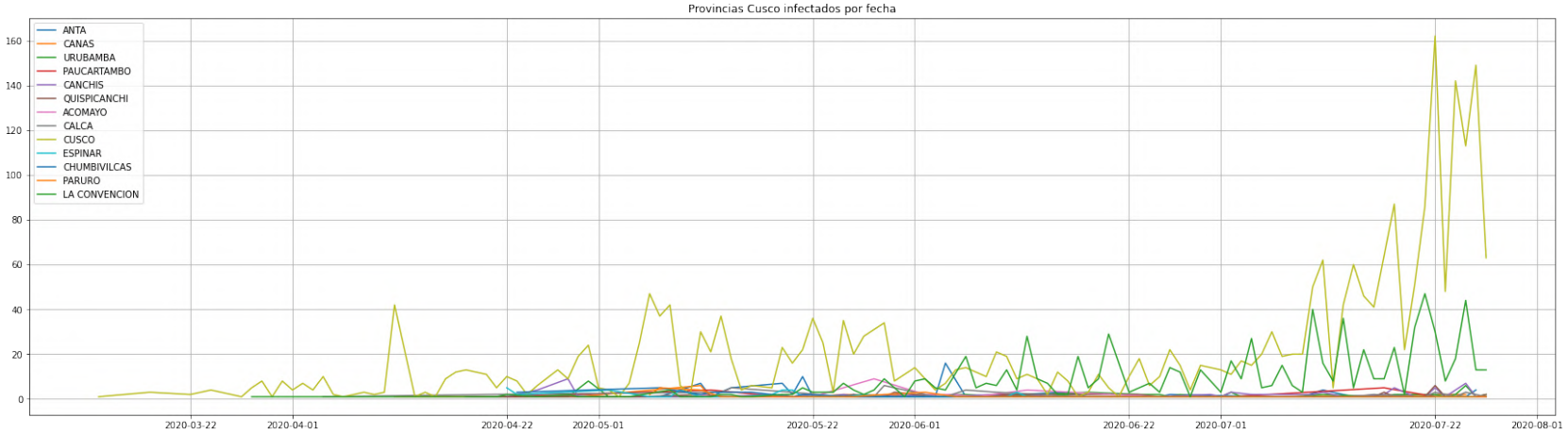


Figura 4.1: *Casos positivos de COVID-19 en el departamento de Cusco en 22 de julio de 2020*
Fuente: *Propia*

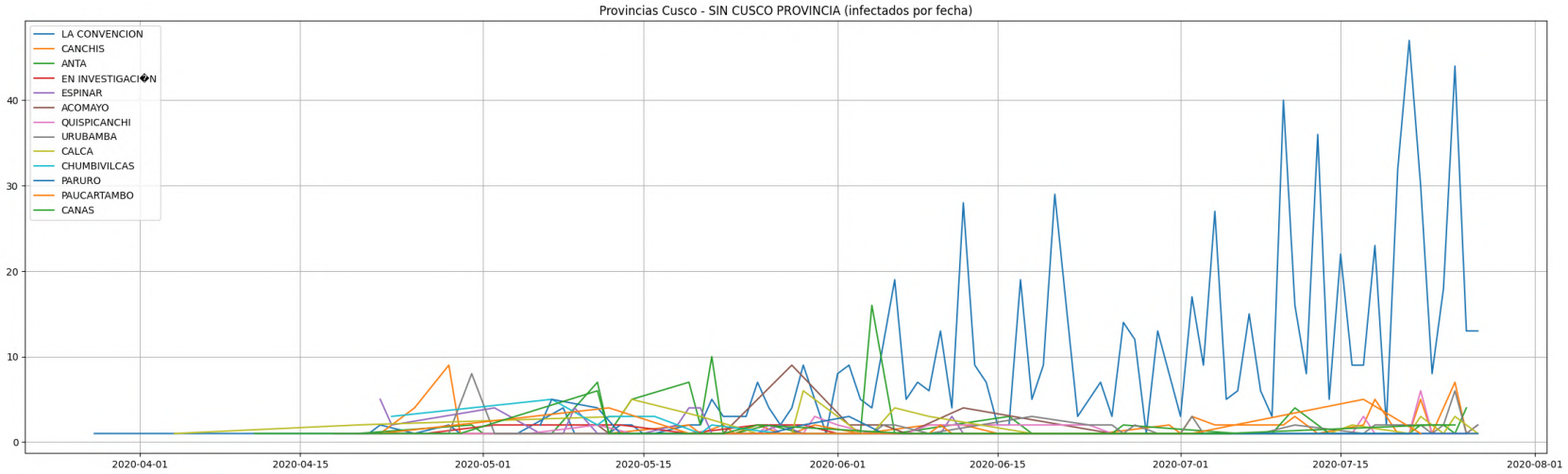


Figura 4.2: *Casos positivos de COVID-19 en el departamento de Cusco, sin considerar Cusco como provincia en 22 de julio de 2020*
Fuente: *Propia*

2. **Uso de técnicas de preprocesamiento:** En esta dataset de COVID-19 se realizaron pruebas unitarias e integral, seguidamente se obtuvo datos limpios para realizar un analisis y visualización.

4.2 Caso de Uso 2: Dataset sismos

En este caso de uso se muestra el análisis de Big Data tomando como fuente *dataset* de actividad sísmica, utilizado en proyectos de geo-ciencia, se utiliza la herramienta Bokeh para visualización de los datos y el proyecto fue implementado en Jupyter Notebook.

4.2.1 Dataset de sismos

El *dataset* contiene una cantidad considerable de datos desde 1965 hasta el 2016, cuenta con 23412 registros y 21 columnas de FECHA, LATITUD, LONGITUD, PROFUNDIDAD EN KILÓMETROS, MAGNITUD y otros datos[2].

Esta dataset se descargo del repositorio de Kaggle. Este trabajo se centra en realizar un preprocesamiento de datos para ello se usan ciertas técnicas, como por ejemplo, extracción, carga y transformación de datos (ETL), seleción de instancias y tratamiento de datos imperfectos (Sección 2.5.3, página 24).

En este dataset también se realizaron pruebas unitarias, por cuestiones de simplicidad se mostrará la comparación de la etapa de los datos sin procesar y después de filtrar los datos, se validó el campo fecha (Date) y se verificó que los campos no sean nulos ni duplicados:

```
raw_test_df_validation = raw_test_df.validate()
print(f"""1. Raw dataset validations: {raw_test_df_validation.
    success}; {raw_test_df_validation.statistics['success_percent']}
    successful""")
filtered_test_df_validation = filtered_test_df.validate()
print(f"""2. Filtered dataset validations: {
    filtered_test_df_validation.success}; {
    filtered_test_df_validation.statistics['success_percent']}
    successful""")
==========================ouput=========================
1. Raw dataset validations: False; 93.75 successful
2. Filtered dataset validations: True; 100.0 successful
```

[2]Dataset de actividad sísmica mayor igual a 5 grados, de todo el mundo (1965-2016) https://github.com/EBISYS/WaterWatch/blob/master/database.csv

4.2.2 Preprocesamiento de Datos con Apache Spark

1. **Integración de Pyspark con Jupyter Notebook:** Para está primera tarea ejecutamos el siguiente código para inicializar apache spark.

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *
spark = SparkSession.builder.appName("equake_visualization").
    getOrCreate()
```

2. **Extraer, transformar y cargar los datos (ETL):** Se cargan los datos de un directorio local.

```
#load the dataset
df_load = spark.read.csv('/content/gdrive/MyDrive/Colab
    Notebooks/database.csv',header=True)
#preview the load
#df_load.take(1)
df_load.toPandas()
```

Después de cargar los datos podemos usar la función *df_load.take*(1) para mostrar una fila:

	Date	Time	Latitude	Longitude	Type	Depth	Depth Error	Depth Seismic Stations	Magnitude	Magnitude Type	...	Magnitude Seismic Stations	Azimuthal Gap	Horizontal Distance	Horizontal Error	Root Mean Square	ID	Source	Location Source	Magnitude Source	Status
0	01/02/1965	13:44:18	19.246	145.616	Earthquake	131.6	None	None	6	MW	...	None	None	None	None	None	ISCGEM860706	ISCGEM	ISCGEM	ISCGEM	Automatic
1	01/04/1965	11:29:49	1.863	127.352	Earthquake	80	None	None	5.8	MW	...	None	None	None	None	None	ISCGEM860737	ISCGEM	ISCGEM	ISCGEM	Automatic
2	01/05/1965	18:05:58	-20.579	-173.972	Earthquake	20	None	None	6.2	MW	...	None	None	None	None	None	ISCGEM860762	ISCGEM	ISCGEM	ISCGEM	Automatic
3	01/08/1965	18:49:43	-59.076	-23.557	Earthquake	15	None	None	5.8	MW	...	None	None	None	None	None	ISCGEM860856	ISCGEM	ISCGEM	ISCGEM	Automatic
4	01/09/1965	13:32:50	11.938	126.427	Earthquake	15	None	None	5.8	MW	...	None	None	None	None	None	ISCGEM860890	ISCGEM	ISCGEM	ISCGEM	Automatic
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23407	12/28/2016	08:22:12	38.3917	-118.8941	Earthquake	12.3	1.2	40	5.6	ML	...	18	42.47	0.12	None	0.1898	NN00570710	NN	NN	NN	Reviewed
23408	12/28/2016	09:13:47	38.3777	-118.8957	Earthquake	8.8	2	33	5.5	ML	...	18	48.58	0.129	None	0.2187	NN00570744	NN	NN	NN	Reviewed
23409	12/28/2016	12:38:51	36.9179	140.4262	Earthquake	10	1.8	None	5.9	MWW	...	None	91	0.992	4.8	1.52	US10007NAF	US	US	US	Reviewed
23410	12/29/2016	22:30:19	-9.0283	118.6639	Earthquake	79	1.8	None	6.3	MWW	...	None	26	3.553	6	1.43	US10007NL0	US	US	US	Reviewed
23411	12/30/2016	20:08:28	37.3973	141.4103	Earthquake	11.94	2.2	None	5.5	MB	...	428	97	0.681	4.5	0.91	US10007NTD	US	US	US	Reviewed

23412 rows × 21 columns

En el dataset existen varias columnas que no necesitaremos, para lo cual procedemos a eliminar algunas con la función df_load.drop.

```
lst_dropped_columns = ['Time','Depth Error','Depth Seismic
    Stations','Magnitude Error','Magnitude Seismic Stations',
                       'Azimuthal Gap','Horizontal Distance','
    Horizontal Error','Root Mean Square','Source','Location
    Source','Magnitude Source','Status']
df_load = df_load.drop(*lst_dropped_columns)
# Preview df_load
# df_load.show(5)
df_load.toPandas()
```

Date	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	ID
01/02/1965	19.246	145.616	Earthquake	131.6	6	MW	ISCGEM860706
01/04/1965	1.863	127.352	Earthquake	80	5.8	MW	ISCGEM860737
01/05/1965	-20.579	-173.972	Earthquake	20	6.2	MW	ISCGEM860762
01/08/1965	-59.076	-23.557	Earthquake	15	5.8	MW	ISCGEM860856
01/09/1965	11.938	126.427	Earthquake	15	5.8	MW	ISCGEM860890
...	...	...	...	...	...	...	...
12/28/2016	38.3917	-118.8941	Earthquake	12.3	5.6	ML	NN00570710
12/28/2016	38.3777	-118.8957	Earthquake	8.8	5.5	ML	NN00570744
12/28/2016	36.9179	140.4262	Earthquake	10	5.9	MWW	US10007NAF
12/29/2016	-9.0283	118.6639	Earthquake	79	6.3	MWW	US10007NL0
12/30/2016	37.3973	141.4103	Earthquake	11.94	5.5	MB	US10007NTD

Con este procedimiento los datos quedaran más limpios (ver Sección 2.5.3, página 24).
Luego procedemos a dar un formato más manejable al campo fecha con la función to_timestamp

```
df_load = df_load.withColumn('Year',year(to_timestamp('Date','dd
    /MM/yyyy')))
df_load.show(5)
```

Date	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	ID	Year
01/02/1965	19.246	145.616	Earthquake	131.6	6	MW	ISCGEM860706	1965.0
01/04/1965	1.863	127.352	Earthquake	80	5.8	MW	ISCGEM860737	1965.0
01/05/1965	-20.579	-173.972	Earthquake	20	6.2	MW	ISCGEM860762	1965.0
01/08/1965	-59.076	-23.557	Earthquake	15	5.8	MW	ISCGEM860856	1965.0
01/09/1965	11.938	126.427	Earthquake	15	5.8	MW	ISCGEM860890	1965.0

Seguidamente se cuenta las actividades sísmicas por año.

```
df_quake_freq = df_load.groupBy('Year').count().
    withColumnRenamed('count','Counts')
df_quake_freq.show(5)
```

Year	Counts
1990.0	196
1975.0	150
1977.0	148
2003.0	187
2007.0	211
1974.0	147
2015.0	175
2006.0	176
1978.0	158
2013.0	202

Verificar los tipos de datos del dataset para organizar de mejor forma.

```
df_load.printSchema()
```

```
root
 |-- Date: string (nullable = true)
 |-- Latitude: double (nullable = true)
 |-- Longitude: double (nullable = true)
 |-- Type: string (nullable = true)
 |-- Depth: double (nullable = true)
 |-- Magnitude: double (nullable = true)
 |-- Magnitude Type: string (nullable = true)
 |-- ID: string (nullable = true)
 |-- Year: integer (nullable = true)
```

Utilizamos la función *cast* para cambiar de tipo de dato a los campos pertinentes

```
# cast some fields from string into numeric types
df_load = df_load.withColumn('Latitude',df_load['Latitude'].cast
    (DoubleType()))\
     .withColumn('Longitude',df_load['Longitude'].cast(DoubleType
    ()))\
     .withColumn('Depth',df_load['Depth'].cast(DoubleType()))\
     .withColumn('Magnitude',df_load['Magnitude'].cast(DoubleType
    ()))
df_load.show(5)
```

Date	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	ID	Year
01/02/1965	19.2460	145.6160	Earthquake	131.60	6.0	MW	ISCGEM860706	1965.0
01/04/1965	1.8630	127.3520	Earthquake	80.00	5.8	MW	ISCGEM860737	1965.0
01/05/1965	-20.5790	-173.9720	Earthquake	20.00	6.2	MW	ISCGEM860762	1965.0
01/08/1965	-59.0760	-23.5570	Earthquake	15.00	5.8	MW	ISCGEM860856	1965.0
01/09/1965	11.9380	126.4270	Earthquake	15.00	5.8	MW	ISCGEM860890	1965.0

3. **Análisis de datos:** Para el análisis se calcula el valor máximo y promedio por año dentro de un nuevo *dataframe* df_quake_freq

```
# create avg and max magnitude fields and add to df_quake_freq
df_max = df_load.groupBy('Year').max('Magnitude').
    withColumnRenamed('max(Magnitude)','Max_Magnitude')
df_avg = df_load.groupBy('Year').avg('Magnitude').
    withColumnRenamed('avg(Magnitude)','Avg_Magnitude')
df_quake_freq = df_quake_freq.join(df_avg,['Year']).join(df_max
    ,['Year'])
df_quake_freq = df_quake_freq.orderBy(asc('Year'))
df_quake_freq.show(5)
```

Year	Counts	Avg_Magnitude	Max_Magnitude
1965	156	6.009615	8.7
1966	98	6.060714	7.7
1967	103	5.962136	7.2
1968	106	6.070755	7.6
1969	114	6.015789	7.5
1970	142	6.019718	7.4
1971	148	5.986486	7.8
1972	178	5.931461	8.0
1973	144	5.833611	7.4
1974	147	5.890476	7.6
1975	150	5.848667	7.8

En los *dataframes* es necesario borrar los valores nulos con la función *dropna*

```
df_load.dropna()
df_quake_freq.dropna()
```

4. **Visualizar los datos:** Se realizo visualización de datos en barras y en curvas (Figura 4.3).

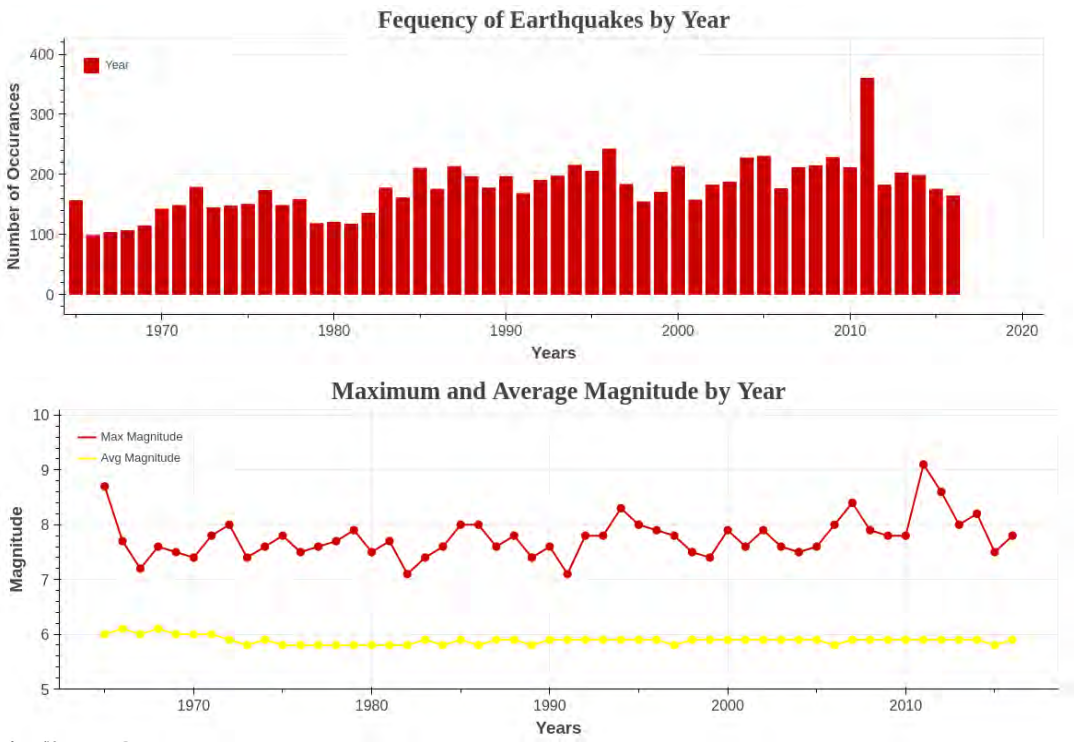


Figura 4.3: *Actividad sísmica en el mundo desde 1965-2016*
Fuente: *Propia*

4.3 Caso de uso 3: Dataset Diabetes

En este caso de uso se tiene una dataset de personas con y sin diabetes, para el entrenamiento se hace uso de herramientas de preprocesamiento de datos o técnicas que predicen la aparición de diabetes basándose en medidas de diagnóstico.

4.3.1 Dataset Diabetes

Los conjuntos de datos constan de varias variables predictoras médicas y una variable objetivo (Outcome). Las variables predictoras incluyen el NÚMERO DE EMBARAZOS que tuvieron los pacientes, IMC, NIVEL DE INSULINA, EDAD, etc.

Este conjunto de datos proviene del Instituto Nacional de Diabetes y Enfermedades Digestivas y Renales. El objetivo del conjunto de datos es predecir de forma diagnóstica si un paciente tiene diabetes o no, basándose en determinadas medidas de diagnóstico incluidas en el conjunto de datos. Se impusieron varias restricciones a la selección de estas instancias de una base de datos más grande. En particular, todos los pacientes son mujeres de al menos 21 años de edad de origen indio Pima (Smith et al., 1988).

Esta dataset se descargo del repositorio de Kaggle [3]. Este trabajo se centra en realizar un preprocesamiento de datos para ello se usan las técnicas de preprocesamiento, como por ejemplo, transformación de datos (OneHotEncoder y StringIndexer).

4.3.2 Preprocesamiento de Datos con Apache Spark

1. **Inicialización de Apache Spark:** Se importa una librería de Pyspark para iniciar sesión y se pasan los parámetros de configuración, como nombre de la aplicación, recurso computacional a usar, por ejemplo memoria RAM de 1GB.

```
# Import SparkSession
from pyspark.sql import SparkSession

# Build the SparkSession
spark = SparkSession.builder \
   .master("local") \
   .appName("Diabetes prediction") \
   .config("spark.executor.memory", "1gb") \
   .getOrCreate()

sc = spark.sparkContext
```

2. **Extraer, transformar y cargar los datos (ETL):** Se tiene la dataset en formato .csv para este caso se esta trabajando en googlecolab para ello se sube el documento en el drive para luego leer.

```
#Pima Indians Diabetes Database
#Predict the onset of diabetes based on diagnostic measures
#UCI Machine Learning
#The Applied options are for CSV files
df = spark.read.format("csv") \
     .option("inferSchema","true") \
     .option("header","true") \
     .option("sep",",") \
     .load(folder+"/ds_diabetes.csv")
```

3. **Analizar los datos:** Se muestra los 5 primeros datos de la dataset con sus respectivos campos; pregnancies, glucose, bloodpressure, skinthickness, insulin, bmi, diabetespedigreefunction,age, outcome.

[3]UCI Machine Learning, Pima Indians Diabetes Database, "Predict the onset of diabetes based on diagnostic measures", url: https://www.kaggle.com/uciml/pima-indians-diabetes-database

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1

En esta sección lo que se hace es mostrar los tipos de datos que corresponde a cada campo de nuestra dataset.

```
from collections import defaultdict
data_types = defaultdict(list)
for entry in df.schema.fields:
  data_types[str(entry.dataType)].append(entry.name)
data_types
```

```
defaultdict(list,
            {'IntegerType()': ['Pregnancies',
              'Glucose',
              'BloodPressure',
              'SkinThickness',
              'Insulin',
              'Age',
              'Outcome'],
             'DoubleType()': ['BMI', 'DiabetesPedigreeFunction']})
```

4. **Visualización de los datos:** Con el uso de las librerías de pandas y matplot es posible visualizar los gráficos.

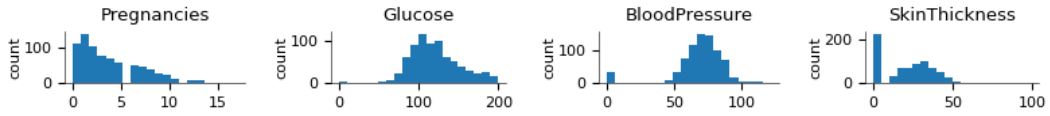


Figura 4.4: *Distribución de los campos Pregnancies, Glucose, BloodPressure y SkinThickness del dataset diabetes*
Fuente: *Propia*

5. **Uso de técnicas de preprocesamiento:** Se utiliza dos funciones OneHotEncoder (Sección 2.5.2, página 24) y StringIndexer, estas son técnicas de transformación para convertir atributos categóricos en un vector con dos componentes. Este preprocesamiento nos permite acelerar el procesamiento de los valores numéricos, como se muestra en la Sección 2.5.1 (página 21)

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer

```

```
strings_used = ["Pregnancies"]

stage_string = [StringIndexer(inputCol= c, outputCol = c+"
    _string_encoded") for c in strings_used]
stage_one_hot = [OneHotEncoder(inputCol= c+"_string_encoded",
    outputCol= c+ "_one_hot") for c in strings_used]
ppl = Pipeline(stages= stage_string + stage_one_hot)
df = ppl.fit(df).transform(df)
```

BMI	DiabetesPedigreeFunction	Age	Outcome	Pregnancies_string_encoded	Pregnancies_one_hot
33.6	0.627	50	1	6.0	(16,[6],[1.0])
26.6	0.351	31	0	0.0	(16,[0],[1.0])
23.3	0.672	32	1	8.0	(16,[8],[1.0])
28.1	0.167	21	0	0.0	(16,[0],[1.0])
43.1	2.288	33	1	1.0	(16,[1],[1.0])
25.6	0.201	30	0	5.0	(16,[5],[1.0])
31.0	0.248	26	1	3.0	(16,[3],[1.0])
35.3	0.134	29	0	10.0	(16,[10],[1.0])
30.5	0.158	53	1	2.0	(16,[2],[1.0])

Uso de la función vectorAssembler para vectorizar

```
#Transformer: VectorAssembler
from pyspark.ml.feature import VectorAssembler
features = ['Pregnancies_one_hot','Glucose','BloodPressure',
'SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age'
    ]
vector_assembler = VectorAssembler(inputCols = features,
    outputCol= "features")
data_training_and_test = vector_assembler.transform(df)
```

DiabetesPedigreeFunction	Age	Outcome	Pregnancies_string_encoded	Pregnancies_one_hot	features
0.627	50	1	6.0	(16,[6],[1.0])	(23,[6,16,17,18,20,21,22],[1.0,148.0,72.0,35.0,33.6,0.627,50.0])
0.351	31	0	0.0	(16,[0],[1.0])	(23,[0,16,17,18,20,21,22],[1.0,85.0,66.0,29.0,26.6,0.351,31.0])
0.672	32	1	8.0	(16,[8],[1.0])	(23,[8,16,17,20,21,22],[1.0,183.0,64.0,23.3,0.672,32.0])
0.167	21	0	0.0	(16,[0],[1.0])	(23,[0,16,17,18,19,20,21,22],[1.0,89.0,66.0,23.0,94.0,28.1,0.167,21.0])
2.288	33	1	1.0	(16,[1],[1.0])	(23,[1,16,17,18,19,20,21,22],[1.0,137.0,40.0,35.0,168.0,43.1,2.288,33.0])
0.201	30	0	5.0	(16,[5],[1.0])	(23,[5,16,17,20,21,22],[1.0,116.0,74.0,25.6,0.201,30.0])
0.248	26	1	3.0	(16,[3],[1.0])	(23,[3,16,17,18,19,20,21,22],[1.0,78.0,50.0,32.0,88.0,31.0,0.248,26.0])
0.134	29	0	10.0	(16,[10],[1.0])	(23,[10,16,20,21,22],[1.0,115.0,35.3,0.134,29.0])
0.158	53	1	2.0	(16,[2],[1.0])	(23,[2,16,17,18,19,20,21,22],[1.0,197.0,70.0,45.0,543.0,30.5,0.158,53.0])
0.232	54	1	8.0	(16,[8],[1.0])	(23,[8,16,17,21,22],[1.0,125.0,96.0,0.232,54.0])
0.191	30	0	4.0	(16,[4],[1.0])	(23,[4,16,17,20,21,22],[1.0,110.0,92.0,37.6,0.191,30.0])
0.537	34	1	10.0	(16,[10],[1.0])	(23,[10,16,17,20,21,22],[1.0,168.0,74.0,38.0,0.537,34.0])
1.441	57	0	10.0	(16,[10],[1.0])	(23,[10,16,17,20,21,22],[1.0,139.0,80.0,27.1,1.441,57.0])
0.398	59	1	0.0	(16,[0],[1.0])	(23,[0,16,17,18,19,20,21,22],[1.0,189.0,60.0,23.0,846.0,30.1,0.398,59.0])
0.587	51	1	5.0	(16,[5],[1.0])	(23,[5,16,17,18,19,20,21,22],[1.0,166.0,72.0,19.0,175.0,25.8,0.587,51.0])

6. **Entrenamiento del modelo de aprendizaje automático:** En esta sección se empieza con el entrenamiento del algoritmo de aprendizaje automático, que corresponde a una clasificación binaria. No es el objetivo de la investigación realizar el aprendizaje automático, sin embargo, nos permite mostrar la necesidad del preprocesamiento y la calidad de datos.

```
#Etapa de entrenamiento
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
(training_data, test_data) = data_training_and_test.randomSplit
    ([0.7, 0.3], 2017)
```

```
rf = RandomForestClassifier(labelCol = "Outcome",
                        featuresCol = "features", numTrees = 20)
rf_model = rf.fit(training_data)
predictions = rf_model.transform(test_data)
evaluator= BinaryClassificationEvaluator(labelCol = "Outcome",
   rawPredictionCol="probability", metricName= "areaUnderROC")
accuracy = evaluator.evaluate(predictions)
print("Accuracy:",accuracy*100)
========================output============================
Accuracy: 80.80255828347421
```

Los métodos de preprocesamiento utilizados beneficiaron para obtener un aceptable porcentaje de precisión en este experimento, de lo contrario el algoritmo calcularía un porcentaje de precisión poco realista.

Capítulo V

Análisis de datos

5 Análisis de datos

Análisis de las técnicas de preprocesamiento más usadas en cada caso de uso del Capítulo 4.

5.1 Dataset de COVID-19

Para el caso de covid se tiene la siguiente imagen donde se puede observar la limpieza de datos que se realiza (Figura 5.1).

UUID	DEPARTAMENTO	PROVINCIA	DISTRITO	METODODX	EDAD	SEXO	FECHA_RESULTADO
7320cabdc1aaca6c5...	LIMA	LIMA	LOS OLIVOS	PCR	55	MASCULINO	2020-05-26
cecdbf10074dbc011...	LIMA	LIMA	SAN JUAN DE LURIG...	PCR	34	MASCULINO	2020-05-21
71ecb6bccb248b0bb...	LIMA	LIMA	COMAS	PCR	42	FEMENINO	2020-05-28
566af4276cbe9359a...	LIMA	LIMA	SAN MIGUEL	PCR	55	MASCULINO	2020-05-12
f016889b9ba5bd95c...	LIMA	LIMA	EL AGUSTINO	PCR	48	FEMENINO	2020-04-06
971f8e1295583756d...	LIMA	LIMA	BRE�A	PCR	59	MASCULINO	2020-04-20
0e2a1928ddd07d999...	CALLAO	CALLAO	BELLAVISTA	PCR	29	FEMENINO	2020-05-07
7d943c682f6a35d8c...	LIMA	LIMA	LIMA	PCR	33	FEMENINO	2020-04-09
fcdc23719caee45f2...	LIMA	LIMA	VILLA MARIA DEL T...	PCR	40	MASCULINO	2020-05-15
e320d0efac97be16e...	LIMA	LIMA	LIMA	PCR	70	MASCULINO	2020-05-28
3ffa9b891550ebfb4...	LIMA	LIMA	LIMA	PCR	45	FEMENINO	2020-05-16
6ae8311b8bd08b622...	LIMA	LIMA	SAN MARTIN DE PORRES	PCR	31	FEMENINO	2020-05-19
00b1af8adc3f43c22...	LIMA	LIMA	SAN JUAN DE LURIG...	PCR	62	FEMENINO	2020-06-02
fb2fe4181c0bec769...	LIMA	LIMA	CHORRILLOS	PCR	36	MASCULINO	2020-05-09
a4d48cfda78e6d746...	LIMA	LIMA	JESUS MARIA	PCR	38	MASCULINO	2020-05-12
c1f091ccef4b8a7d2...	LIMA	LIMA	ATE	PCR	61	MASCULINO	2020-05-11
4246e44f68a5776f9...	LIMA	LIMA	RIMAC	PCR	29	MASCULINO	2020-04-21
75573dee492c7ad5c...	LIMA	LIMA	SAN JUAN DE MIRAF...	PCR	30	MASCULINO	2020-05-25
a2be046f90cf99306...	LIMA	LIMA	SANTIAGO DE SURCO	PCR	44	FEMENINO	2020-04-21
ed2b1b555ff87801d...	LIMA	LIMA	LA MOLINA	PCR	30	MASCULINO	2020-04-18

Figura 5.1: *Datos sin procesar de COVID19 en Perú*
Fuente: *Propia*

Antes de preprocesamiento de datos: En esta dataset se tiene errores en el ingreso de los campos de distrito y en el formato de fecha (Figura 5.2).

.

ba40ca6bcbd9f6	29/08/1974		APURIMAC	ANDAHUAYLAS	SAN JERONIMO	2020-04-17	PR
ba40ca6bcbd9f6	1974-08-29	FEMENINO	APURIMAC	ANDAHUAYLAS	SAN JERÓNIMO	2020-05-04	PR
ba40ca6bcbd9f6	1974-08-29	FEMENINO	APURIMAC	ANDAHUAYLAS	SAN JERÓNIMO	2020-04-17	PR

Error en el nombre del distrito

5b4ef7c71e1a414835c533ebee6!		FEMENINO	LIMA	LIMA	SANTA ANITA	2020-04-27	PR
79219debe70ac8d8825adb8db2b		MASCULINO	LIMA	LIMA	COMAS	2020-04-16	PR
6fa1a28244dad:	2020-01-23	FEMENINO	CALLAO	PROV. CONST.	CALLAO	2020-04-23	PR
316b472f71675af2f1056a6c818682e7			LIMA	LIMA	CARABAYLLO	2020-04-11	PR
316b472f71675af2f1056a6c818682e7			LIMA	LIMA	CHACLACAYO	2020-04-08	PR
c38008428e066	1982-07-25	Masculino	CAJAMARCA	JAEN	HUABAL	16/05/2020	PCR
539d534463948	1982-01-18	Femenino	LORETO	MAYNAS	IQUITOS	16/05/2020	PCR
17ccd9ab83766	1986-08-03	Masculino	LORETO	MAYNAS	BELEN	16/05/2020	PCR
fd3edb4230f8a9	1984-09-04	Masculino	LIMA	LIMA	SAN MARTIN DE PORR	14/05/2020	PCR
92b59cdbe9e98	1975-06-03	Femenino	LIMA	LIMA	BREÑA	14/05/2020	PCR

Fechas con diferente formato

Figura 5.2: *Datos sin procesar*
Fuente: *Propia*

Después del preprocesamiento: En esta última tabla nos enfocamos en el Departamento de Cusco para el campo de DISTRITO se hizo las correcciones de algunos campos con valores inconsistentes o mal registrados además se hizo algunas validaciones con los caracteres especiales para que la información este más clara. Para el campo de FECHA RESULTADO se verifico el formato fecha (Figura 5.3).

UUID	DEPARTAMENTO	PROVINCIA	DISTRITO	METODODX	EDAD	SEXO	FECHA_RESULTADO
b47182e3d1ab6b1b1...	CUSCO	CUSCO	CUSCO	PCR	49	MASCULINO	2020-07-05
c1e97f0913a3491c8...	CUSCO	CUSCO	CUSCO	PCR	65	FEMENINO	2020-07-05
699e470bc9ac42ef6...	CUSCO	CUSCO	SANTIAGO	PCR	54	MASCULINO	2020-07-05
a4ce0fc24b8aa540d...	CUSCO	CUSCO	CUSCO	PCR	33	MASCULINO	2020-07-05
19b907a5600f1e5a0...	CUSCO	CUSCO	SANTIAGO	PCR	29	MASCULINO	2020-07-05
6851e22a1667e6cf9...	CUSCO	CUSCO	CUSCO	PCR	33	FEMENINO	2020-07-05
695edd13175eb0ec8...	CUSCO	LA CONVENCION	PICHARI	PCR	24	MASCULINO	2020-07-16
af314be39e97a8a54...	CUSCO	CUSCO	CUSCO	PCR	37	FEMENINO	2020-03-22
06bcbbadd07a6487a...	CUSCO	CUSCO	CUSCO	PCR	29	FEMENINO	2020-06-01
a84121509b864c947...	CUSCO	CUSCO	SANTIAGO	PCR	74	MASCULINO	2020-06-01
d0c07a7bb8c94066c...	CUSCO	CUSCO	CUSCO	PCR	53	FEMENINO	2020-06-01
2b1bd683f4c352ae7...	CUSCO	LA CONVENCION	SANTA ANA	PCR	50	FEMENINO	2020-07-17
68b4e0e2ff25e224c...	CUSCO	LA CONVENCION	SANTA ANA	PCR	28	FEMENINO	2020-07-17
692c7e85de34204f3...	CUSCO	CUSCO	SANTIAGO	PCR	28	MASCULINO	2020-07-05
ccbdf673ce18ddac5...	CUSCO	CUSCO	WANCHAQ	PCR	59	FEMENINO	2020-07-06
6a08f0dcb92ec3fd5...	CUSCO	CUSCO	WANCHAQ	PCR	21	MASCULINO	2020-07-06
8e3d02379ace174b7...	CUSCO	CUSCO	CUSCO	PCR	33	MASCULINO	2020-07-06
ac037fcd635374136...	CUSCO	CUSCO	CUSCO	PCR	60	MASCULINO	2020-07-06
536b4cea8ae27f198...	CUSCO	CUSCO	CUSCO	PCR	30	MASCULINO	2020-04-24
5b8e2328ca042e533...	CUSCO	ACOMAYO	POMACANCHI	PCR	17	FEMENINO	2020-06-05

Figura 5.3: *Datos procesados*
Fuente: *Propia*

Para las pruebas unitarias no se tiene valores nulos, se verifico el formato de fecha y no se tiene valores duplicados por lo tanto se tiene datos limpios.

5.2 Dataset de Sismos.

Para el caso de sismos se tiene la siguiente Figura 5.4 donde se puede observar la limpieza de datos que se realiza y los métodos o técnicas usadas.

Antes de preprocesamiento de datos: En esta dataset se tiene errores en el formato de fecha.

Date	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	ID
01/02/1965	19.246	145.616	Earthquake	131.6	6	MW	ISCGEM860706
01/04/1965	1.863	127.352	Earthquake	80	5.8	MW	ISCGEM860737
01/05/1965	-20.579	-173.972	Earthquake	20	6.2	MW	ISCGEM860762
01/08/1965	-59.076	-23.557	Earthquake	15	5.8	MW	ISCGEM860856
01/09/1965	11.938	126.427	Earthquake	15	5.8	MW	ISCGEM860890

04/26/1985	15:53:19	-17735	168346	Earthquake
04/27/1985	00:33:13	-15798	-173503	Earthquake
04/27/1985	01:31:28	38599	73173	Earthquake
04/27/1985	10:11:43	-21032	-176.82	Earthquake
1985-04-28T02:53:41.530Z	1985-04-28T02:53:41.530Z	-32998	-71766	Earthquake
04/28/1985	08:30:29	-39728	-75664	Earthquake
04/28/1985	22:56:51	-55494	-26149	Earthquake
04/29/1985	02:20:00	41479	142043	Earthquake

Error en la fecha (diferente formato)

Figura 5.4: *Datos sin procesar*
Fuente: *Propia*

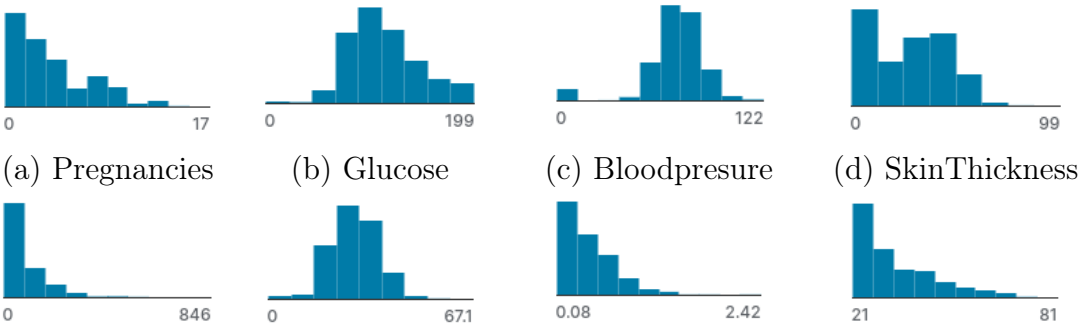
Después del preprocesamiento de datos: En esta última tabla nos enfocamos en limpiar los datos; convertir las cadenas en valores numéricos de los campos de Latitude, Longitude, Depth, y Magnitude (campos con prioridad) y también fue verificado el formato de fecha, creamos otro campo Year para análisis de datos para subsanar los errores de fecha (Figura 5.5).

Date	Latitude	Longitude	Type	Depth	Magnitude	Magnitude Type	ID	Year
01/02/1965	19.246	145.616	Earthquake	131.6	6.0	MW	ISCGEM860706	1965
01/04/1965	1.863	127.352	Earthquake	80.0	5.8	MW	ISCGEM860737	1965
01/05/1965	-20.579	-173.972	Earthquake	20.0	6.2	MW	ISCGEM860762	1965
01/08/1965	-59.076	-23.557	Earthquake	15.0	5.8	MW	ISCGEM860856	1965
01/09/1965	11.938	126.427	Earthquake	15.0	5.8	MW	ISCGEM860890	1965

Figura 5.5: *Datos procesados*
Fuente: *Propia*

5.3 Dataset de Diabetes

En la Figura 5.6 se muestra la distribución de los valores de cada característica.



(a) Pregnancies (b) Glucose (c) Bloodpresure (d) SkinThickness

(e) Insulin (f) BMI (g) DiabetesPedigree (h) Age

Figura 5.6: *Distribución de valores de las características del dataset de diabetes*
Fuente: *Propia*

Antes del preprocesamiento de datos (Figura 5.7), el campo **Pregnancies** (Figura 5.6a) presenta un problema de categorización, por esa razón fue utilizado la técnica de preprocesamiento OneHotEncoder y StringIndexer (Sección 4.3).

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
6	148	72	35	0	33.6	0.627	50
1	85	66	29	0	26.6	0.351	31
8	183	64	0	0	23.3	0.672	32
1	89	66	23	94	28.1	0.167	21
0	137	40	35	168	43.1	2.288	33
5	116	74	0	0	25.6	0.201	30
3	78	50	32	88	31.0	0.248	26
10	115	0	0	0	35.3	0.134	29

Figura 5.7: *Datos sin procesar en el dataset de diabetes*
Fuente: *Propia*

Después del preprocesamiento (Figura 5.8) se muestra el mapeo de la característica **Pregnancies** con preprocesamiento para su posterior entrenamiento en el aprendizaje automático (*Machine Learning*).

DiabetesPedigreeFunction	Age	Outcome	Pregnancies_string_encoded	Pregnancies_one_hot
0.627	50	1	6.0	(16,[6],[1.0])
0.351	31	0	0.0	(16,[0],[1.0])
0.672	32	1	8.0	(16,[8],[1.0])
0.167	21	0	0.0	(16,[0],[1.0])
2.288	33	1	1.0	(16,[1],[1.0])
0.201	30	0	5.0	(16,[5],[1.0])

Figura 5.8: *Datos preprocesados por transformación con OneHotEncoder y StringIndexer*
Fuente: *Propia*

Conclusiones

1. Las conclusiones en función al objetivo general son:
   Se logro aplicar las técnicas de preprocesamiento en tres datasets utilizando el framework Apache Spark.

2. Las conclusiones en función al primer objetivo especifico son:
   Se seleccionó 3 datasets de diferentes fuentes; dataset de diabetes y de sismos se descargaron del repositorio de Kaggle y dataset de COVID-19 se obtuvo de la página web del ministerio de salud del gobierno peruano.

3. Las conclusiones en base al segundo objetivo especifico son:
   Se realizó la exploración y análisis de tres datasets con el fin de verificar que no se hayan realizado ningún proceso anterior, es decir estos datos no fueron modificados por lo tanto contienen ruido.

4. Las conclusiones en base al tercer objetivo especifico son:
   Se logro aplicar las técnicas de preprocesamiento con el uso de Apache Spark en 3 datasets:
   Dataset de COVID-19, esta dataset contiene 39 5005 registros con 8 columnas (UUID, DEPARTAMENTO, PROVINCIA, DISTRITO, METODODX,EDAD, SEXO, FECHA RESULTADO) se realizó la limpieza de valores nulos, estandarización, y dar formato a los valores de acuerdo al tipo de dato, después de cada proceso se utilizó el test unitario integral para verificar la limpieza de los datos .
   Dataset de sismos con una cantidad considerable de registros y cuenta con los siguientes campos de FECHA, LATITUD, LONGITUD, PROFUNDIDAD EN KILÓMETROS, MAGNITUD, en el preprocesamiento se elimino algunos campos innecesarios luego se dio formato al campo de fecha también se cambio los tipos de datos de algunos campos, después de cada proceso se utilizó el test unitario integral para verificar la limpieza de los datos.
   Por último en la dataset de diabetes con una cantidad considerable de registros y cuenta con los siguientes campos; PREGNANCIES, GLUCOSE, BLOODPRESURE, SKINTHICKNESS, INSULIN, BMI, DIABETESPEDIGREEFUNCTION,AGE, OUTCOME en esta dataset se utilizo técnicas de preprocesamiento por categorización.

5. Las conclusiones en base al cuarto objetivo especifico son:
   Se validó la propuesta de la mejora de limpieza de los datos, se realizó una comparación antes y después del preprocesamiento para cada dataset mediante el uso de las librerías de pandas y el test unitario e integral (Great Expectations) también se realizó la visualización de los datos para verificar la limpieza de los datos.

6. Se subió el código y las datasets de los 3 casos de uso en un repositorio, https://github.com/Cadindira/COVID-19.

Recomendaciones y Trabajos Futuros

1. Elaborar una plataforma para compartir datos, utilizando nuestra metodología para la generación de datos de calidad, con métodos de administración de recursos (Restfull) utilizando formatos JSON o CSV, y políticas de digitalización de recursos para el gobierno peruano.

2. Se recomienda usar el lenguaje Scala debido a que es más rápido que pyspark, sin embargo es muy limitado para interactuar con otros lenguajes y librerias, por otro lado, puede ser utilizado para implementación de algoritmos de nivel más bajo.

3. Se recomienda aplicar nuestra metodología en datos de sismos de Perú (IGP - Instituto Geofísico del Perú).

4. Explorar otras formas de evaluar la calidad de datos.

5. Extender la metodología propuesta para *data streming* con Apache Flink.

Bibliografía

Chan, Y., Zhang, X. J., and Zhu, J. H. (2019). A distributed big data discretization algorithm under spark. In Jin, H., Lin, X., Cheng, X., Shi, X., Xiao, N., and Huang, Y., editors, *Big Data*, pages 107–119, Singapore. Springer Singapore.

García, S., Luengo, J., and Herrera, F. (2015). *Data preprocessing in data mining*, volume 72. Springer.

García, S., Ramírez-Gallego, S., Luengo, J., and Herrera, F. (2017). Big data: Preprocesamiento y calidad de datos. In *Monografía Big Data*, pages 17–23. Asociación de Técnicas de Información, Novática (Revista de la Asociación de Técnicos de Informática).

García-Gil., D., Alcalde-Barros., A., Luengo., J., García., S., and Herrera., F. (2019). Big data preprocessing as the bridge between big data and smart data: Bigdapspark and bigdapflink libraries. In *Proceedings of the 4th International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS,*, pages 324–331. INSTICC, SciTePress.

García-Gil, D., Luengo, J., García, S., and Herrera, F. (2019). Enabling smart data: Noise filtering in big data classification. *Information Sciences*, 479:135 – 152.

Hindman, B., Konwinski, A. D., Zaharia, M. A., Ghodsi, A., Joseph, A. D., Katz, R., and Stoica, S. J. S. I. (2011). Mesos: A platform for fine-grained resource sharing in the data center. pages 295–308.

Ji, C., Li, Y., Qiu, W., Awada, U., and Li, K. (2012). Big data processing in cloud computing environments. In *2012 12th International Symposium on Pervasive Systems, Algorithms and Networks*, pages 17–23.

Luengo, J., García-Gil, D., Ramírez-Gallego, S., García, S., and Herrera, F. (2020a). *Final Thoughts: From Big Data to Smart Data*, pages 183–186. Springer International Publishing, Cham.

Luengo, J., García-Gil, D., Ramírez-Gallego, S., García, S., and Herrera, F. (2020b). *Big Data Preprocessing: Enabling Smart Data*.

Luu, H. (2018). *Resilient Distributed Datasets*, pages 51–86. Apress, Berkeley, CA.

Michael, K. and Miller, K. W. (2013). Big data: New opportunities and new challenges [guest editors' introduction]. *Computer*, 46(6):22–24.

Muhtaroğlu, F. C. P., Demir, S., Obalı, M., and Girgin, C. (2013). Business model canvas perspective on big data applications. In *2013 IEEE International Conference on Big Data*, pages 32–37.

Olarte C., A. E. and Casaverde L., A. C. (2020). *ANALISIS MASIVO DE DATOS EN TWITTER PARA IDENTIFICACIÓN DE OPINIÓN*. UNSAAC.

Schell, R. (2013). Security — a big question for big data. In *2013 IEEE International Conference on Big Data*, pages 5–5.

Scott, J. A. (2015). Getting start with apache spark. MapR Technologie.

Smith, J. W., Everhart, J., Dickson, W., Knowler, W., and Johannes, R. (1988). Using the adap learning algorithm to forecast the onset of diabetes mellitus. pages 261–265.

Taleb, I. and Serhani, M. A. (2017). Big data pre-processing: Closing the data quality enforcement loop. In *2017 IEEE International Congress on Big Data (BigData Congress)*, pages 498–501.

Wang, S., Shrestha, N., Subburaman, A. K., Wang, J., Wei, M., and Nagappan, N. (2021). Automatic unit test generation for machine learning libraries: How far are we? In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 1548–1560.