Ramírez - Araiza - Salazar

Fundamentos de Analítica de Datos con Python:

ETL, limpieza e ingeniería de datos

WORKBOOK VERSION

APRENDA

Fundamentos de Analítica de Datos con Python:

ETL, limpieza e ingeniería de datos.

Felipe Ramírez
Ma. de Jesús Araiza
Francisco Salazar

Facultad de Contaduría Pública y Administración
Universidad Autónoma de Nuevo León, México.

APRENDA EDICIONES

Ramírez, J. F., Araiza, M. J., & Salazar, Á. F. (2023). Fundamentos de analítica de datos con Python: ETL, limpieza e ingeniería de datos. Monterrey, N.L.: Aprenda Ediciones.
ISBN: **978-607-95979-4-8 (Extracto)**

Páginas: 336 Formato: 17.5 x 22.75 cm.

APRENDA PRACTICANDO
SAN FELIX 5432-D, VISTA SOL,
GUADALUPE, NUEVO LEÓN,
MÉXICO.
AÑO DE EDICIÓN: 2023
1ª EDICION.
ISBN: 978-607-95979-4-8
Release 4.0

El contenido de este libro forma parte del curso "**Python para Data Science: ETL, limpieza e ingeniería de datos.**", disponible desde nuestras plataformas:

Aprenda.mx

AprendaStudio.com

AprendaPracticando.com

Descarga recursos, presentaciones y ejercicios, desde estos sitios.

Haz clic aquí, y recibe información adicional de este curso

Contenido

Introducción

¿Para quién es este libro?

Este libro forma parte de la colección Python para Data Science, desarrollado por Aprenda. La serie trata todos aquellos temas que van, desde lo más fundamental de la modelación y la analítica de datos, hasta temas especializados de la ciencia de datos.

Cada tomo de la serie se enfoca en un tema en particular.

Audiencia específica

Este libro está dirigido para personas con el siguiente perfil:

- Son personas no especialistas, ni en programación, ni en matemáticas, ni en estadística. Este es un curso para administradores, contadores, financieros, y todo aquél que maneje datos y necesite analizarlos.
- Tienen la necesidad de aprender los fundamentos de la analítica de datos, desde el punto de vista teórico.
- Tienen la necesidad de aprender cómo se integra información de diferentes fuentes, para integrar un solo conjunto de datos a partir del cual hacer trabajo de analítica de datos.
- Tienen la necesidad de aprender cómo desarrollar tareas de limpieza de datos, es decir, corregir en la medida de lo posible datos incorrectos, incompletos o incoherentes.

- Tienen la necesidad de aprender cómo desarrollar tareas de ingeniería de datos, es decir, seleccionar los datos que son relevantes, y excluir aquellos que no lo son.
- Tienen conocimientos básicos de programación en Python.
- Saben el conocimiento básico de cómo funcionan las libretas Jupyter, ya sea usando **Jupyter Notebook** o **Google Colab**.

¿Por qué leerlo?

Una habilidad *commodity* es aquella que todos deben saber desarrollar, y tenerla no te da ventaja competitiva profesional, pero no tenerla sí te deja en seria desventaja. Ejemplos de habilidades commodity hoy en día son hacer un documento en Word, o saber usar el correo electrónico.

Según Gartner, en el corto y mediano plazo, tener habilidades para realizar trabajos de analítica de datos van a ser un commodity. En la actualidad, los ingenieros de datos preparan las fuentes de datos, y se las dan a los analistas de datos, que preparan informes y tableros de datos, y se los dan al tomador de decisiones.

En el mediano y corto plazo, los ingenieros de datos prepararán fuentes de uso general, y los depositarán en un repositorio central; los tomadores de decisiones deberán tomar los datos desde ahí, y ser ellos mismos quienes hagan la analítica de datos que requieren para tomar sus decisiones.

La figura del analista de datos desaparece como tal, porque los tomadores de decisiones serán sus propios analistas. Esto nos deja con la necesidad de que todos los profesionales que en un momento dado requieran tomar decisiones, deben aprender a hacer analítica. El problema es que hacer analítica no es tan sencillo como aprender a usar el correo electrónico: requiere mucho más técnica.

En el mercado hay muchos libros y muchos cursos para aprender analítica de datos, ya sea con herramientas gráficas (**Power BI**, **Tableau**, **Excel**), y lenguajes de programación (**Python**, **R**). El problema con muchos de ellos es que se empeñan en complicarlo todo, explicando modelos matemáticos, estadística y modelación de datos, que mucha gente no especializada no entiende.

Este libro no es así: explica la teoría y las técnicas desde un punto de vista práctico y de utilidad. Hace énfasis en la práctica, y utiliza ejemplos sencillos que faciliten la comprensión de la técnica.

Estructura del libro

Si quieres aprender a hacer analítica de manera profesional, lo recomendable es leer todo el libro de manera secuencial.

En general, tiene dos partes:

1. **Parte teórica / formal (capítulos 1 al 5):** En esta parte se tratan conceptos y técnicas para analizar situaciones del mundo real, y traducir las observaciones en objetivos de análisis de datos, variables e hipótesis. El entregable de esta sección es haber definido las fuentes de datos que han de utilizarse, y haber identificado los datos que es necesario tener para realizar los trabajos de analítica. En esencia, esta sección trata el ***qué***.
    a. Si realizarás proyectos de analítica de datos desde cero, esta sección es indispensable para que adquieras habilidades blandas que te permitirán entregar el mayor valor que los datos pueden entregar, con conocimiento de causa.
    b. Si tu interés es programar Python con tareas relacionadas con la analítica de datos, puedes saltarte esta sección, sin problema.
        i. En los capítulos 4 y 5 se ven algunos temas de Python y la librería pandas, pero es con la finalidad de analizar la disponibilidad de datos, y tomar decisiones respecto a los objetivos de análisis y las hipótesis planteadas.

2. **Parte técnica / práctica (capítulos 6 al 13):** En esta parte se tratan a detalle técnicas de programación específica en lenguaje Python, para el desarrollo de tareas propias del modelo ETL (*Extract – Transform – Load*). Se cubren a mucho detalle las tareas de limpieza de

datos, así como de ingeniería de datos (*feature engineering*), tratamiento de datos ausentes y atípicos. El entregable de esta sección es un conjunto de datos de alta calidad, requerido para el procesamiento de datos con la herramienta de tu elección (**Excel**, **Power BI**, **Tableau**, **Python** o **R**).

  - Esta sección puede tomarse como referencia técnica para realizar tareas de limpieza de datos e ingeniería de datos, usando Python.

En el libro encontrarás:

- **Lecciones:** Explicación teórica y sintáctica de los temas de estudio para comprender la analítica de datos y las técnicas de limpieza e ingeniería de datos.
- **LABS:** Aplicación del lenguaje Python para desarrollar un trabajo de analítica completo. Los ejercicios van en secuencia, y requieren la realización de los ejercicios previos.
  - Se recomienda que realices por ti mismo los ejercicios.
  - Los recursos para el desarrollo de los ejercicios los encontrarás en **GitHub**:

https://github.com/AprendaPracticando/AnaliticaPythonR1

Cursos en línea y presenciales

La versión en videocurso de este libro está disponible en Udemy y en AprendaStudio.com.

Si deseas el contenido de este curso en modalidad Presencial o Virtual, o si deseas sesiones LIVE, Webinars y Conferencias con los autores, la información está disponible en Aprenda.mx

Estamos seguros de que este libro te será de utilidad.

Felipe Ramírez, Ma. de Jesús Araiza, Francisco Salazar

Archivos y recursos de trabajo

Formato de los archivos de trabajo

Los ejercicios de este curso son ***libretas de Jupyter***, de extensión **`.ipynb`**.

Si no sabes qué son las libretas de Jupyter, cómo se editan y cómo se ejecuta código en ellas, te recomiendo que tomes el curso Jupyter Notebook, Google Colab y Markdown para todos, disponible en Udemy en la siguiente liga:

https://www.udemy.com/course/jupyter-notebook-y-markdown-para-todos/

Plataforma de trabajo

Para editar las libretas de Jupyter y ejecutar el código que contienen, necesitas una plataforma que los soporte.

Sugerimos dos:

1. **Google Colab**, si dispones de una conexión a Internet.

   https://colab.research.google.com/

2. **Jupyter Notebook**, disponible con **Anaconda**, si no dispones de una conexión a Internet.

https://www.anaconda.com

Se recomienda ampliamente que al iniciar el curso ya hayas registrado una cuenta de **Google** en **Google Colab**, y que cuando estés estudiando el material de este libro, estés dentro de tu ambiente de **Google Colab**, listo para poner manos a la obra en los ejercicios.

Archivos de trabajo y material adicional

Los archivos complementarios a este libro se encuentran en el siguiente repositorio **GitHub**.

https://github.com/AprendaPracticando/AnaliticaPythonR1

La estructura de carpetas es la siguiente:

1. **`<main>`**: Contiene las libretas Jupyter (**`.ipynb`**) generales del curso, así como el folleto y el mapa del curso, en PDF.
2. **`data`**: Contiene los archivos de datos que se utilizan para los Demos y Ejercicios.
3. **`images`**: Contiene las imágenes de algunas porciones explicativas del contenido.
4. **`labs`**: Contiene los archivos base para la realización de los LABS contenidos en el curso.

Los archivos de datos son accesibles desde **GitHub**.

1. **`pasajeros_titanic.csv`**: está disponible mediante la liga:

https://raw.githubusercontent.com/AprendaPracticando/AnaliticaPythonR1/main/data/pasajeros_titanic.csv

2. **`clases.csv`**: Está disponible mediante la liga:

https://raw.githubusercontent.com/AprendaPracticando/AnaliticaPythonR1/main/data/clases.csv

LAB 00.01: Verificar el acceso a los datos de prueba

En este Lab se verifica la posibilidad de acceder a los datos de prueba que serán utilizados en el libro.

Lo recomendable es que hayas registrado una cuenta de **Google** para acceder a **Google Colab**, y que ingreses a **Google Colab** con tu cuenta. Esto no es requerido, pero es ampliamente recomendable.

Las tareas por realizar son:

1. Abrir una libreta de Jupyter que está en **GitHub**.
2. Abrir la libreta de Jupyter en **Google Colab**.
3. Ejecutar el código Python de la libreta Jupyter.
4. Guardar una copia tu propio **Google Colab**.

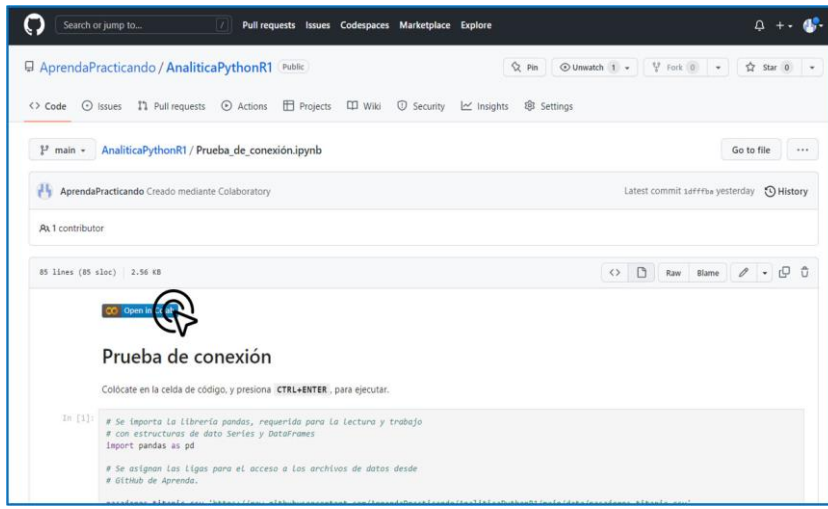
---

1. Abrir una libreta de Jupyter que está en **GitHub**.
    a. Ingresa al repositorio de GitHub donde están los archivos de trabajo del libro. La liga es esta:

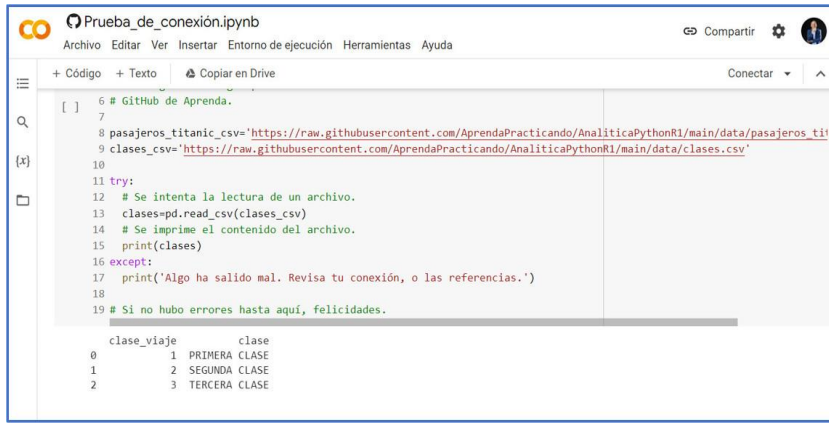
https://github.com/AprendaPracticando/AnaliticaPythonR1

2. Haz clic sobre el archivo **`prueba_conexión.ipynb`**, para abrirlo.

3. Abrir la libreta de Jupyter en **Google Colab**.
    a. Haz clic en el botón ***Open in Colab***, que aparece en la parte superior del código.
    b. Esto abrirá la libreta de Jupyter en **Google Colab**.



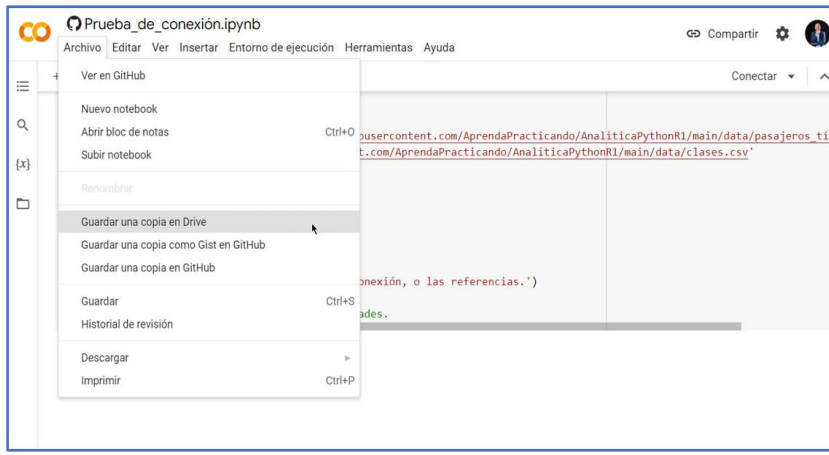
4. Ejecutar el código Python de la libreta Jupyter.
    a. Colócate en la celda de código Python de la libreta Jupyter.
    b. Presiona **CTRL+ENTER** para ejecutar el código.
    c. Si aparecen datos, ¡Listo! Todo ha transcurrido bien.



5. Guardar una copia tu propio **Google Colab**.

    a. En **Google Colab**, ve al menú ***Archivo***, y selecciona ***Guardar copia en Drive***; con esto, podrás guardar el archivo en tu ambiente de Google Colab.

    b. A partir de este momento, trabaja con tu propia libreta Jupyter, donde podrás hacer comentarios y anotaciones que encuentres pertinentes.



Esta mecánica se seguirá cada vez que utilices una libreta de Jupyter del repositorio **GitHub** donde están los archivos complementarios del libro.

Uso de los LABS

En el folder **`labs`** del repositorio **Github**, busca el archivo correspondiente al LAB que estés trabajando.

Su nombre será **`LAB_CA_SE.ext`**

Donde aparece <**CA**> vendrá el número del capítulo al que pertenece, mientras que donde aparece <**SE**>, aparecerá el número de LAB del capítulo.

En lugar de <**`.ext`**>, aparecerá la extensión del tipo de archivo que se trate: si es **`.pdf`**, quiere decir que el LAB no requiere ejecutar código; si es **`.ipynb`**, quiere decir que es una libreta Jupyter donde codificarás y ejecutarás líneas de código Python.

En este segundo caso: ***a)*** Abres la libreta Jupyter; ***b)*** Solicitas abrirlo en Colab (***Open in Colab***); ***c)*** Guarda tu propia copia de la libreta en **Colab**; ***d)*** Codifica lo que se pida en el LAB, por ti mismo; ***e)*** Comprueba que el resultado que obtienes es el mismo que se señala en el libro.

FIN DEL LAB

Capítulo 1:

Analítica de datos, datos e información

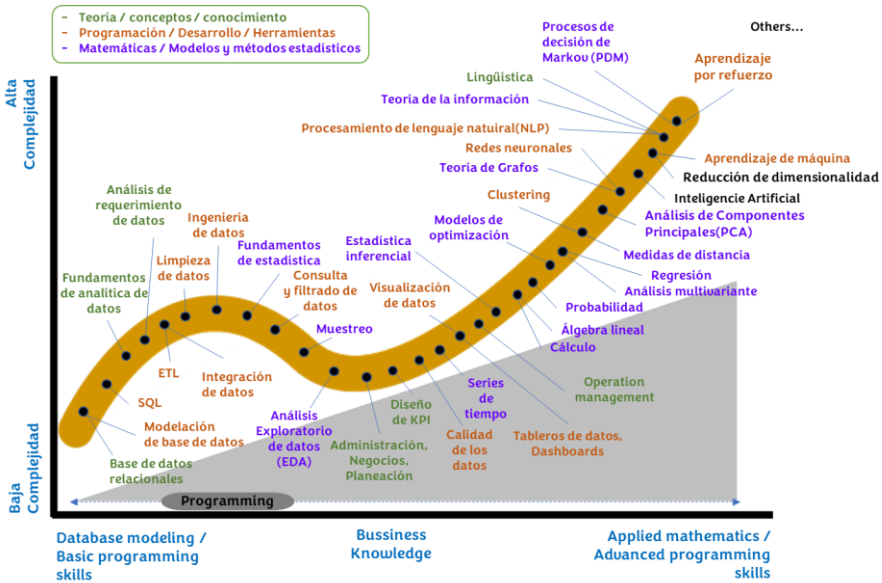


FIGURA 01.01: *Ruta de aprendizaje para la ciencia de datos.*

Este libro se mantiene en el ámbito de la analítica de datos. Big Data y Data Science quedan fuera de su alcance.

Capítulo 2:

ETL (extract, transform, load)

Analiza este modelo de datos:

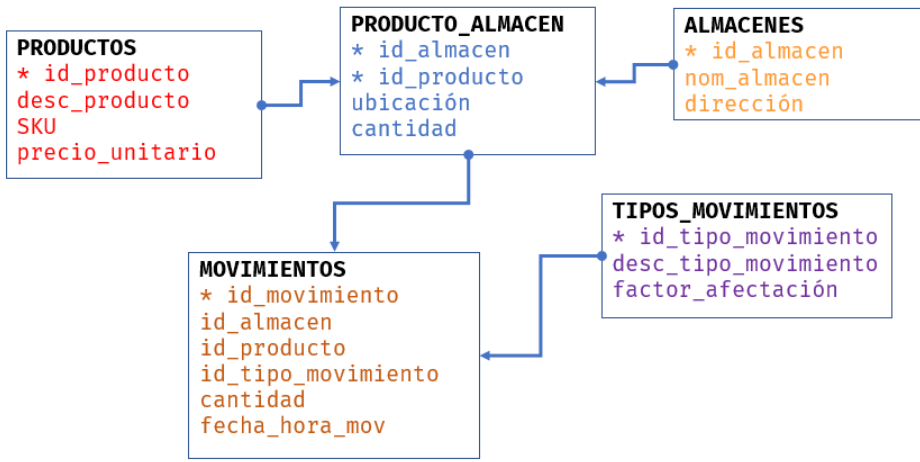


FIGURA 02.01: *Diagrama DER de un control de almacenes.*

La tabla más detallada es **`MOVIMIENTOS`**: Es tabla débil para **`PRODUCTOS_ALMACEN`** y **`TIPOS_MOVIMIENTO`**, y no es tabla fuerte de nadie.

La segunda tabla más detallada es **`PRODUCTO_ALMACEN`**, que es tabla débil para **`PRODUCTOS`** y **`ALMACENES`**, y es tabla fuerte para **`MOVIMIENTOS`**.

1. Primero, se integran los campos de la tabla **`PRODUCTO_ALMACEN`**, con los campos de sus tablas fuertes.

```
id_almacen + id_producto + ubicación + cantidad +
nom_almacen + dirección + desc_producto + SKU +
precio_unitario
```

2. Luego, se integran los campos de la tabla **`MOVIMIENTOS`**, con los campos de sus tablas fuertes. Los atributos de coincidencia persisten los de la tabla débil.

```
id_movimiento + id_almacen + id_producto +
id_tipo_movimiento +  cantidad + fecha_hora_mov +
ubicación + cantidad + nom_almacen + dirección +
desc_producto + SKU + precio_unitario +
desc_tipo_movimiento + factor_afectación
```

3. Si disponemos de este master de datos, no necesitamos conocer la estructura de la base de datos, ni las relaciones entre tablas ni la información de las llaves: todo lo que ocupamos, está ahí.

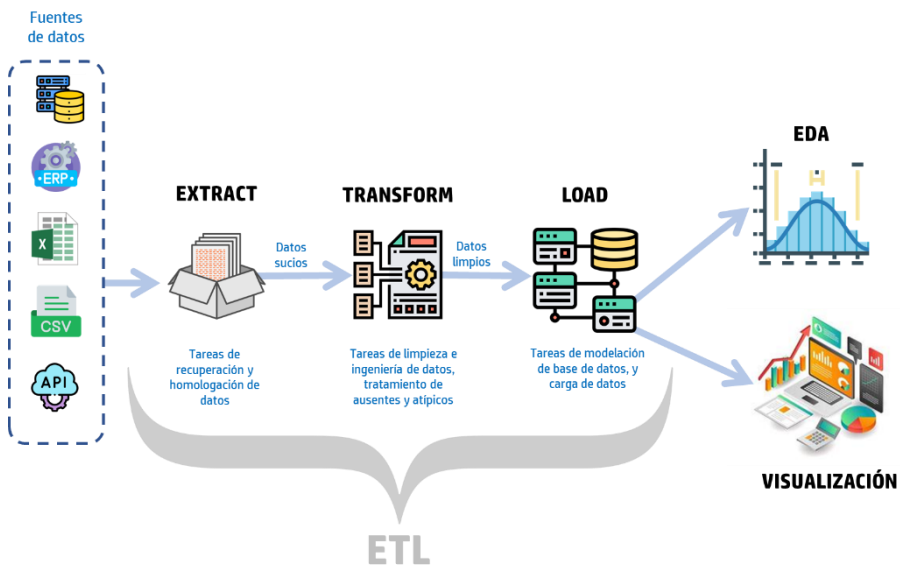


FIGURA 02.02: *Modelo ETL (Extract – Transform – Load)*

CAPÍTULO 3:
Análisis semántico del caso

FIGURA 03.01: *Secuencia de análisis semántico. Del análisis del caso, a las variables.*

LAB 03.01: Analizar el contexto del caso

En este Lab se analiza el contexto del caso, tratando de entender cuál es el tema y sus alcances.

Las tareas por realizar son:

1. Entender de qué se trata el tema.
2. Suponer las probables fuentes de datos.

Entender de qué se trata el tema.

El **RMS Titanic** fue un transatlántico británico, el mayor barco de pasajeros del mundo al finalizar su construcción, que se hundió durante la noche del 14 y la madrugada del 15 de abril de 1912 durante su viaje inaugural desde Southampton a Nueva York.

En el hundimiento del Titanic murieron 1,496 personas de las 2,224 que iban a bordo, lo que convierte a esta catástrofe en uno de los mayores naufragios de la historia ocurridos en tiempos de paz.

El naufragio del Titanic conmocionó e indignó al mundo entero por el elevado número de víctimas mortales y por los errores cometidos en el accidente. Las investigaciones públicas realizadas en Reino Unido y los Estados Unidos llevaron a la implementación de importantes mejoras en la seguridad marítima y a la creación en 1914 del *Convenio Internacional para la Seguridad de la Vida Humana en el Mar* (SOLAS, por sus siglas en inglés), que todavía hoy rige la seguridad marítima.

Se ha podido documentar algo de información relacionado con la tragedia. La enciclopedia británica, una de las más completas al respecto, detalla información de 1,310 pasajeros, lo que permite hacer un poco de trabajos de analítica.

Una de las fuentes más confiables de datos respecto al tema es la Encyclopedia Britannica.

Suponer las probables fuentes de datos.

Si se deseas obtener el origen de datos, pueden obtenerse aquí: https://data.world/nrippner/titanic-disaster-dataset (Fuente: Encyclopedia Britannica, Marzo 3, 2023).

FIN DEL LAB

LAB 03.02: Identificar los supuestos del caso

En este Lab se analiza el contexto del caso, y a partir de ahí, se generarán algunos supuestos o afirmaciones que deberán ser demostrada de manera técnica a través de técnicas de analítica.

Las tareas por realizar son:

1. Generar supuestos a partir del contexto.

Todos o casi todos vimos la película del TITANIC. Tomando en cuenta el entendimiento generalizado del tema que nos proporciona la película, podemos llegar a los siguientes supuestos.

1. Parecería que la gente que viajaba en primera clase tuvo más acceso a los botes salvavidas, y, por tanto, sobrevivieron.
2. Parecería que las mujeres se salvaron más que los hombres.
3. Parecería que la gente de más edad tenía más probabilidad de salvarse, así como los niños.
4. Parecería que las personas devotas y religiosas tuvieron mejores probabilidades de salvarse.
5. Parecería que las personas que viajaban con otras personas corrieron más riesgo de morir, por andar buscando a alguien más.
6. Seguramente se cumplió alguna superstición en el TITANIC, por ejemplo, que se haya quebrado un espejo, o que alguien haya tocado un cuchillo que se cayó al suelo, lo que provocó la mala suerte de la embarcación.

Estos supuestos son la materia prima para formular hipótesis.

FIN DEL LAB

LAB 03.03: Redactar el objetivo del análisis del caso

En este Lab se analizan los supuestos coloquiales del caso, y se establece un *objetivo de análisis de datos*, que represente los alcances y límites definitivos del estudio.

Las tareas por realizar son:

7. Redactar el *objetivo de análisis de datos*.

Para el caso de estudio, podemos proponer el siguiente objetivo de análisis.

OBJETIVO DE ANÁLISIS:

"Considerando la información histórica contenida en la enciclopedia británica, relativa a personas que iban en el TITANIC y la suerte que tuvieron, queremos hacer un análisis exploratorio histórico muestral que nos permita saber cómo afectaron a la sobrevivencia de las personas, aspectos como la clase de pasajero, sexo, rango de edad, religión, y si la persona viajaba sola o acompañada".

¿Por qué esta declaratoria de objetivo es buena? Porque contiene los siguientes elementos:

1. Se hace referencia a que se trabajará con información ya existente. Eso quiere decir que no hay un flujo de información constante que se esté actualizando constantemente. Se usa lo que ya hay.
2. Se hace referencia a que es un análisis exploratorio, lo que quiere decir que se centrará en técnicas de estadística descriptiva; otras alternativas son análisis inferenciales, como por ejemplo las regresiones, análisis de clasificación y conglomerados, etcétera.

3. Se hace referencia a que es un análisis histórico, por lo cual se usa información del pasado, que puede no estar completa y es difícil de complementar.
4. Se hace referencia a que es un análisis muestral. Al no especificar qué tipo de muestra se utilizará, se asume una muestra aleatoria. Esto nos hace saber que no es un estudio censal, y que probablemente no tenemos todos los datos.
5. Se señalan una o más variables dependientes (variables que se explican en función a otras), en este caso, la sobrevivencia.
6. Se señalan una o más variables independientes (variables que explican a las variables dependientes), en este caso, clase de pasajero, sexo, rango de edad, y acompañamiento de la persona.

Para este caso de estudio, y dado que nos guiaremos por una declaratoria de objetivo de análisis, es claro que optaremos por un análisis **basado en objetivo**.

FIN DEL LAB

LAB 03.04: Redactar las hipótesis del caso

En este Lab se analiza el objetivo de análisis de datos, y a partir de ahí se generan hipótesis de investigación, que incluyan variables, relaciones entre ellas, y un supuesto a demostrar.

Las tareas por realizar son:

1. Analizar el *objetivo de análisis de datos* e identificar variables.
2. Elaborar hipótesis que incluyan variables, relaciones entre ellas, y supuestos a comprobar.

Analizar el objetivo de análisis de datos e identificar variables.

El objetivo de análisis de datos es el siguiente:

"Considerando la información histórica contenida en la enciclopedia británica, relativa a personas que iban en el TITANIC y la suerte que tuvieron, queremos hacer un análisis exploratorio histórico que nos permita saber cómo afectaron a la sobrevivencia de las personas, aspectos como la clase de pasajero, sexo, rango de edad, religión, y si la persona viajaba sola o acompañada".

Las variables identificables pueden ser estas:

[Considerando la información histórica contenida en la enciclopedia británica, relativa a personas que iban en el TITANIC y la suerte que tuvieron,] ***(fuente)*** [queremos hacer un análisis exploratorio histórico que nos permita saber] ***(tipo de estudio)*** como afectaron a la ***sobrevivencia de las personas***, aspectos como la ***clase de pasajero***, ***sexo***, ***rango de edad***, ***religión***, y si la persona viajaba sola o ***acompañada***.

Elaborar hipótesis que incluyan variables, relaciones entre ellas, y supuestos a comprobar.

De cada *supuesto* del caso, infiere las *variables*.

Trata de establecer relaciones entre variables (si es que existen), y menciona el supuesto, desde el punto de vista de las variables.

Supuesto	Hipótesis
Parecería que la gente que viajaba en primera clase tuvo más acceso a los botes salvavidas, y por tanto sobrevivieron.	**H1:** La ***clase*** en la que viajaba el pasajero, sí tuvo que ver con la probabilidad de ***sobrevivencia***. Creemos que, a mayor clase, más probabilidad de sobrevivencia.
Parecería que las mujeres se salvaron más que los hombres.	**H2:** El ***sexo*** del pasajero, sí tuvo que ver con la probabilidad de ***sobrevivencia***. Creemos que las mujeres se salvaron más que los hombres.
Parecería que la gente de más edad tenía más probabilidad de salvarse, así como los niños.	**H3:** La ***edad*** del pasajero, sí tuvo que ver con la probabilidad de ***sobrevivencia***. Creemos que los infantes y los viejos tuvieron más probabilidad de sobrevivir.
Parecería que las personas devotas y religiosas tuvieron mejores probabilidades de salvarse.	**H4:** La ***religión*** del pasajero, no tuvo que ver con la probabilidad de ***sobrevivencia***.
Supuesto	Hipótesis
Parecería que las personas que viajaban con otras personas corrieron más riesgo de morir, por andar buscando a alguien más.	**H5:** El ***número de acompañantes*** del pasajero, sí tuvo que ver con la probabilidad de ***sobrevivencia***. Creemos que las personas que viajaban solas, al no tener que preocuparse por nadie más, tuvieron más probabilidad de sobrevivir.
Es probable que alguien no cumplió con una superstición, lo que trajo mala suerte y provocó el hundimiento.	**H6:** El ***cumplimiento de supersticiones*** tuvo que ver con que ocurriera el ***hundimiento*** de la embarcación.

FIN DEL LAB

LAB 03.05: Revisar la coherencia de las hipótesis del caso

En este Lab se analiza si las hipótesis y el objetivo de análisis de datos son coherentes. Esto es, asegurarse que no hay hipótesis que no tengan que ver con el objetivo.

Las tareas por realizar son:

1. Analizar la coherencia entre las hipótesis y el objetivo de análisis de datos.
2. Corregir diferencias o inexactitudes.

Analizar la coherencia entre las hipótesis y el objetivo de análisis de datos.

En nuestro caso, podría ser lo siguiente.

1. Respecto a la hipótesis 5:
    a. La hipótesis 5 es más amplia o detallada que el objetivo de análisis.
    b. Mientras que el objetivo el saber si la persona viajaba o no acompañada, la hipótesis profundiza en el número de acompañantes, que es más complicado de conocer.

2. Respecto a la hipótesis 6:
    a. La hipótesis 6 no es coherente. No hay registro de cumplimiento de supersticiones o no, y, además, trata del suceso en general, siendo que el resto de las hipótesis analizan rasgos de una persona, y consecuencias para una persona.
    b. Esta hipótesis sería coherente si estuviéramos comparando múltiples tragedias de hundimiento, y el hecho de que las personas en la embarcación cumplieran o no con supersticiones, y además, se documentara.

Corregir diferencias o inexactitudes.

En nuestro caso, serían las siguientes modificaciones:

1. Respecto a la hipótesis 5:
    a. Se decide modificar la hipótesis, ajustándola al objetivo, que refiere al hecho de que la persona viajara sola o acompañada, sin importar el número de acompañantes.
    b. Queda como sigue: **H5:** ~~El número de acompañantes del pasajero~~ El hecho de que la persona viajara sola o acompañada sí tuvo que ver con la probabilidad de sobrevivencia. Creemos que las personas que viajaban solas, al no tener que preocuparse por nadie más, tuvieron más probabilidad de sobrevivir.

2. Respecto a la hipótesis 6:
    a. Se elimina la hipótesis, por no ser consistente con el resto de las hipótesis.
    b. ~~**H6:** El cumplimiento de supersticiones tuvo que ver con que ocurriera el hundimiento de la embarcación.~~

FIN DEL LAB

LAB 03.06: Documentar información de la fuente

En este Lab se revisa la forma en que se tiene que documentar la información de una fuente de datos.

Las tareas por realizar son.

1. Documentar la información de una fuente de datos.

Esta tabla muestra la información de la fuente para el caso de análisis de datos del TITANIC.

Parámetro	Detalle
Nombre corto:	TITANIC
Formato:	CSV
Nombre físico de la fuente	pasajeros_titanic.csv
Ubicación	\data
Servidor	GitHub - AnaliticaPythonR1
Dirección IP	No aplica
Última actualización:	3 de marzo de 2020
Frecuencia de actualización:	Versión final. No se actualiza.
Responsable:	*Encyclopedia Britannica*
Permisos requeridos de acceso:	No aplica
Protocolo de solicitud de acceso:	No aplica
Sincroniza con otras fuentes:	No

FIN DEL LAB

LAB 03.07: Identificar variables dependientes e independientes

En este Lab se identifican las variables dependientes y dependientes que se tienen en el caso.

Las tareas por realizar son:

1. Enumerar las variables del caso.
2. Distinguir las variables dependientes e independientes.

Enumerar las variables del caso.

Las variables que se identifican en el objetivo de análisis son las siguientes:

1. sobrevivencia
2. clase de pasajero
3. sexo
4. rango de edad
5. religión
6. acompañado

Distinguir las variables dependientes e independientes.

En cuanto a su dependencia, las variables serían de esta manera.

Dependiente	Independiente
sobrevivencia	**clase de pasajero**
	sexo
	rango de edad
	religión
	acompañamiento

La mayoría de las variables anteceden al hecho de que sobrevivieran o no en el accidente del TITANIC, y no hay manera de que la sobrevivencia las determine.

Tenemos 1 variable dependiente, y 5 variables independientes. En total tenemos 6 variables.

En este momento aún no sabemos si los datos están disponibles o no en las fuentes.

FIN DEL LAB

CAPÍTULO 4:
Análisis preliminar de los datos

LAB 04.01: Calcular el tamaño de la muestra para el caso

En este Lab se calcula la muestra estadística para el universo de datos del TITANIC.

Las tareas por realizar son:

1. Calcular el tamaño de la muestra estadística para el caso de análisis.

En el caso de que no tuviéramos todos los datos de los pasajeros, ¿cuál sería el mínimo de observaciones que necesitamos para realizar operaciones estadísticas con un 99% de nivel de confianza y un 5% de margen de error?

El universo de datos es equivalente al número de pasajeros que viajaban en el TITANIC, en este caso 2,225.

El cálculo sería así:

```
# Declara las variables, cuidando que sean del tipo correcto.
N=2224
p=0.50
q=1-p
E=0.05
Z=2.576

# Se codifica la fórmula, para calcular el tamaño de la
# muestra (n), y muestra el resultado.
# Toma en cuenta la propiedad conmutativa 'Cuando
# multiplicamos, el orden de los factores no afecta
# al producto'.
n=int((Z**2*p*q*N)/((N*E**2)+(Z**2*p*q)))

print(f'El tamaño de la muestra es {n}')
```

```
El tamaño de la muestra es 511
```

El tamaño de la muestra es de 511 observaciones. Menos que eso, ya no permite el nivel de confianza que se espera, con el margen de error tolerado.

Desde luego, existen otras fórmulas para el cálculo de la muestra, que tienen que ver con el tipo de hipótesis que se desea demostrar o comprobar, el número de colas de la curva, etcétera.

FIN DEL LAB

LAB 04.02: Trabajar con listas, diccionarios y tuplas

En este Lab se revisa la técnica para crear, recuperar elementos e iterar las colecciones más importantes de Python: listas, diccionarios y tuplas.

Las tareas por realizar son:

1. Codificar la creación, recuperación de valores e iteración de una lista.
2. Codificar la creación, recuperación de valores e iteración de un diccionario.
3. Codificar la creación, recuperación de valores e iteración de una tupla.

Codificar la creación, recuperación de valores e iteración de una lista.

```
# Crear una lista con 5 elementos.
lista = [10, 20, 30, 40, 50]

# Imprimir la lista creada.
print(lista)



# Imprimir el tipo de objeto y comprobar que es una lista.
print(type(lista))

# Recuperación de un elemento de la lista usando el índice
# base cero.  Recuperar el elemento con el índice 2
# (tercer valor de la lista).
print(lista[2])

# Iterar una lista usando for e imprimiendo cada elemento
# de la lista.
for e in lista:
  print(e)
```

```
[10, 20, 30, 40, 50]
<class 'list'>
30
10
20
30
40
50
```

Codificar la creación, recuperación de valores e iteración de un diccionario.

```
# Crear un diccionario.
diccionario={
  1:'uno',
  2:'dos',
  3:'tres',
  4:'cuatro',
  5:'cinco'
}

# Imprimir un diccionario.
print(diccionario)

# Imprimir el tipo de objeto y ver que es un diccionario.
print(type(diccionario))

# Recuperación de un elemento del diccionario, usando la llave.
print(diccionario[2])

print(diccionario[10]) # Provoca error: no existe la llave.

# Recuperación de un elemento del diccionario, usando
# la llave y get().
print(diccionario.get(2,'No encontrado'))
print(diccionario.get(10,'No encontrado'))

# Iteración de las llaves de un diccionario.
for k in diccionario.keys():
  print(k)
```

```
# Iteracion de los valores de un diccionario.
for v in diccionario.values():
  print(v)

# Iteración de llaves y valores de un diccionario, en
# forma de tupla.
for t in diccionario.items():
  print(t)

for k,v in diccionario.items():
  print(f'La llave {k} está asociada al valor {v}')
```

```
{1: 'uno', 2: 'dos', 3: 'tres', 4: 'cuatro', 5: 'cinco'}
<class 'dict'>
dos
dos
No encontrado
1
2
3
4
5
uno
dos
tres
cuatro
cinco
(1, 'uno')
(2, 'dos')
(3, 'tres')
(4, 'cuatro')
(5, 'cinco')
La llave 1 está asociada al valor uno
La llave 2 está asociada al valor dos
La llave 3 está asociada al valor tres
La llave 4 está asociada al valor cuatro
La llave 5 está asociada al valor cinco
```

Codificar la creación, recuperación de valores e iteración de una tupla.

```
# Crear una Tupla.
tupla=('A','B','C')

# Imprimir una tupla.
print(tupla)

# Imprimir el tipo de objeto y ver que es una tupla.
print(type(tupla))

# Recuperación de un elemento de la tupla usando el
# índice base cero. Recuperar el elemento con el índice1
# (segundo valor de la tupla).
print(tupla[1])

# Iteración de los elementos de una tupla.
for e in tupla:
  print(e)
```

```
('A', 'B', 'C')
<class 'tuple'>
B
A
B
C
```

FIN DEL LAB

LAB 04.03: Cargar datos a un DataFrame desde un archivo CSV

En este Lab se revisa el código requerido para cargar datos, desde un archivo CSV alojado en GitHub, y cargar los datos en un DataFrame de pandas.

Las tareas por realizar son:

1. Cargar datos desde un archivo CSV disponible en **GitHub**.
2. Revisar la forma de un conjunto de datos usando **shape**.
3. Analizar si la forma del DataFrame es apropiada.

Cargar datos desde un archivo CSV disponible en GitHub.

```
# Se importa la librería pandas, para el manejo de datos.
# Se le asigna el alias "pd", que es el estándar de
# codificación de la comunidad.

import pandas as pd

# Como los datos están en GitHub, en lugar del nombre de
# archivo se utiliza la URL para acceso raw a los datos,
# que es lo mismo que leer un archivo que tenemos físicamente
# en nuestro equipo.
# Se declara una variable para almacenar la URL.

pasajeros_titanic_csv='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
pasajeros_titanic.csv'

# Se cargan los datos del archivo "pasajeros_titanic.csv" a un
# DataFrame de pandas llamado "titanic", usando la función
# pd.read_csv(), usando como nombre de archivo la variable
# que almacena la URL para acceder a los datos.

titanic = pd.read_csv(pasajeros_titanic_csv)
```

IMPORTANTE: Con pandas trabajaremos con un arreglo bidimensional de datos. En la práctica, hablar del arreglo bidimensional de datos, es lo mismo que hablar de los datos, del conjunto de datos, o del DataFrame; por otro lado, será común que hagamos referencia a las columnas del DataFrame, como columna, campo, característica, e incluso dato. A partir de este momento, nos referiremos de forma indistinta a los elementos, con cualquiera de sus sinónimos.

Revisar la forma de un conjunto de datos usando shape

```
# Muestra la forma del conjunto de datos titanic,
# usando la propiedad shape

titanic.shape
```

```
(1310, 14)
```

Consulta por separado el número de filas y el número de columnas, aprovechando que **shape** retorna una tupla con los dos valores, mismos que pueden ser consultados usando el subíndice base cero de la tupla.

```
# Imprime el número de filas del DataFrame,
# usando len()
print(len(titanic))

# Imprime el número de filas del DataFrame,
# usando el elemento 0 de shape
print(titanic.shape[0])

# Imprime el número de filas del DataFrame,
# usando el elemento 1 de shape
print(titanic.shape[1])
```

```
1310
1310
14
```

Analizar si la forma del DataFrame es apropiada

En el caso, tenemos lo siguiente:

1. En el objetivo de análisis se detectaron 6 variables: 1 variable dependiente, 5 variables independientes.
2. El universo de datos es de 2,225, y el tamaño de la muestra estadística es de 511 observaciones, para un 99% de nivel de confianza y 5% de margen de error.

Aun sin ver los datos, podemos decir que la forma mínima que debería tener el conjunto de datos debería ser de 511 filas y 6 columnas. Menos que eso, podríamos creer que los datos no son suficientes para trabajos de analítica.

Dado que tenemos 1310 observaciones y necesitamos 511, podemos decir que son suficientes en apariencia; de hecho, andamos bastante holgados de registros, y podemos darnos el lujo de eliminar datos que contengan datos faltantes o atípicos (porque el análisis no se enfoca en ello).

Dado que se requieren 6 variables y tenemos 14, podemos decir que son suficientes en apariencia.

FIN DEL LAB

CAPÍTULO 5:
Análisis de variables

FIGURA 03.02: *Verificación de relevancia de variables.*

LAB 05.01: Análisis semántico de las variables.

En este Lab se revisan los campos requeridos por el objetivo de análisis y las hipótesis, y se identifican las variables requeridas, requeridas indirectas, y no requeridas.

Las tareas por realizar son:

1. Revisar las columnas y tipos de datos en un DataFrame.
2. Analizar el contenido teórico de las columnas.
3. Revisar la cobertura de variables dependientes e independientes.
4. Hacer la especificación de los datos que no se tienen.
5. Enumerar las variables requeridas.
6. Enumerar las variables requeridas indirectas.
7. Enumerar las variables requeridas inviables.
8. Enumerar las variables no requeridas.

Revisar las columnas y tipos de datos en un DataFrame.

```
# Datos base
import pandas as pd

pasajeros_titanic_csv='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
pasajeros_titanic.csv'

titanic = pd.read_csv(pasajeros_titanic_csv)

# Enumerar los campos de un DataFrame, y sus tipos de datos
# usando dtypes.

titanic.dtypes
```

```
clase_viaje      float64
sobrevivencia    float64
nombre            object
sexo              object
edad             float64
parientes          int64
familiares         int64
boleto            object
tarifa           float64
cabina            object
embarque          object
bote              object
cuerpo           float64
residencias       object
dtype: object
```

Analizar el contenido teórico de las columnas

Los campos del DataFrame, en el caso del ejemplo, son los siguientes, y este es su contenido:

1. **`clase_viaje`**: Indica en qué clase viajaba el pasajero. Puede ser **`1`**, **`2`**, o **`3`**, que corresponde a **`PRIMERA CLASE`**, **`SEGUNDA CLASE`** y **`TERCERA CLASE`**, respectivamente.
2. **`sobrevivencia`**: Indica si la persona murió o vivió. Puede ser 0 o 1, que corresponde a **`MUERTO`** y **`VIVO`**, respectivamente.
3. **`nombre`**: Es el nombre del pasajero. Incluye título personal.
4. **`sexo`**: Indica el sexo de la persona. Puede ser 'hombre' o 'mujer'.
5. **`edad`**: Edad en años del pasajero.
6. **`parientes`**: Número de parientes políticos que viajaban con el pasajero.
7. **`familiares`**: Número de parientes consanguíneos que viajaban con el pasajero.
8. **`boleto`**: Folio del boleto que adquirió el pasajero.
9. **`tarifa`**: Precio que pagó por el boleto el pasajero.
10. **`cabina`**: En el caso de primera clase y segunda clase, es el número de cabina donde se hospedaba.

11. **embarque**: Identificador del puerto donde se embarcó el pasajero. Puede ser **Q**, **S** o **C**, que corresponde a **QUEENSTOWN**, **SOUTH HAMPTON** y **CHERSBOURG**, respectivamente.
12. **bote**: Folio del bote al que se subió el pasajero. Todos los que subieron a bote, se salvaron.
13. **cuerpo**: Folio del cuerpo, asignado a las personas que murieron. Todos los que tienen folio de cuerpo, murieron.
14. **residencias**: Información del lugar donde tiene su casa y reside el pasajero.

Revisar la cobertura de variables dependientes e independientes

En nuestro caso, verificamos primero que haya una variable por cada variable dependiente.

1. Variables dependientes del modelo:
    a. sobrevivencia (**sobrevivencia**)
2. Verificamos después que haya una variable por cada variable independiente.
3. Variables independientes del modelo:
    a. Clase de pasajero (**clase_viaje**)
    b. Sexo (**sexo**)
    c. Rango de edad ***No existe***
    d. Religión ***No existe***
    e. Acompañado ***No existe***

Hacer la especificación de los datos que no se tienen

De las variables enumeradas, hay algunas que no existen en el conjunto de datos, así que definimos cómo deben ser:

1. **rango_edad**: Es un categórico descriptivo, que se asigna en función al valor de la columna edad, y a la siguiente tabla.

Categoría	Rango de edad
INFANTES	[0,12)
JÓVENES	[12,21)
ADULTOS	[21,35)
MEDIANA EDAD	[35,45)
ADULTOS MAYORES	[45,INF)

1. **religión**: Es un categórico descriptivo, que indica la religión del pasajero.
2. **acompañado**: Es un categórico dicotómico, que asume el valor de **SÍ**, si la persona viajaba acompañada, y **NO**, si la persona viajaba sola.

Enumerar las variables requeridas.

En el caso, las columnas requeridas son:

1. **sobrevivencia**
2. **clase_viaje**
3. **sexo**
4. **rango_edad** (**No existe**)
5. **religión** (**No existe**)
6. **acompañado** (**No existe**)

Enumerar las variables requeridas indirectas.

En nuestro caso, estos son los requeridos indirectos:

1. **sobrevivencia**: Se puede inferir usando las columnas **bote** y **cuerpo**; si el registro tiene valor en **bote**, quiere decir que la persona

sobrevivió, así que **sobrevivenicia** es **1**; si el registro tiene valor en **cuerpo**, quiere decir que la persona no sobrevivió, así que **sobrevivencia** es **0**.

2. **clase_viaje**: Con la información que tenemos, no hay manera de inferir la clase en que viajaba el pasajero.
3. **sexo**: Se puede inferir a partir de la información contenida en **nombre**, aunque no de manera exacta. Si el nombre es femenino, podemos afirmar que **sexo** es mujer; si el nombre es masculino, podemos afirmar que el **sexo** es hombre. Si el nombre proporciona título personal, puede ayudar aún más a la tarea.
4. **rango_edad**: Se puede inferir a partir de **edad**.
5. **religión**: Con la información que tenemos no hay manera de inferir la religión, y en virtud de que el dato no existe actualmente, se descarta su uso, por inexistente.
6. **acompañado**: Se puede inferir a partir de **familiares** y **parientes**. Si sumamos el valor de ambas columnas, y la suma es mayor a cero, es que **acompañado** es **SÍ**, o de lo contrario, **NO**. Se puede crear incluso un columna que almacene la cantidad de acompañantes, que sería la suma de **familiares** y **parientes**.

Las variables ***requeridas indirectas*** para el caso de ejemplo serían:

Requerido indirecto	Columna que ayuda a inferir
bote	**sobrevivencia**
cuerpo	**sobrevivencia**
nombre	**sexo**
edad	**rango_edad**
familiares	**acompañantes**
parientes	**acompañantes**
acompañantes	**acompañado**

Enumerar las variables requeridas inviables.

En el caso de ejemplo, son requeridos inviables:

1. **religión**.

Dado que **`religion`** no se tiene y no hay manera de inferir dicho dato, se descarta, y todas las partes del análisis que involucra dicha variable.

1. El objetivo de análisis queda como sigue, eliminando el tema de la religión: Considerando la información histórica contenida en la enciclopedia británica, relativa a personas que iban en el TITANIC y la suerte que tuvieron, queremos hacer un análisis exploratorio histórico que nos permita saber cómo afectaron a la sobrevivencia de las personas, aspectos como la clase de pasajero, sexo, rango de edad, ~~religión,~~ y si la persona sola o acompañada.
2. La hipótesis 4, relacionada con la religión, se elimina: ~~**H4:** La religión del pasajero, no tuvo que ver con la probabilidad de que viviera o muriera.~~

Enumerar las variables no requeridas.

En el caso de ejemplo tenemos los siguientes campos **no requeridos**.

1. **`boleto`**
2. **`tarifa`**
3. **`cabina`**
4. **`embarque`**
5. **`residencias`**

FIN DEL LAB

Clasificación de los datos

Categorías de los datos

La ***categorización de datos*** es la parte del proceso donde se especifica el tipo de dato que *debe* tener cada columna para cumplir con su cometido de la mejor manera. En ocasiones, si no se sabe o no se conoce el uso de un dato, puede clasificarse como está (*as-is*).

La clasificación de los datos puede darse desde diferentes aspectos o categorías:

1. ***Tipo de variable*:** Puede ser variable *cualitativa* o *cuantitativa*.
    a. Las variables cualitativas pueden ser, a su vez, *ordinales* o *nominales*.
    b. Las variables cuantitativas pueden ser, en cuanto a su naturaleza, *continuas* o *discretas*.
    c. Las variables cuantitativas pueden ser, en cuanto a su escala, de *intervalo* o de *razón*.
2. ***Tipo de dato (datatype)*:** Pueden ser int, float, bool, datetime, binary, etcétera.
3. ***Uso*:** En cuanto a su uso, pueden ser de identidad, descriptivos, categóricos, de valor, temporales.
    a. A su vez, los categóricos pueden ser numéricos, codificados, descriptivos, dicotómicos y de intervalo.
4. ***Origen*:** Puede ser que los datos ya se tengan, que se calculen, que no estén disponibles.
5. ***Relacionalidad*:** En cuanto a su relacionalidad (participación en un modelo de datos relacional), pueden ser atributos primos, atributos no primos, atributos estándar, etcétera.

Data Taxonomic Code (DTXC)

El ***código DTXC*** permite clasificar, a través de un código, cada columna de un conjunto de datos.

El código DTXC se compone de varias partes:

```
<Medida>-<Tipo de dato>-<Uso>-<Origen>-<Relacionalidad>
```

El guion intermedio puede cambiarse por una diagonal (/) o diagonal invertida (\ ), en caso de requerirse.

TIPOS DE MEDIDA (*Measurement type*)

Categoría	Categoría en inglés	Código DTXC
Cuantitativo	*Quantitative*	**QT**
Cuantitativo Discreto	*Quantitative Discrete*	**QT-DIS**
Cuantitativo Continuo	*Quatitative Continuous*	**QT-CON**
Cuantitativo Discreto de Escala de Intervalo	*Quantitative Discrete, Interval Scale*	**QT-DIS-IS**
Cuantitativo Discreto de Escala de Razón	*Quantitative Discrete, Ratio Scale*	**QT-DIS-RS**
Cuantitativo Continuo de Escala de Intervalo	*Quantitative Continuous, Interval Scale*	**QT-CON-IS**
Cuantitativo Continuo de Escala de Razón	*Quantitative Continuous, Ratio Scale*	**QT-CON-RS**
Cualitativo	*Qualitative*	**QL**
Cualitativo Nominal	*Qualitative Nominal*	**QL-NM**
Cualitativo Ordinal	*Qualitative Ordinal*	**QL-OR**
Tipo de medida no definido	*Not Defined Measurement Type*	**NDMT**

TIPO DE DATO (*Datatype*)

Categoría	Categoría en inglés	Código DTXC
Numérico	*Numeric*	**NUM**
Numérico Flotante	*Numeric Float*	**NUM-FL**
Numérico Entero	*Numeric Integer*	**NUM-INT**
Alfanumérico	*Alphanumeric*	**STR**
Booleano	*Boolean*	**BL**
Marca de tiempo	*Time Stamp*	**TS**
Binario	*Binary*	**BIN**
Binario Muy Largo	*Binary - Binary Large Object*	**BIN-BLOB**
Tipo de dato no definido	*Not Defined Data Type*	**NDDT**

USO DEL DATO (*Use*)

Categoría	Categoría en inglés	Código DTXC
Identidad	*Identity*	**ID**
Identidad no requerida	*Not required identity*	**NRID**
Categórico	*Categorical*	**CAT**
Categórico Numérico	*Categorical Number*	**CAT-NM**
Categórico Codificado	*Categorical Code*	**CAT-CD**
Categórico Descriptivo	*Categorical Description*	**CAT-DES**
Categórico de intervalo	*Categorical Interval*	**CAT-IV**
Categórico Dicotómico	*Categorical Dichotomic*	**CAT-DIC**
Descriptivo	*Description*	**DS**
Valor	*Value*	**VAL**
Valor detallado	*Detail Value*	**VAL-DET**
Valor agregado	*Aggregate Value*	**VAL-AGG**
Temporalidad	*Time Measure*	**TM**
Tiempo - Fecha/Hora	*Date time*	**TM-DTM**
Tiempo - Fecha	*Date*	**TM-DATE**
Tiempo - Hora	*Just Time*	**TM-TIME**
Uso no definido	*Not Defined Use*	**NDU**

ORIGEN DE DATO (*Source*)

Categoría	Categoría en inglés	Código DTXC
Se tiene	*Already Available*	**AA**
Se calcula	*Calculated Data*	**CAL**
No disponible	*Not Available*	**NA**
Origen no definido	*Not Defined Source*	**NDS**

RELACIONALIDAD DEL DATO (*Relationality*)

Categoría	Categoría en inglés	Código DTXC
Atributo primo	*Prime Attribute*	**PK**
Atributo primo y de llave foránea	*Prime Attribute, Foreign Key Attribute*	**PK–FK**
Atributo de llave foránea	*Foreign Key Attribute*	**FK**
Atributo estándar	*Standard Attribute*	**SA**
Relacionalidad no definida	*Not Defined Relationality*	**NDR**

LAB 05.02: Clasificar los datos usando código DTXC

En este Lab se clasificarán los datos existentes en el DataFrame del caso, determinando el código DTXC que le aplican a cada campo.

Las tareas por realizar son:

1. Definir el código DTXC de cada campo en el DataFrame.

```
# Datos base
import pandas as pd

pasajeros_titanic_csv='https://raw.githubusercontent.com/Apren-
daPracticando/AnaliticaPythonR1/main/data/pasajeros_tita-
nic.csv'

titanic = pd.read_csv(pasajeros_titanic_csv)

# Se revisan las columnas del DataFrame.

titanic.dtypes
```

```
clase_viaje      float64
sobrevivencia    float64
nombre            object
sexo              object
edad             float64
parientes          int64
familiares         int64
boleto            object
tarifa           float64
cabina            object
embarque          object
bote              object
cuerpo           float64
residencias       object
dtype: object
```

En el caso de ejemplo, los datos requeridos son clasificados de la siguiente manera:

Variable	DTXC
clase_viaje	QL-OR-STR-CAT-CD-AA-NDR
sobrevivencia	QL-NM-STR-CAT-CD-AA-NDR
nombre	QL-NM-STR-DS-AA-NDR
sexo	QL-NM-STR-CAT-DES-AA-NDR
edad	QT-CON-NUM-FL-VAL-DET-AA-NDR
parientes	QT-DIS-NUM-INT-VAL-DET-AA-NDR
familiares	QT-DIS-NUM-INT-VAL-DET-AA-NDR
bote	QL-NM-STR-NRID-AA-NDR
cuerpo	QL-NM-STR-NRID-AA-NDR
rango_edad	QL-OR-STR-CAT-DES-AA-NDR
acompañado	QL-NM-STR-CAT-DES-CAL-NDR
acompañantes	QT-DIS-NUM-INT-VAL-DET-AA-NDR

Es importante mencionar que los códigos DTXC se determinan respecto a cómo debe ser el dato, y no cómo es.

FIN DEL LAB

CAPÍTULO 6:

Visualización y filtrado de datos

LAB 06.01: Ver datos del DataFrame

En este Lab se revisan estrategias para visualizar datos de un DataFrame.

Las tareas por realizar son:

1. Ver todos los datos de un DataFrame.
    a. Ver las filas en los extremos de un DataFrame.
    b. Ver las primeras filas de un DataFrame.
2. Ver las últimas 13 filas de un DataFrame.
3. Ver el contenido de una columna.
    a. Ver el contenido de una columna, usando notación estándar.
    b. Ver el contenido de una columna, usando *dot notation*.
4. Ver el contenido de solo algunas columnas.

El símbolo ▸ indica que hay más columnas, que se omiten por espacio.

Ver todos los datos de un DataFrame.

```
# Datos base
import pandas as pd
pasajeros_titanic_csv='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
pasajeros_titanic.csv'
titanic = pd.read_csv(pasajeros_titanic_csv)

# Ver todos los campos del DataFrame titanic
titanic
```

```
      clase_viaje  sobrevivencia                                 nombre  \
0             1.0            0.0                  Andrews, Mr. Thomas Jr  ▶
1             1.0            0.0  Chisholm, Mr. Roderick Robert Crispin  ▶
2             1.0            0.0                          Fry, Mr. Richard  ▶
3             1.0            0.0                    Harrison, Mr. William  ▶
4             1.0            1.0                 Ismay, Mr. Joseph Bruce  ▶
...           ...            ...                                     ...
1307          NaN            0.0                     Baumann, Mr. John D  ▶
1308          NaN            NaN             Blackwell, Mr. Stephen Weart  ▶
1309          NaN            0.0                 Baxter, Mr. Quigg Edmond  ▶
[1310 rows x 14 columns]
```

Ver las filas en los extremos de un DataFrame

1. **Ver las primeras filas de un DataFrame.**

```
# Ver las primeras 5 filas del DataFrame titanic.
titanic.head()
```

```
   clase_viaje  sobrevivencia                                 nombre    sexo ▶
0          1.0            0.0                  Andrews, Mr. Thomas Jr  hombre ▶
1          1.0            0.0  Chisholm, Mr. Roderick Robert Crispin  hombre ▶
2          1.0            0.0                          Fry, Mr. Richard  hombre ▶
3          1.0            0.0                    Harrison, Mr. William  hombre ▶
4          1.0            1.0                 Ismay, Mr. Joseph Bruce  hombre ▶
```

```
   clase_viaje  sobrevivencia                                 nombre    sexo ▶
0          1.0            0.0                  Andrews, Mr. Thomas Jr  hombre ▶
1          1.0            0.0  Chisholm, Mr. Roderick Robert Crispin  hombre ▶
2          1.0            0.0                          Fry, Mr. Richard  hombre ▶
3          1.0            0.0                    Harrison, Mr. William  hombre ▶
4          1.0            1.0                 Ismay, Mr. Joseph Bruce  hombre ▶
```

2. Ver las últimas 13 filas de un DataFrame.

```
# Ver las últimas 13 filas del DataFrame titanic.
titanic.tail(13)
```

```
      clase_viaje  sobrevivencia                          nombre    sexo ▶
1297          3.0            0.0                Sage, Miss. Ada   mujer ▶
1298          3.0            0.0       Sage, Miss. Constance Gladys   mujer ▶
1299          3.0            0.0  Sage, Miss. Dorothy Edith "Dolly"   mujer ▶
1300          3.0            0.0            Sage, Miss. Stella Anna   mujer ▶
1301          3.0            0.0          Sage, Mr. Douglas Bullen  hombre ▶
1302          3.0            0.0               Sage, Mr. Frederick  hombre ▶
1303          3.0            0.0          Sage, Mr. George John Jr  hombre ▶
1304          3.0            0.0             Sage, Mr. John George  hombre ▶
1305          3.0            0.0    Sage, Mrs. John (Annie Bullen)   mujer ▶
1306          3.0            0.0                Storey, Mr. Thomas  hombre ▶
1307          NaN            0.0               Baumann, Mr. John D  hombre ▶
1308          NaN            NaN      Blackwell, Mr. Stephen Weart  hombre ▶
1309          NaN            0.0          Baxter, Mr. Quigg Edmond  hombre ▶
```

Ver el contenido de una columna.

1. Ver el contenido de una columna, usando notación estándar.

```
# Ver el contenido de la columna nombre
# usando notación estándar (entre square brackets)
titanic['nombre']
```

```
0                      Andrews, Mr. Thomas Jr
1       Chisholm, Mr. Roderick Robert Crispin
2                           Fry, Mr. Richard
3                     Harrison, Mr. William
4                     Ismay, Mr. Joseph Bruce
                        ...
1307                    Baumann, Mr. John D
1308              Blackwell, Mr. Stephen Weart
1309                  Baxter, Mr. Quigg Edmond
Name: nombre, Length: 1310, dtype: object
```

2. **Ver el contenido de una columna, usando dot notation.**

```
# Ver el contenido de la columna nombre,
# usando dot notation.
titanic.nombre
```

```
0                      Andrews, Mr. Thomas Jr
1       Chisholm, Mr. Roderick Robert Crispin
2                           Fry, Mr. Richard
3                     Harrison, Mr. William
4                    Ismay, Mr. Joseph Bruce
                          ...
1305           Sage, Mrs. John (Annie Bullen)
1306                      Storey, Mr. Thomas
1307                    Baumann, Mr. John D
1308              Blackwell, Mr. Stephen Weart
1309                  Baxter, Mr. Quigg Edmond

Name: nombre, Length: 1310, dtype: object
```

Ver el contenido de solo algunas columnas.

```
# Ver el contenido de las columnas clase_viaje, nombre,
# y tarifa, solamente.
titanic[['clase_viaje', 'nombre', 'tarifa']]
```

```
      clase_viaje                                nombre    tarifa
0             1.0                 Andrews, Mr. Thomas Jr    0.0000
1             1.0  Chisholm, Mr. Roderick Robert Crispin    0.0000
2             1.0                       Fry, Mr. Richard    0.0000
3             1.0                  Harrison, Mr. William    0.0000
4             1.0                Ismay, Mr. Joseph Bruce    0.0000
...           ...                                    ...       ...
1305          3.0         Sage, Mrs. John (Annie Bullen)   69.5500
1306          3.0                     Storey, Mr. Thomas       NaN
1307          NaN                   Baumann, Mr. John D   25.9250
1308          NaN           Blackwell, Mr. Stephen Weart   35.5000
1309          NaN               Baxter, Mr. Quigg Edmond  247.5208
[1310 rows x 3 columns]
```

FIN DEL LAB

LAB 06.02: Técnicas de filtrado de filas

En este Lab se revisan estrategias para filtrar filas usando **`loc[]`** y **`where()`**.

Las tareas por realizar son:

1. Filtrado usando **`loc[]`**, con una condición.
2. Filtrado usando **`loc[]`**, con varias condiciones.
3. Filtrado usando **`loc[]`**, combinado con filtrado de columnas.
4. Filtrado usando **`loc[]`**, con **`isin()`**.
    a. Valores incluidos en una lista.
    b. Valores no incluidos en una lista.
5. Filtrado usando **`where()`**.

El símbolo ▸ indica que hay más columnas, que se omiten por espacio.

```
# Datos base
import pandas as pd
pasajeros_titanic_csv='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
pasajeros_titanic.csv'
titanic = pd.read_csv(pasajeros_titanic_csv)
```

Filtrado usando loc[], con una condición

```
# Mostrar las filas donde clase_viaje sea 1.0, usando loc[].
# Como sólo es una condición, no se requiere encerrar la
# condición entre paréntesis.
primera_clase=titanic.loc[titanic['clase_viaje']==1.0]

primera_clase
```

```
     clase_viaje  sobrevivencia  \
0            1.0            0.0
1            1.0            0.0
2            1.0            0.0
..           ...            ...
317          1.0            1.0
318          1.0            1.0
319          1.0            1.0
[320 rows x 14 columns]
```

Filtrado usando loc[], con varias condiciones

```
# Mostrar las filas donde clase_viaje sea 1.0,
# y sexo sea 'mujer', usando loc[].
# Como se usa más de una condición, éstas se
# encierran entre paréntesis.
mujeres_primera_clase=titanic.loc[
    (titanic['clase_viaje']==1.0) &
    (titanic['sexo']=='mujer')
]

# Se muestran las filas seleccionadas.
mujeres_primera_clase
```

```
     clase_viaje  sobrevivencia   ▶
9            1.0            1.0   ▶
11           1.0            1.0   ▶
12           1.0            1.0   ▶
21           1.0            1.0   ▶
28           1.0            1.0   ▶
..           ...            ...
311          1.0            1.0   ▶
312          1.0            1.0   ▶
315          1.0            1.0   ▶
317          1.0            1.0   ▶
319          1.0            1.0   ▶

[143 rows x 14 columns]
```

Filtrado usando loc[], combinado con filtrado de columnas

```
# Combinación de filtrado de columnas y filas.
# Mostrar el nombre y la edad, de las filas
# donde sexo sea mujer.
columnas_filas=titanic[['nombre',
                'edad']].loc[titanic['sexo']=='mujer']

columnas_filas
```

```
                                                nombre  edad
9     Cornell, Mrs. Robert Clifford (Malvina Helen L...  55.0
11                          Leader, Dr. Alice (Farnham)  49.0
12    Swift, Mrs. Frederick Joel (Margaret Welles Ba...  48.0
21                         Newsom, Miss. Helen Monypeny  19.0
28                             Bonnell, Miss. Elizabeth  58.0
...                                                 ...   ...
1297                                   Sage, Miss. Ada   NaN
1298                       Sage, Miss. Constance Gladys   NaN
1299                  Sage, Miss. Dorothy Edith "Dolly"   NaN
1300                            Sage, Miss. Stella Anna   NaN
1305                       Sage, Mrs. John (Annie Bullen)   NaN
[465 rows x 2 columns]
```

Filtrado usando loc[], con isin().

1. **Valores incluidos en una lista.**

```
# Uso de filtro, comparando con el contenido de una lista.
# Recuperar las filas donde las cabinas sean C32, C26 y C80.
# Mostrar sólo las columnas nombre y cabina.
cabinas_especiales=['C32','C26','C80']

especiales=titanic[['nombre','cabina']].
      loc[titanic['cabina'].isin(cabinas_especiales)]

especiales
```

```
                                        nombre cabina
263         White, Mrs. John Stuart (Ella Holmes)    C32
264                       Young, Miss. Marie Grice    C32
289                    Widener, Mr. George Dunton    C80
291  Widener, Mrs. George Dunton (Eleanor Elkins)    C80
```

2. **Valores no incluidos en una lista.**

```
# Uso de filtro, comparando con el contenido de una lista.
# Recuperar las filas donde las cabinas NO sean C32, C26 y C80.
# Mostrar sólo las columnas nombre y cabina.
cabinas_especiales=['C32','C26','C80']

no_especiales=titanic[['nombre','cabina']].
          loc[~titanic['cabina'].isin(cabinas_especiales)]

no_especiales
```

```
                                  nombre        cabina
0                    Andrews, Mr. Thomas Jr          A36
1     Chisholm, Mr. Roderick Robert Crispin          NaN
2                           Fry, Mr. Richard         B102
3                     Harrison, Mr. William          B94
4                   Ismay, Mr. Joseph Bruce  B52 B54 B56
...                                    ...          ...
1305         Sage, Mrs. John (Annie Bullen)          NaN
1306                     Storey, Mr. Thomas          NaN
1307                   Baumann, Mr. John D           NaN
1308             Blackwell, Mr. Stephen Weart            T
1309                 Baxter, Mr. Quigg Edmond      B58 B60
[1306 rows x 2 columns]
```

Filtrado usando where().

```
# Se filtra el DataFrame usando where, de tal manera que
# se recuperan todas las filas; en aquellos casos donde
# sexo no sea mujer o sobrevivencia no sea 0, las columnas
# son rellenadas con NaN.
mujeres_vivas_where=titanic.where((titanic.sexo=='mujer') &
                                  (titanic.sobrevivencia==1.0))

# Se muestra el resultado. Son 1310 filas de 1310, las
# que se recuperan.
mujeres_vivas_where
```

```
      clase_viaje  sobrevivencia nombre sexo  edad  parientes  familiares  ▶
0             NaN            NaN    NaN  NaN   NaN        NaN         NaN  ▶
1             NaN            NaN    NaN  NaN   NaN        NaN         NaN  ▶
2             NaN            NaN    NaN  NaN   NaN        NaN         NaN  ▶
3             NaN            NaN    NaN  NaN   NaN        NaN         NaN  ▶
4             NaN            NaN    NaN  NaN   NaN        NaN         NaN  ▶
...           ...            ...    ...  ...   ...        ...         ...
1305          NaN            NaN    NaN  NaN   NaN        NaN         NaN  ▶
1306          NaN            NaN    NaN  NaN   NaN        NaN         NaN  ▶
1307          NaN            NaN    NaN  NaN   NaN        NaN         NaN  ▶
1308          NaN            NaN    NaN  NaN   NaN        NaN         NaN  ▶
1309          NaN            NaN    NaN  NaN   NaN        NaN         NaN  ▶
[1310 rows x 14 columns]
```

FIN DEL LAB

CAPÍTULO 7:

Limpieza de datos e ingeniería de datos (feature engineering)

LAB 07.01: Tareas generales con DataFrames

En este Lab se revisan estrategias para realizar tareas generales sobre filas y columnas en un DataFrame.

Las tareas por realizar son:

1. Eliminar columnas no requeridas usando **`drop()`**.
2. Eliminar filas duplicadas usando **`drop_duplicates()`**.
3. Eliminar filas con datos vacíos usando **`dropna()`**.
   a. Eliminar las filas con todas las columnas vacías.
   b. Eliminar las filas con 3 o más columnas vacías.
4. Crear un indicador ficticio.
5. Verificar unicidad de columnas usando **`duplicated()`**.
   a. Verificar la unicidad de un campo de identidad.
   b. Verificar la unicidad de un campo categórico.
   c. Verificar la unicidad de un campo con vacíos.

```
# Datos base
import pandas as pd
pasajeros_titanic_csv='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
pasajeros_titanic.csv'
titanic = pd.read_csv(pasajeros_titanic_csv)
```

Eliminar columnas no requeridas usando `drop()`

```
# Se eliminan los campos que no son requeridos del
# DataFrame titanic
titanic.drop(columns=['boleto','tarifa',
            'cabina','embarque','residencias'],
            inplace=True)

titanic.dtypes
```

```
clase_viaje      float64
sobrevivencia    float64
nombre            object
sexo              object
edad             float64
parientes          int64
familiares         int64
bote              object
cuerpo           float64
dtype: object
```

Eliminar filas duplicadas usando drop_duplicates()

```
# Eliminar duplicados en titanic, donde todas las columnas
# sean exactamente iguales en todos sus valores, guardando
# el resultado en el mismo DataFrame
titanic.drop_duplicates(inplace=True)
```

Eliminar filas con datos vacíos usando dropna()

```
# Se revisa cuántos datos hay.
print(f'Se tienen {titanic.shape[0]:,} filas.')

# Elimina todos los registros en donde todas las columnas
# estén vacías.
titanic.dropna(how='all', inplace=True)

# Se revisa cuántos datos hay.
print(f'Se tienen {titanic.shape[0]:,} filas.')

# Se concluye que no había filas completamente vacías.
```

```
Se tienen 1,309 filas.
Se tienen 1,309 filas.
```

```
# Elimina todos los registros con más de tres columnas vacías.
titanic.dropna(thresh=titanic.shape[1]-3, inplace=True)

# Se revisa si se eliminaron filas.
print(f'Se tienen {titanic.shape[0]:,} filas.')

# Se concluye que se eliminaron 2 filas.
```

```
Se tienen 1,307 filas.
```

Crear un indicador ficticio

```
# Generar una llave primaria (columna de identidad) para
# los datos del titanic.
# Se desea que el primer identificador sea 10000

inicial=10000
titanic['identificador']=pd.Series(range(inicial,
       len(titanic)+inicial))

# Ver los primeros registros, para verificar la creación
# del identificador.

titanic.head()

# Se debió haber agregado una columna nueva, llamada
# identificador, que es una serie numérica que inicia en 10000.
# Toma en cuenta que se ha generado como flotante, cosa que
# se tendrá que corregir más adelante.
```

```
   edad  parientes  familiares bote  cuerpo  identificador  ▶
0  39.0          0           0  NaN     NaN        10000.0  ▶
1   NaN          0           0  NaN     NaN        10001.0  ▶
2   NaN          0           0  NaN     NaN        10002.0  ▶
3  40.0          0           0  NaN   110.0        10003.0  ▶
4  49.0          0           0    C     NaN        10004.0  ▶
```

Verificar unicidad de columnas usando duplicated()

```
# Se valida si en identificador hay algún valor duplicado.
# El valor se asigna a la variable hay duplicados.
hay_duplicados = titanic['identificador'].duplicated().any()

# Se imprime el resultado. Debe ser False.
print(hay_duplicados)
```

```
False
```

```
# Se valida si en clase_viaje hay algún valor duplicado.
# El valor se asigna a la variable hay_duplicados.
hay_duplicados = titanic['clase_viaje'].duplicated().any()

# Se imprime el resultado. Debe ser True.
print(hay_duplicados)
```

```
True
```

Si queremos validar que una columna no tenga valores repetidos, sin tomar en cuenta los datos vacíos o nulos, primero se deben eliminar los datos vacíos o nulos, y luego verificar la unicidad.

En nuestro caso, el campo `cuerpo`, que indica el número de cuerpo asignado a las personas fallecidas, no debe tener repetidos, pero hay muchas filas en donde ese dato no se tiene.

```
# Se genera una Serie de pandas que contenga únicamente
# los valores de la columna cuerpo, cuando cuerpo no sea
# nulo.
valores_cuepo = titanic['cuerpo'][titanic['cuerpo'].notnull()]

# Se valida si en valores_cuerpo hay algún valor duplicado.
# El valor se asigna a la variable hay_duplicados.
hay_duplicados = valores_cuepo.duplicated().any()

# Se imprime el resultado. Debe ser False.
print(hay_duplicados)
```

```
False
```

FIN DEL LAB

LAB 07.02: Trabajo con columnas

En este Lab se revisan estrategias para filtrar filas usando **`loc[]`** y **`where()`**.

Las tareas por realizar son:

1. Eliminar columnas no requeridas usando **`drop()`**.
2. Eliminar filas duplicadas usando **`drop_duplicates()`**.
3. Eliminar filas con datos vacíos usando **`dropna()`**.
    a. Eliminar las filas con todas las columnas vacías.
    b. Eliminar las filas con 3 o más columnas vacías.
4. Crear un indicador ficticio.
5. Verificar unicidad de columnas usando **`duplicated()`**.
6. Verificar la unicidad de un campo de identidad.
    a. Verificar la unicidad de un campo categórico.
    b. Verificar la unicidad de un campo con vacíos.

```
# Datos base
import pandas as pd
pasajeros_titanic_csv='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
pasajeros_titanic.csv'
titanic = pd.read_csv(pasajeros_titanic_csv)
titanic.drop(columns=['boleto','tarifa','cabina',
         'embarque','residencias'],inplace=True)
titanic['identificador']=pd.Series(range(10000,
         len(titanic)+10000))
```

Reordenar las columnas de un DataFrame.

En nuestro caso, tenemos actualmente las siguientes columnas:

Variable	DTXC
clase_viaje	QL-OR-STR-CAT-CD-AA-NDR
sobrevivencia	QL-NM-STR-CAT-CD-AA-NDR
nombre	QL-NM-STR-DS-AA-NDR
sexo	QL-NM-STR-CAT-DES-AA-NDR
edad	QT-CON-NUM-FL-VAL-DET-AA-NDR
parientes	QT-DIS-NUM-INT-VAL-DET-AA-NDR
familiares	QT-DIS-NUM-INT-VAL-DET-AA-NDR
rango_edad	QL-OR-STR-CAT-DES-AA-NDR
bote	QL-NM-STR-NRID-AA-NDR
cuerpo	QL-NM-STR-NRID-AA-NDR
identificador	QL-NM-STR-ID-AA-NDR

Recuerda que los códigos DTXC contienen los datos como deben ser, y no como son.

Se desean colocar los campos siguiendo este orden:

1. Primero los datos de identidad (`identificador`)
2. Luego los datos correspondientes a las variables dependientes (`sobrevivencia`).
3. Luego los datos correspondientes a las variables independientes (todos los demás campos).
4. El orden para las variables dependientes e independientes, dentro de su categoría, seguirá este orden.
5. Datos descriptivos (`nombre`)
6. Luego los temporales (**no hay**)
7. Luego los categóricos descriptivos (`sexo`)
8. Luego los categóricos numéricos (**no hay**)
9. Luego los categóricos codificados (`clase_viaje`)
10. Luego los de identidad no requerido (`bote`, `cuerpo`)
11. Luego los datos de valor (`edad`, `parientes`, `familiares`).

```
# Genera una lista que contenga los nombres de columnas
# en el orden deseado.
campos_ordenados=['identificador', 'sobrevivencia',
        'nombre', 'sexo', 'clase_viaje', 'bote', 'cuerpo',
                  'edad', 'parientes', 'familiares']

# Recupera los campos en el orden especificado en la lista, y
# guárdalos en el mismo DataFrame origen. Dicho de otra forma,
# el DataFrame es la representación ordenada de sí mismo.
titanic=titanic[campos_ordenados]

# Enumera los campos del DataFrame para comprobar el
# nuevo orden, usando dtypes.
titanic.dtypes
```

```
identificador      int64
sobrevivencia    float64
nombre            object
sexo              object
clase_viaje      float64
bote              object
cuerpo           float64
edad             float64
parientes          int64
familiares         int64
dtype: object
```

Cambiar nombre de columnas en un DataFrame.

Se desean hacer los siguientes cambios de nombre de columna:

1. Los identificadores, requeridos o no requeridos, deben iniciar con **`id_`**.
2. Los datos de valor que indiquen cantidad deben iniciar con **`cantidad_`**.
3. Los categóricos numéricos y codificados, deben iniciar con **`clave_`**, porque requieren interpretación.
4. Se deben hacer precisiones de sustantivos. Por ejemplo, usar *pasajeros* resulta inexacto, pues la *tripulación* no era pasajera, entonces, lo adecuado sería hablar de *personas*.

Los nombres por modificar son estos:

Nombre actual	Nombre nuevo
identificador	**id_persona**
sobrevivencia	**clave_sobrevivencia**
bote	**id_bote**
cuerpo	**id_cuerpo**
parientes	**cantidad_parientes**
familiares	**cantidad_familiares**

```
# Genera un diccionario que contenga la equivalencia
# las llaves son los nombres de columna actuales, y los
# valores son los nombres que queremos asignar.
# Las columnas que no sufren cambio no se ponen.
nuevos_nombres={
    'identificador':'id_persona',
    'sobrevivencia':'clave_sobrevivencia',
    'bote':'id_bote',
    'cuerpo':'id_cuerpo',
    'parientes':'cantidad_parientes',
    'familiares':'cantidad_familiares'
}

# Renombra las columnas, usando rename(), aplica los
# cambios sobre el mismo DataFrame, usando inplace.
titanic.rename(columns=nuevos_nombres,inplace=True)

# Enumera los campos del DataFrame para comprobar los nuevos
# nombres, usando dtypes.
titanic.dtypes
```

```
id_persona              int64
clave_sobrevivencia    float64
nombre                  object
sexo                    object
clase_viaje            float64
id_bote                 object
id_cuerpo              float64
edad                   float64
cantidad_parientes       int64
cantidad_familiares      int64
dtype: object
```

Escribir un CSV a partir de un DataFrame.

Se toma como base el DataFrame **`titanic`**, con las transformaciones realizadas hasta el momento, y se guarda en un archivo en formato CSV llamado **`personas_titanic.csv`**, sin incluir el índice.

```
# Guarda el contenido del DataFrame titanic en un archivo
# CSV, usando to_csv(), incluyendo el parámetro index, para
# no incluir el índice.
titanic.to_csv('personas_titanic.csv', index=False)
```

Haz clic en el ícono de carpeta que aparece en la extrema izquierda de **Google Colab**, y comprueba que el archivo que has guardado está ahí.

Integrar dos DataFrames en un Libro en Excel con dos hojas

Supongamos que queremos generar dos Libros de Excel.

1. Un Libro de Excel llamado **`datos_actualizados.xlsx`**, que contiene una hoja llamado **`datos`**, que contiene todos los datos, sin incluir el índice.

2. Un Libro de Excel llamado **`vivos_muertos.xlsx`**, que:
    a. Contiene una hoja llamado **`vivos`**, que contiene el nombre, el sexo, clave sobrevivencia y la edad de las personas que hayan sobrevivido (**`clave_sobrevivencia=1`**).
    b. Contiene una hoja llamado **`muertos`**, que contiene el nombre, el sexo, clave sobrevivencia y la edad de las personas que hayan sobrevivido (**`clave_sobrevivencia=0`**).

```
# Se genera el primer Libro de Excel, llamado
# datos_actualizados.xlsx
# No se incluye el índice.
# El nombre de la hoja será datos.
titanic.to_excel('datos_actualizados.xlsx',
      index=False, sheet_name='datos' )

# Se recuperan las columnas nombre, sexo y edad, de
# las personas que vivieron.
vivieron=titanic[['nombre','sexo',
         'clave_sobrevivencia',
         'edad']][(titanic['clave_sobrevivencia']==1.0)]

vivieron
```

```
                                                nombre    sexo  ▶
4                                Ismay, Mr. Joseph Bruce  hombre  ▶
9     Cornell, Mrs. Robert Clifford (Malvina Helen L...   mujer  ▶
10                             Omont, Mr. Alfred Fernand  hombre  ▶
11                           Leader, Dr. Alice (Farnham)   mujer  ▶
12    Swift, Mrs. Frederick Joel (Margaret Welles Ba...   mujer  ▶
...                                                 ...     ...
1288                                  Chip, Mr. Chang  hombre  ▶
1289                                  Foo, Mr. Choong  hombre  ▶
1290                                    Hee, Mr. Ling  hombre  ▶
1291                                     Lam, Mr. Ali  hombre  ▶
1293                                   Lang, Mr. Fang  hombre  ▶
[500 rows x 4 columns]
```

```
# Se recuperan las columnas nombre, sexo y edad,
# de las personas que murieron.
murieron=titanic[['nombre','sexo',
          'clave_sobrevivencia',
          'edad']][(titanic['clave_sobrevivencia']==0.0)]

murieron
```

```
                                    nombre    sexo  clave_sobrevivencia  edad
0                      Andrews, Mr. Thomas Jr  hombre                  0.0  39.0
1     Chisholm, Mr. Roderick Robert Crispin  hombre                  0.0   NaN
2                          Fry, Mr. Richard  hombre                  0.0   NaN
3                       Harrison, Mr. William  hombre                  0.0  40.0
5              Parr, Mr. William Henry Marsh  hombre                  0.0   NaN
...                                      ...     ...                  ...   ...
1304                    Sage, Mr. John George  hombre                  0.0   NaN
1305          Sage, Mrs. John (Annie Bullen)   mujer                  0.0   NaN
1306                      Storey, Mr. Thomas  hombre                  0.0  60.5
1307                    Baumann, Mr. John D  hombre                  0.0   NaN
1309                Baxter, Mr. Quigg Edmond  hombre                  0.0  24.0

[807 rows x 4 columns]
```

```
# Se genera una Hoja de Excel por cada DataFrame, y se
# almacenan en un mismo libro (vivos_muertos.xlsx).
with pd.ExcelWriter('vivos_muertos.xlsx') as flujo:
    vivieron.to_excel(flujo, sheet_name='vivos', index=False)
    murieron.to_excel(flujo, sheet_name='muertos', index=False)
```

Haz clic en el ícono de carpeta que aparece en la extrema izquierda de **Google Colab**, y comprueba que los archivos que has guardado están ahí.

FIN DEL LAB

CAPÍTULO 8:
Conversión de datos

LAB 08.01: Ejecutando conversiones de tipo y de moneda

En este Lab se realizan conversiones de tipos para que las columnas se ajusten a los tipos especificados por los códigos DTXC del caso. También se realiza un demo de conversiones de moneda, tomando tipos de cambio desde un servicio de la nube.

Las tareas por realizar son:

1. Realizar conversiones de tipo de dato.
2. Realizar conversiones de tipo de moneda.
1. instalar **`forex_python`**
2. Realizar conversiones de moneda con tipos de cambio en línea.

```
# Datos base
import pandas as pd
personas_titanic_csv='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
personas_titanic.csv'
titanic = pd.read_csv(personas_titanic_csv)
```

Realizar conversiones de tipo de dato.

En el caso, tenemos las siguientes columnas con sus tipos:

Variable	DTXC	Tipo actual
id_persona	QL-NM-STR-ID-AA-NDR	float64
clave_sobrevivencia	QL-NM-STR-CAT-CD-AA-NDR	float64
nombre	QL-NM-STR-DS-AA-NDR	object
clave_sexo	QL-NM-STR-CAT-DES-AA-NDR	object
clase_viaje	QL-OR-STR-CAT-CD-AA-NDR	float64
id_bote	QL-NM-STR-NRID-AA-NDR	object
id_cuerpo	QL-NM-STR-NRID-AA-NDR	float64
edad	QT-CON-NUM-FL-VAL-DET-AA-NDR	float64
cantidad_parientes	QT-DIS-NUM-INT-VAL-DET-AA-NDR	int64
cantidad_familiares	QT-DIS-NUM-INT-VAL-DET-AA-NDR	int64

Para que los campos correspondan a los tipos requeridos por los códigos DTXC, se requieren hacer las siguientes modificaciones.

1. **id_persona** es un identificador, cualitativo nominal.
    a. Pasar de **float** a **object** sin decimales.
2. **clave_sobrevivencia** es un categórico codificado, cualitativo nominal.
    a. Pasar de **float** a **object** sin decimales.
3. **clase_viaje** es un categórico codificado, y es cualitativo ordinal.
    a. Pasar de **float** a **object** sin decimales.
4. **id_cuerpo** es un identificador no requerido, cualitativo nominal.
    a. Pasar de **float** a **object** sin decimales.

```
# Transformación de id_persona
titanic['id_persona']=titanic['id_persona'].astype('str')
titanic['id_persona']=titanic['id_persona'].
      str.split('.').str.get(0)

# Transformación de clave_sobrevivencia
titanic['clave_sobrevivencia']=titanic['clave_sobrevivencia'].
      astype('str')
titanic['clave_sobrevivencia']=titanic['clave_sobrevivencia'].
      str.split('.').str.get(0)

# Transformación de clase_viaje
titanic['clase_viaje']=titanic['clase_viaje'].astype('str')
titanic['clase_viaje']=titanic['clase_viaje'].str.split('.').
      str.get(0)

# Transformación de id_cuerpo
titanic['id_cuerpo']=titanic['id_cuerpo'].astype('str')
titanic['id_cuerpo']=titanic['id_cuerpo'].str.split('.').
      str.get(0)

# Ver los tipos transformados
titanic.dtypes
```

```
id_persona              object
clave_sobrevivencia     object
nombre                  object
sexo                    object
clase_viaje             object
id_bote                 object
id_cuerpo               object
edad                   float64
cantidad_parientes       int64
cantidad_familiares      int64
dtype: object
```

Realizar conversiones de moneda con tipos de cambio en línea

Se instala el paquete **`Forex_python`**.

```
%pip install forex_python
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheel
s/public/simple/
Requirement already satisfied: forex_python in /usr/local/lib/python3.9/dist-packa
ges (1.8)
Requirement already satisfied: simplejson in /usr/local/lib/python3.9/dist-package
s (from forex_python) (3.18.4)
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages
(from forex_python) (2.27.1)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3
.9/dist-packages (from requests->forex_python) (2.0.12)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist
-packages (from requests->forex_python) (2022.12.7)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/d
ist-packages (from requests->forex_python) (1.26.15)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packa
ges (from requests->forex_python) (3.4)
```

Instalado el paquete, se puede usar en el programa.

```
# Se importa la librería para recuperar tipos de cambio.
from forex_python.converter import CurrencyRates

# Creamos el DataFrame de ejemplo.
df = pd.DataFrame({'moneda_origen': ['USD','EUR','GBP','JPY'],
                   'cantidad': [100, 200, 300, 400]})
# Ver los datos.
df
```

```
  moneda_origen  cantidad
0           USD       100
1           EUR       200
2           GBP       300
3           JPY       400
```

```
# Creamos una instancia de CurrencyRates para realizar
# la conversión.
c = CurrencyRates()

# Definimos la moneda a la que queremos convertir las demás,
# que en este caso es euros.
moneda_destino = 'EUR'

# Creamos una función que aplica la conversión a
# cada fila.
def convertir_moneda(fila):
    tasa = c.get_rate(fila['moneda_origen'], moneda_destino)
    return fila['cantidad'] * tasa

# Aplicamos la función a cada fila y almacenamos los resultados
# en una nueva columna
df['equivalente_euros'] = df.apply(convertir_moneda, axis=1)

# Imprimimos el DataFrame resultante
print(df)
```

```
  moneda_origen  cantidad  equivalente_euros
0           USD       100          92.721372
1           EUR       200         200.000000
2           GBP       300         341.199886
3           JPY       400           2.794662
```

FIN DEL LAB

CAPÍTULO 9:

Columnas derivadas o calculadas

LAB 09.01: Cálculos con columnas

En este Lab se desea generar una nueva columna llamada **`acompañantes`**, que sea la suma de **`cantidad_familiares`** y **`cantidad_parientes`**, usando todas las variantes.

Las tareas por realizar son:

1. Generar una nueva columna usando fórmulas.
2. Generar una nueva columna usando UDF's.
   1. Pasando todas las columnas.
   2. Pasando sólo las columnas requeridas.

```
# Datos base
import pandas as pd
personas_titanic_csv='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
personas_titanic_v2.csv'
titanic = pd.read_csv(personas_titanic_csv)
```

Generar una nueva columna usando fórmulas

```
# La tarea, en su forma más sencilla (fórmula simple)
titanic['acompañantes']=(titanic['cantidad_familiares']+
                         titanic['cantidad_parientes'])

# Se comprueba la generación de la suma y la creación
# de la nueva columna. Se genera un DataFrame que contenga
# las filas donde acompañantes sea mayor a cero.
pasajeros_acompañados=titanic.loc[titanic['acompañantes']>0]

# Se muestran los resultados.
pasajeros_acompañados
```

```
      acompañantes  ▶
9                2  ▶
21               2  ▶
33               2  ▶
51               1  ▶
67               1  ▶
...            ...
1302            10  ▶
1303            10  ▶
1304            10  ▶
1305            10  ▶
1309             1  ▶

[519 rows x 11 columns]
```

Generar una nueva columna usando UDF's

1. **Pasando todas las columnas.**

```
# Se declara la función externa.
def calcula_acompañantes(fila):
  return fila['cantidad_familiares']+fila['cantidad_parientes']

# Se invoca la función.
titanic['acompañantes_1']=titanic.apply(calcula_acompañantes,
      axis=1)

# Filtrar para ver solo personas con acompañantes.
pasajeros_acompañados=titanic.loc[titanic['acompañantes']>0]

# Ver el resultado
pasajeros_acompañados
```

```
      acompañantes  acompañantes_1  ▶
9                2               2  ▶
21               2               2  ▶
33               2               2  ▶
51               1               1  ▶
67               1               1  ▶
…
[519 rows x 12 columns]
```

2. **Pasando sólo las columnas requeridas.**

```
# Se declara la función externa.
def calcula_acompañantes(fila):
  return fila['cantidad_familiares']+fila['cantidad_parientes']

# Se invoca la función, pasando sólo algunas columnas.
titanic['acompañantes_2']=titanic[['cantidad_familiares',
     'cantidad_parientes']].apply(calcula_acompañantes,axis=1)

# Filtrar para ver solo personas con acompañantes.
pasajeros_acompañados=titanic.loc[titanic['acompañantes']>0]

# Ver el resultado
pasajeros_acompañados
```

```
      acompañantes  acompañantes_1  acompañantes_2  ▶
9                2               2               2  ▶
21               2               2               2  ▶
33               2               2               2  ▶
51               1               1               1  ▶
67               1               1               1  ▶
...            ...             ...             ...
1302            10              10              10  ▶
1303            10              10              10  ▶
1304            10              10              10  ▶
1305            10              10              10  ▶
1309             1               1               1  ▶
[519 rows x 13 columns]
```

Generar una nueva columna usando funciones anónimas (lamda)

```
# Se aplica la función lambda usando apply()
titanic['acompañantes_3']=titanic.apply(lambda x:
    x['cantidad_familiares'] +
    x['cantidad_parientes'], axis=1)

# Filtrar para ver solo personas con acompañantes.
pasajeros_acompañados=titanic.loc[titanic['acompañantes']>0]

# Ver el resultado
pasajeros_acompañados
```

```
      acompañantes  acompañantes_1  acompañantes_2  acompañantes_3  ▶
9                2               2               2               2  ▶
21               2               2               2               2  ▶
33               2               2               2               2  ▶
51               1               1               1               1  ▶
67               1               1               1               1  ▶
...            ...             ...             ...             ...
1302            10              10              10              10  ▶
1303            10              10              10              10  ▶
1304            10              10              10              10  ▶
1305            10              10              10              10  ▶
1309             1               1               1               1  ▶
[519 rows x 14 columns]
```

```
# Se eliminan los campos que no son requridos del DataFrame
# titanic
titanic.drop(columns=['acompañantes_1','acompañantes_2',
            'acompañantes_3'],
            inplace=True)
```

FIN DEL LAB

CAPÍTULO 10:

Transformación de cadenas

LAB 10.01: Cálculos con columnas

En este Lab se desea generar nuevas columnas, o transformar las existentes, aplicando técnicas de cálculo de valores a partir de los datos existentes en el DataFrame.

Las tareas por realizar son:

1. Eliminar las advertencias de Python.
2. Modificar el *casing* de las columnas, usando **`lower()`**, **`upper()`**, y **`title()`**.
3. Derivar columnas por separación, usando **`split()`**.
    a. Alternativa usando **`split()`**.
    b. Alternativa usando **`split()`** con **`extend`**.
    c. Alternativa usando **`loc[]`**.
4. Extraer el título de la persona.
5. Eliminar los espacios en los extremos, usando **`strip()`**.
6. Derivar columnas por concatenación.
7. Derivar columnas por extracción.

Habilita las advertencias de Python

```
# Datos base
import pandas as pd
personas_titanic_csv='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
personas_titanic_v3.csv'
titanic = pd.read_csv(personas_titanic_csv)
```

Eliminar las advertencias de Python

```
# Se inhabilitan temporalmente las advertencias de Python.
# Se establece un filtro de advertencias, para que las ignore.
import warnings
warnings.filterwarnings("ignore")
```

Modificar el casing de las columnas, usando lower(), upper(), y title()

Modifica el casing de las siguientes columnas:

1. **sexo**: Pasarlo todo a mayúsculas.
2. **nombre**: Pasarlo a primera letra de cada palabra en mayúscula y el resto a minúscula (ya está así, pero no en todos los casos).

```
# Se revisa el estado actual de los datos, en cuanto a casing.
titanic[['sexo','nombre']]
```

```
        sexo                                    nombre
0     hombre                  Andrews, Mr. Thomas Jr
1     hombre  Chisholm, Mr. Roderick Robert Crispin
2     hombre                       Fry, Mr. Richard
3     hombre                 Harrison, Mr. William
4     hombre               Ismay, Mr. Joseph Bruce
...      ...                                     ...
1305   mujer        Sage, Mrs. John (Annie Bullen)
1306  hombre                     Storey, Mr. Thomas
1307  hombre                   Baumann, Mr. John D
1308  hombre          Blackwell, Mr. Stephen Weart
1309  hombre              Baxter, Mr. Quigg Edmond
[1310 rows x 2 columns]
```

```
# Se pasa sexo a mayúsculas.
titanic['sexo']=titanic['sexo'].str.upper()

# Se pasa nombre a mayúscula la primera letra de cada palabra.
titanic['nombre']=titanic['nombre'].str.title()

# Se revisa el estado actual de los datos, en cuanto a casing.
titanic[['sexo','nombre']]
```

```
        sexo                                  nombre
0     HOMBRE                 Andrews, Mr. Thomas Jr
1     HOMBRE  Chisholm, Mr. Roderick Robert Crispin
2     HOMBRE                       Fry, Mr. Richard
3     HOMBRE                  Harrison, Mr. William
4     HOMBRE               Ismay, Mr. Joseph Bruce
...      ...                                   ...
1305   MUJER         Sage, Mrs. John (Annie Bullen)
1306  HOMBRE                     Storey, Mr. Thomas
1307  HOMBRE                   Baumann, Mr. John D
1308  HOMBRE            Blackwell, Mr. Stephen Weart
1309  HOMBRE                Baxter, Mr. Quigg Edmond
[1310 rows x 2 columns]
```

Derivar columnas por separación, usando split()

Realiza las siguientes tareas:

1. Genera un DataFrame de trabajo llamado **cadenas**, que contenga solo los campos **id_persona**, **nombre**, y **sexo** del DataFrame **titanic**.
2. A partir de **nombre**, usando como separador la coma, genera dos columnas: **apellidos**, con la primera parte de la cadena, y **primer_nombre**, con la segunda parte de la cadena. Utiliza varias estrategias.
3. A partir de **primer_nombre**, extrae el título de la persona. Utiliza la estrategia que quieras.

```
# Se genera un nuevo DataFrame de trabajo llamado cadenas
cadenas=titanic[['id_persona','nombre','sexo']]

cadenas
```

```
      id_persona                                  nombre    sexo
0          10000                 Andrews, Mr. Thomas Jr  HOMBRE
1          10001  Chisholm, Mr. Roderick Robert Crispin  HOMBRE
2          10002                       Fry, Mr. Richard  HOMBRE...          ...
...      ...
1307       11307                   Baumann, Mr. John D  HOMBRE
1308       11308            Blackwell, Mr. Stephen Weart  HOMBRE
1309       11309                Baxter, Mr. Quigg Edmond  HOMBRE
[1310 rows x 3 columns]
```

1. **Alternativa usando `split()`.**

```
# Se derivan nuevas columnas por separación, usando split,
# extrayendo elemento por elemento.
# La primera parte de la cadena (índice 0) formará la
# columna apellidos.
# La segunda parte de la cadena (índice 1) formará la
# columna primer_nombre.
cadenas['apellidos']=cadenas['nombre'].
     str.split(',').str.get(0)
cadenas['primer_nombre']=cadenas['nombre'].
      str.split(',').str.get(1)

# Se muestra el resultado
cadenas
```

```
      id_persona                                    nombre    sexo  apellidos  \
0          10000                 Andrews, Mr. Thomas Jr  HOMBRE    Andrews
1          10001  Chisholm, Mr. Roderick Robert Crispin  HOMBRE   Chisholm
2          10002                       Fry, Mr. Richard  HOMBRE        Fry
3          10003                  Harrison, Mr. William  HOMBRE   Harrison
4          10004                 Ismay, Mr. Joseph Bruce  HOMBRE      Ismay
...          ...                                    ...     ...        ...
1305       11305         Sage, Mrs. John (Annie Bullen)   MUJER       Sage
1306       11306                    Storey, Mr. Thomas  HOMBRE     Storey
1307       11307                   Baumann, Mr. John D  HOMBRE    Baumann
1308       11308           Blackwell, Mr. Stephen Weart  HOMBRE  Blackwell
1309       11309               Baxter, Mr. Quigg Edmond  HOMBRE     Baxter

                    primer_nombre
0                    Mr. Thomas Jr
1      Mr. Roderick Robert Crispin
2                      Mr. Richard
3                      Mr. William
4                 Mr. Joseph Bruce
...                            ...
1305      Mrs. John (Annie Bullen)
1306                    Mr. Thomas
1307                    Mr. John D
1308             Mr. Stephen Weart
1309              Mr. Quigg Edmond
[1310 rows x 5 columns]
```

2. Alternativa usando `split()` con `extend`.

```
# Otra alternativa, en una sola línea, usando expand=True.
# Se coloca una lista de columnas a crear, mismas que se
# poblarán con las correspondientes porciones de la cadena,
# en el orden en que aparecen.
cadenas[['apellidos','primer_nombre']]=cadenas['nombre'].
      str.split(',', expand=True)

# Se muestra el resultado
cadenas
```

```
      id_persona                                  nombre    sexo  apellidos  \
0          10000                 Andrews, Mr. Thomas Jr  HOMBRE    Andrews
1          10001  Chisholm, Mr. Roderick Robert Crispin  HOMBRE   Chisholm
2          10002                        Fry, Mr. Richard  HOMBRE        Fry
3          10003                  Harrison, Mr. William  HOMBRE   Harrison
4          10004                Ismay, Mr. Joseph Bruce  HOMBRE      Ismay
...          ...                                    ...     ...        ...
1305       11305         Sage, Mrs. John (Annie Bullen)   MUJER       Sage
1306       11306                   Storey, Mr. Thomas  HOMBRE     Storey
1307       11307                  Baumann, Mr. John D  HOMBRE    Baumann
1308       11308           Blackwell, Mr. Stephen Weart  HOMBRE  Blackwell
1309       11309               Baxter, Mr. Quigg Edmond  HOMBRE     Baxter

                    primer_nombre
0                    Mr. Thomas Jr
1      Mr. Roderick Robert Crispin
2                      Mr. Richard
3                      Mr. William
4                Mr. Joseph Bruce
...                           ...
1305      Mrs. John (Annie Bullen)
1306                     Mr. Thomas
1307                     Mr. John D
1308              Mr. Stephen Weart
1309               Mr. Quigg Edmond
[1310 rows x 5 columns]
```

3. Alternativa usando `loc[]`.

```
# Otra alternativa, usando loc[]
cadenas.loc[:,'apellidos']=cadenas['nombre'].
      str.split(',').str.get(0)
cadenas.loc[:,'primer_nombre']=cadenas['nombre'].
      str.split(',').str.get(1)

# Se muestra el resultado
cadenas
```

```
      id_persona                                  nombre    sexo  apellidos  \
0          10000                 Andrews, Mr. Thomas Jr  HOMBRE    Andrews
1          10001  Chisholm, Mr. Roderick Robert Crispin  HOMBRE   Chisholm
2          10002                      Fry, Mr. Richard  HOMBRE        Fry
3          10003                Harrison, Mr. William  HOMBRE   Harrison
4          10004               Ismay, Mr. Joseph Bruce  HOMBRE      Ismay
...          ...                                  ...     ...        ...
1305       11305        Sage, Mrs. John (Annie Bullen)   MUJER       Sage
1306       11306                  Storey, Mr. Thomas  HOMBRE     Storey
1307       11307                 Baumann, Mr. John D  HOMBRE    Baumann
1308       11308           Blackwell, Mr. Stephen Weart  HOMBRE  Blackwell
1309       11309               Baxter, Mr. Quigg Edmond  HOMBRE     Baxter

                    primer_nombre
0                    Mr. Thomas Jr
1      Mr. Roderick Robert Crispin
2                      Mr. Richard
3                      Mr. William
4                Mr. Joseph Bruce
...                           ...
1305      Mrs. John (Annie Bullen)
1306                     Mr. Thomas
1307                     Mr. John D
1308              Mr. Stephen Weart
1309               Mr. Quigg Edmond

[1310 rows x 5 columns]
```

Extraer el título de la persona

```
# Para obtener el título de la persona (Miss, Mr, Mrs,
# Dr, etc), y colocarlo en una columna llamada titulo.
# Se divide a partir del punto, y se extrae la primera porción.
cadenas['titulo']=cadenas['primer_nombre'].
      str.split('.').str.get(0)

# Se muestra el resultado
cadenas
```

```
      id_persona                                nombre    sexo  apellidos  \
0          10000                 Andrews, Mr. Thomas Jr  HOMBRE    Andrews
1          10001  Chisholm, Mr. Roderick Robert Crispin  HOMBRE   Chisholm
2          10002                       Fry, Mr. Richard  HOMBRE        Fry
3          10003                  Harrison, Mr. William  HOMBRE   Harrison
4          10004               Ismay, Mr. Joseph Bruce  HOMBRE      Ismay
...          ...                                   ...     ...        ...
1305       11305         Sage, Mrs. John (Annie Bullen)   MUJER       Sage
1306       11306                     Storey, Mr. Thomas  HOMBRE     Storey
1307       11307                   Baumann, Mr. John D  HOMBRE    Baumann
1308       11308          Blackwell, Mr. Stephen Weart  HOMBRE  Blackwell
1309       11309              Baxter, Mr. Quigg Edmond  HOMBRE     Baxter

                    primer_nombre titulo
0                     Mr. Thomas Jr     Mr
1      Mr. Roderick Robert Crispin      Mr
2                       Mr. Richard     Mr
3                       Mr. William     Mr
4                 Mr. Joseph Bruce      Mr
...                            ...    ...
1305      Mrs. John (Annie Bullen)    Mrs
1306                    Mr. Thomas      Mr
1307                    Mr. John D      Mr
1308             Mr. Stephen Weart      Mr
1309              Mr. Quigg Edmond      Mr

[1310 rows x 6 columns]
```

Eliminar los espacios en los extremos, usando strip()

Realiza las siguientes tareas:

1. Elimina espacios en los extremos de las columnas **apellidos** y **primer_nombre**.
2. Utiliza el DataFrame **cadenas** y la función **strip()**.

```
# Se eliminan los espacios en los extremos de las
# columnas indicadas.
cadenas['apellidos']=cadenas['apellidos'].str.strip()
cadenas['primer_nombre']=cadenas['primer_nombre'].str.strip()

# Se muestra el resultado.
cadenas
```

```
      id_persona                                  nombre    sexo  apellidos  \
0          10000                   Andrews, Mr. Thomas Jr  HOMBRE    Andrews
1          10001  Chisholm, Mr. Roderick Robert Crispin  HOMBRE   Chisholm
2          10002                         Fry, Mr. Richard  HOMBRE        Fry
3          10003                    Harrison, Mr. William  HOMBRE   Harrison
4          10004                  Ismay, Mr. Joseph Bruce  HOMBRE      Ismay
...          ...                                     ...     ...        ...
1305       11305          Sage, Mrs. John (Annie Bullen)   MUJER       Sage
1306       11306                     Storey, Mr. Thomas  HOMBRE     Storey
1307       11307                    Baumann, Mr. John D  HOMBRE    Baumann
1308       11308             Blackwell, Mr. Stephen Weart  HOMBRE  Blackwell
1309       11309                 Baxter, Mr. Quigg Edmond  HOMBRE     Baxter

                    primer_nombre titulo
0                    Mr. Thomas Jr     Mr
1     Mr. Roderick Robert Crispin     Mr
2                       Mr. Richard     Mr
3                       Mr. William     Mr
4                 Mr. Joseph Bruce     Mr
...                            ...    ...
1305     Mrs. John (Annie Bullen)    Mrs
1306                      Mr. Thomas     Mr
1307                      Mr. John D     Mr
1308               Mr. Stephen Weart     Mr
1309                Mr. Quigg Edmond     Mr

[1310 rows x 6 columns]
```

Derivar columnas por concatenación

Realiza las siguientes tareas:

1. Concatena los campos **`primer_nombre`** y **`apellidos`**, en una nueva columna llamada **`nombre_completo`**.
2. Ten cuidado de agregar un espacio intermedio entre ambas columnas, para que el contenido no quede junto.
3. Utiliza el DataFrame **`cadenas`**.

```
# Se concatenan las columnas deseadas (pueden generarse
# advertencias).
cadenas['nombre_completo']=cadenas['primer_nombre']+
      ' '+cadenas['apellidos']

# Se muestra el resultado.
cadenas
```

```
      id_persona                                  nombre    sexo  apellidos  \
0          10000                   Andrews, Mr. Thomas Jr  HOMBRE    Andrews
1          10001  Chisholm, Mr. Roderick Robert Crispin  HOMBRE   Chisholm
2          10002                         Fry, Mr. Richard  HOMBRE        Fry
3          10003                   Harrison, Mr. William  HOMBRE   Harrison
4          10004                 Ismay, Mr. Joseph Bruce  HOMBRE      Ismay
...          ...                                     ...     ...        ...
1305       11305         Sage, Mrs. John (Annie Bullen)   MUJER       Sage
1306       11306                   Storey, Mr. Thomas  HOMBRE     Storey
1307       11307                  Baumann, Mr. John D  HOMBRE    Baumann
1308       11308            Blackwell, Mr. Stephen Weart  HOMBRE  Blackwell
1309       11309                Baxter, Mr. Quigg Edmond  HOMBRE     Baxter

                    primer_nombre titulo                       nombre_completo
0                   Mr. Thomas Jr     Mr                 Mr. Thomas Jr Andrews
1     Mr. Roderick Robert Crispin     Mr  Mr. Roderick Robert Crispin Chisholm
2                     Mr. Richard     Mr                        Mr. Richard Fry
3                     Mr. William     Mr                  Mr. William Harrison
4                Mr. Joseph Bruce     Mr                Mr. Joseph Bruce Ismay
...                           ...    ...                                   ...
1305     Mrs. John (Annie Bullen)    Mrs         Mrs. John (Annie Bullen) Sage
1306                   Mr. Thomas     Mr                      Mr. Thomas Storey
1307                   Mr. John D     Mr                    Mr. John D Baumann
1308            Mr. Stephen Weart     Mr           Mr. Stephen Weart Blackwell
1309             Mr. Quigg Edmond     Mr                Mr. Quigg Edmond Baxter

[1310 rows x 7 columns]
```

Derivar columnas por extracción

Realiza las siguientes tareas:

1. Genera una nueva columna llamada **`clave_sexo`**, que contenga la primera letra de la columna **`sexo`**.
2. Trabaja con el DataFrame **cadenas**.
3. Utiliza *slices* para hacer la extracción de la subcadena requerida.

```
# Se extrae el primer símbolo de la cadena, usando slicers
# y se genera una nueva columna (puede hacer advertencias).
cadenas['clave_sexo']=cadenas['sexo'].str[0]

# Se ven los resultados
cadenas
```

```
      id_persona                                  nombre    sexo  apellidos  \
0          10000                   Andrews, Mr. Thomas Jr  HOMBRE    Andrews
1          10001  Chisholm, Mr. Roderick Robert Crispin  HOMBRE   Chisholm
2          10002                         Fry, Mr. Richard  HOMBRE        Fry
3          10003                    Harrison, Mr. William  HOMBRE   Harrison
4          10004                  Ismay, Mr. Joseph Bruce  HOMBRE      Ismay
...          ...                                     ...     ...        ...
1305       11305          Sage, Mrs. John (Annie Bullen)   MUJER       Sage
1306       11306                      Storey, Mr. Thomas  HOMBRE     Storey
1307       11307                     Baumann, Mr. John D  HOMBRE    Baumann
1308       11308             Blackwell, Mr. Stephen Weart  HOMBRE  Blackwell
1309       11309                 Baxter, Mr. Quigg Edmond  HOMBRE     Baxter


                          nombre_completo clave_sexo
0                    Mr. Thomas Jr Andrews          H
1     Mr. Roderick Robert Crispin Chisholm          H
2                          Mr. Richard Fry          H
3                     Mr. William Harrison          H
4                   Mr. Joseph Bruce Ismay          H
...                                    ...        ...
1305         Mrs. John (Annie Bullen) Sage          M
1306                    Mr. Thomas Storey          H
1307                   Mr. John D Baumann          H
1308           Mr. Stephen Weart Blackwell          H
1309               Mr. Quigg Edmond Baxter          H

[1310 rows x 8 columns]
```

Habilita las advertencias de Python

```
# Se habilitan de nuevo las advertencias de Python
warnings.resetwarnings()
```

FIN DEL LAB

CAPÍTULO 11:

Tratamiento de categóricos e integración de datos

LAB 11.01: Generación de categóricos descriptivos equivalentes

En este Lab se desea generar nuevas columnas, o transformar las existentes, aplicando técnicas de cálculo de valores a partir de los datos existentes en el DataFrame.

Las tareas por realizar son:

1. Generación de categóricos usando **`map()`**.
2. Generación de categóricos a partir de otros archivos, usando **`merge()`**.
3. Generación de categóricos usando UDF's.

```
# Datos base
import pandas as pd
tipos_correctos={
    'id_persona':object,
    'clave_sobrevivencia':object,
    'clase_viaje':object
}
personas_titanic_csv='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
personas_titanic_v3.csv'
titanic = pd.read_csv(personas_titanic_csv,
      dtype=tipos_correctos)
# Al momento de la lectura especificamos que los valores
# que son categóricos, los interprete como object, y no como
# enteros o flotantes.
# Es importante hacer notar que si pandas detecta datos
# que no sabe cómo convertir, por ejemplo NaN, proporcionará
# un tipo de datos más amplio, y por eso pone columnas
# que son enteras (como clave_sobrevivencia) como flotante,
# y al convertir a object será '0.0'.

# Para eliminar los '.0' de sobra, se aplica lo siguiente.

titanic['id_persona']=titanic['id_persona'].
      str.split('.').str.get(0)
titanic['clave_sobrevivencia']=titanic['clave_sobrevivencia'].
      str.split('.').str.get(0)
titanic['clase_viaje']=titanic['clase_viaje'].
      str.split('.').str.get(0)
```

Generación de categóricos usando map()

Realiza las siguientes tareas para generar un categórico descriptivo llamado **`sobrevivencia`**.

1. Toma como base el campo **`clave_sobrevivencia`**.
2. Si **`clave_sobrevivencia`** es **`'0'`**, es **`'MURIÓ'`**; si es **`'1'`**, es **`'VIVIÓ'`**.
3. Usa **`map()`**.

```
# Se definen las equivalencias para sobrevivencia
# usando un diccionario, donde la llave es el valor
# de referencia, y el valor es el categórico descriptivo.

catalogo_sobrevivencia={
    '0':'MURIÓ',
    '1':'VIVIÓ'
}

# Se genera una nueva columna, con las equivalencias
# categóricas, usando map sobre el campo a considerar
# como base para el cálculo proporcionando como parámetro
# el diccionario que contiene las equivalencias.
titanic['sobrevivencia']=titanic['clave_sobrevivencia'].
      map(catalogo_sobrevivencia)

# Se muestra el resultado.
titanic[['nombre','clave_sobrevivencia','sobrevivencia']]
```

```
                                 nombre clave_sobrevivencia sobrevivencia
0                  Andrews, Mr. Thomas Jr                  0         MURIÓ
1     Chisholm, Mr. Roderick Robert Crispin                  0         MURIÓ
2                         Fry, Mr. Richard                  0         MURIÓ
3                    Harrison, Mr. William                  0         MURIÓ
4                  Ismay, Mr. Joseph Bruce                  1         VIVIÓ
...                                    ...                ...           ...
1305        Sage, Mrs. John (Annie Bullen)                  0         MURIÓ
1306                    Storey, Mr. Thomas                  0         MURIÓ
1307                   Baumann, Mr. John D                  0         MURIÓ
1308          Blackwell, Mr. Stephen Weart                NaN           NaN
1309              Baxter, Mr. Quigg Edmond                  0         MURIÓ

[1310 rows x 3 columns]
```

Generación de categóricos a partir de otros archivos, usando merge()

Realiza las siguientes tareas para generar un categórico descriptivo a partir de los datos de un archivo que contiene un catálogo de clases.

1. Carga en un DataFrame llamado **`catalogo_clases`** el contenido del archivo **`clases.csv`**.
2. Usa **`merge()`** para agregar los datos de **`catalogo_clases`** al DataFrame **`titanic`**.
3. Toma como campo de coincidencia **`clase_viaje`**.

```
# Se importan las clases del catálogo, desde GitHub.

catalogo_clases = pd.read_csv('https://raw.githubusercontent.
com/AprendaPracticando/AnaliticaPythonR1/main/data/clases.csv')

catalogo_clases
```

```
   clase_viaje          clase
0            1  PRIMERA CLASE
1            2  SEGUNDA CLASE
2            3  TERCERA CLASE
```

```
# Revisamos los tipos de los atributos de coincidencia.
# Vemos que no coinciden.
print(titanic.dtypes['clase_viaje'])
print(catalogo_clases.dtypes['clase_viaje'])
```

```
object
int64
```

```
# Cambiamos el tipo de dato del DataFrame de catálogo,
# para homologar tipos.
catalogo_clases['clase_viaje']=catalogo_clases['clase_viaje'].
      astype('str')

# Se hace el merge, de tipo outer join, para que no elimine
# registros en caso de que clase sea un dato ausente.
titanic=titanic.merge(catalogo_clases,on='clase_viaje',
      how='outer')

# Se muestran los resultados
titanic[['nombre','clase_viaje','clase']]
```

```
                                  nombre clase_viaje          clase
0                    Andrews, Mr. Thomas Jr           1  PRIMERA CLASE
1     Chisholm, Mr. Roderick Robert Crispin           1  PRIMERA CLASE
2                           Fry, Mr. Richard           1  PRIMERA CLASE
3                      Harrison, Mr. William           1  PRIMERA CLASE
4                    Ismay, Mr. Joseph Bruce           1  PRIMERA CLASE
...                                      ...         ...            ...
1305         Sage, Mrs. John (Annie Bullen)           3  TERCERA CLASE
1306                      Storey, Mr. Thomas           3  TERCERA CLASE
1307                    Baumann, Mr. John D         NaN            NaN
1308           Blackwell, Mr. Stephen Weart         NaN            NaN
1309               Baxter, Mr. Quigg Edmond         NaN            NaN

[1310 rows x 3 columns]
```

Generación de categóricos usando UDF's.

Realiza las siguientes tareas, para generar un categórico descriptivo llamado **`acompañada`**.

1. Toma como base el campo **`acompañantes`**.
2. Si **`acompañantes`** es mayor a cero, debe ser **`'ACOMPAÑADA'`**, y si es cero, debe ser **`'SOLA'`**.
3. Usa funciones definidas por el usuario.

```
# Se define la función que evalúa la cantidad de acompañantes
# y retorna el categórico descriptivo correspondiente.
def acompañamiento(x):
  etiqueta='NO DEFINIDO'
  if (x['acompañantes']==0):
    etiqueta='SOLA'
  if (x['acompañantes']>0):
    etiqueta='ACOMPAÑADA'
  return etiqueta

# Se aplica la función a cada filas.
titanic['acompañada']=titanic.apply(acompañamiento, axis=1)

# Se ven los resultados.
titanic[['nombre','acompañantes','acompañada']]
```

```
                                 nombre  acompañantes  acompañada
0                    Andrews, Mr. Thomas Jr             0        SOLA
1     Chisholm, Mr. Roderick Robert Crispin             0        SOLA
2                           Fry, Mr. Richard             0        SOLA
3                     Harrison, Mr. William              0        SOLA
4                   Ismay, Mr. Joseph Bruce              0        SOLA
...                                    ...            ...         ...
1305         Sage, Mrs. John (Annie Bullen)            10  ACOMPAÑADA
1306                     Storey, Mr. Thomas             0        SOLA
1307                    Baumann, Mr. John D             0        SOLA
1308           Blackwell, Mr. Stephen Weart              0        SOLA
1309               Baxter, Mr. Quigg Edmond              1  ACOMPAÑADA

[1310 rows x 3 columns]
```

FIN DEL LAB

LAB 11.02: Generación de categóricos de intervalo

En este Lab se desea generar, paso a paso, un categórico de intervalo.

Las tareas por realizar son:

1. Generar rangos de intervalos estándar, usando **`cut()`**.
2. Generar una tabla de frecuencia absoluta usando **`value_counts()`**.
3. Establecer manualmente los límites de intervalo (**`bins`**).
4. Cambiar intervalos a semi-cerrado a la derecha **`[x,y)`** (**`right`**).
5. Definir las etiquetas de intervalo (**`labels`**).
6. Generación de categórico de intervalo usando UDF's.

```
# Datos base
import pandas as pd
tipos_correctos={
    'id_persona':object,
    'clave_sobrevivencia':object,
    'clase_viaje':object
}
personas_titanic_csv='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
personas_titanic_v4.csv'
titanic = pd.read_csv(personas_titanic_csv,
      dtype=tipos_correctos)

titanic['id_persona']=titanic['id_persona'].str.split('.').
      str.get(0)
titanic['clave_sobrevivencia']=titanic['clave_sobrevivencia'].
      str.split('.').str.get(0)
titanic['clase_viaje']=titanic['clase_viaje'].str.split('.').
      str.get(0)
```

Generar rangos de intervalos estándar, usando cut()

En el caso de ejemplo, queremos generar un categórico descriptivo equivalente al campo **rango_edad**, tomando como base los valores de la columna **edad**, de acuerdo con la siguiente tabla.

Como puede apreciarse, se trata de intervalos semi-cerrados a la derecha.

Categoría	Rango
INFANTES	[0,12)
JÓVENES	[12,21)
ADULTOS	[21,35)
MEDIANA EDAD	[35,45)
ADULTOS MAYORES	[45,∞)

Lo que se terminará haciendo es lo siguiente:

1. Evoluciona el proceso, para entender el uso de la función **cut()**.
2. Genera 5 (**k=5**) rangos de intervalo (*clases*), a partir de los valores de **edad**, en una nueva columna llamada **rango_edad**.
3. Genera una tabla de frecuencias para la columna **rango_edad**, usando la función **value_counts()**.
4. Cambiar de semi-cerrado a la derecha, a semi-cerrado a la izquierda, es decir, de **(a,b]** a **[a,b)**.
5. Establece valores específicos como límite de intervalo.
6. Establece las etiquetas correctas para las clases.

```
# Generar 5 intervalos, con un número de clases igual a 5.
k=5
titanic['rango_edad']=pd.cut(titanic['edad'],
                             bins=k)

# Ver el resultado, mostrando id_persona, edad y rango_edad
titanic[['id_persona', 'edad', 'rango_edad']]
```

```
     id_persona  edad        rango_edad
0         10000  39.0    (32.1, 48.067]
1         10001   NaN               NaN
```

```
2         10002   NaN                NaN
3         10003  40.0    (32.1, 48.067]
4         10004  49.0  (48.067, 64.033]
...         ...   ...                ...
1307      11307   NaN                NaN
1308      11308   NaN                NaN
1309      11309  24.0    (16.133, 32.1]
[1310 rows x 3 columns]
```

Lo que hizo pandas fue dividir la escala en 5 partes iguales **k=5**, determinando automáticamente el límite inferior y límite superior de cada clase.

Como puedes apreciar, pandas determinó los límites en un formato **(a,b]**, (semi-cerrado a la derecha), que quiere decir que para cada clase, están incluidos los valores mayores al límite inferior, y menores o iguales al límite superior. Pandas analiza el valor de **edad**, y establece el **rango_edad** que le corresponde.

El valor asignado es una etiqueta, pero al no tener más información de esta, se limita a colocar la notación correspondiente al rango de intervalo.

Generar una tabla de frecuencia absoluta usando value_counts()

Podemos usar el método **value_counts()**, que genera una tabla de frecuencias, y que es muy útil para analizar campos categóricos.

```
# Imprime la frecuencia absoluta de cada clase generada
titanic['rango_edad'].value_counts()
```

```
(16.133, 32.1]      525
(32.1, 48.067]      268
(0.0869, 16.133]    134
(48.067, 64.033]    106
(64.033, 80.0]       13

Name: rango_edad, dtype: int64
```

Cambiar intervalos a semi-cerrado a la derecha [a,b) (right).

Ahora, genera los rangos de intervalo, de tal manera que queden en un modo semi-cerrado a la derecha **[a,b)**, es decir, donde la clase contenga los valores mayores o iguales al valor inicial, y menores al valor final.

Muestra la tabla de frecuencias, para validar.

```
# Generar 5 intervalos, con un número de clases igual a 5,
# donde se tenga un modo [x,y), es decir, donde la clase
# contenga los valores mayores o iguales al valor inicial,
# y menores al valor final.
k=5
titanic['rango_edad']=pd.cut(titanic['edad'],
                             bins=k,
                             right=False)

# Imprime la frecuencia absoluta de cada clase generada
titanic['rango_edad'].value_counts()
```

```
[16.133, 32.1)      525
[32.1, 48.067)      268
[0.167, 16.133)     134
[48.067, 64.033)    106
[64.033, 80.08)      13

Name: rango_edad, dtype: int64
```

Establecer manualmente los límites de intervalo (bins)

Ahora, toma el control de los límites de los intervalos, para que se ajusten a las siguientes clases.

Categoría	Rango
INFANTES	[0,12)
JÓVENES	[12,21)
ADULTOS	[21,35)
MEDIANA EDAD	[35,45)
ADULTOS MAYORES	[45,∞)

```
# Determinamos el mímino y el máximo del rango.
# A máximo le sumamos 1, para que, en caso de haber
# edades iguales al máximo (cosa que va a suceder),
# no queden fuera del análisis, debido a que el modo
# que estamos usando excluye el límite superior.
mínimo=titanic['edad'].min()
máximo=titanic['edad'].max()+1
```

```
# Se colocan en una lista los valores de referencia para
# los rangos de valores.
# Ya no es necesario especificar k, porque inferirá el número
# a partir de los valores límite que pongamos en la lista,
# menos 1.
limites=[mínimo,12,21,35,45,máximo]

# Se generan de nuevo los rangos de intervalo.
titanic['rango_edad']=pd.cut(titanic['edad'],
                             bins=limites,
                             right=False)

# Imprime la frecuencia absoluta de cada clase generada
titanic['rango_edad'].value_counts()
```

```
[21.0, 35.0)     453
[45.0, 81.0)     175
[35.0, 45.0)     169
[12.0, 21.0)     158
[0.167, 12.0)     91
Name: rango_edad, dtype: int64
```

Definir las etiquetas de intervalo (labels)

Ahora, ajusta las etiquetas para las clases.

```
# Almacena las etiquetas de las clases en una lista.
etiquetas_clase=['INFANTES','JÓVENES','ADULTOS',
      'MEDIANA EDAD','ADULTOS MAYORES']

mínimo=titanic['edad'].min()
máximo=titanic['edad'].max()+1
limites=[mínimo,12,21,35,45,máximo]

# Se generan de nuevo los rangos de intervalo.
titanic['rango_edad']=pd.cut(titanic['edad'],
                             bins=limites,
                             right=False,
                             labels=etiquetas_clase)

# Imprime la frecuencia absoluta de cada clase generada
titanic['rango_edad'].value_counts()
```

```
ADULTOS            453
ADULTOS MAYORES    175
MEDIANA EDAD       169
JÓVENES            158
INFANTES            91
Name: rango_edad, dtype: int64
```

```
# Comprobamos que se ha creado una columna categórica
# descriptiva, calculada en función a la columna edad
titanic[['nombre','edad','rango_edad']]
```

```
                                  nombre  edad       rango_edad
0                    Andrews, Mr. Thomas Jr  39.0     MEDIANA EDAD
1     Chisholm, Mr. Roderick Robert Crispin   NaN              NaN
2                          Fry, Mr. Richard   NaN              NaN
3                      Harrison, Mr. William  40.0     MEDIANA EDAD
4                    Ismay, Mr. Joseph Bruce  49.0  ADULTOS MAYORES
...                                      ...   ...              ...
1305         Sage, Mrs. John (Annie Bullen)   NaN              NaN
1306                     Storey, Mr. Thomas  60.5  ADULTOS MAYORES
1307                   Baumann, Mr. John D   NaN              NaN
1308           Blackwell, Mr. Stephen Weart   NaN              NaN
1309               Baxter, Mr. Quigg Edmond  24.0          ADULTOS

[1310 rows x 3 columns]
```

Generación de categóricos usando UDF's

Otra alternativa que se tiene es codificar una función... esto es útil cuando la lógica condicional de clasificación tiene excepciones especiales que no se pueden manejar con **`cut()`**.

```
def RangoEdad(x):
  categoria=None
  if (x['edad']>=0.00 and x['edad']<12.00): categoria='INFANTES'
  if (x['edad']>=12.00 and x['edad']<21.00): categoria='JOVENES'
  if (x['edad']>=21.00 and x['edad']<35.00): categoria='ADULTOS'
  if (x['edad']>=35.00 and x['edad']<45.00): categoria='MEDIANA EDAD'
  if (x['edad']>=45.00): categoria='ADULTOS MAYORES'
  return categoria

titanic['rango_edad']=titanic.apply(RangoEdad,axis=1)

titanic[['nombre','edad','rango_edad']]
```

```
                                     nombre  edad       rango_edad
0                       Andrews, Mr. Thomas Jr  39.0     MEDIANA EDAD
1     Chisholm, Mr. Roderick Robert Crispin   NaN             None
2                            Fry, Mr. Richard   NaN             None
3                        Harrison, Mr. William  40.0     MEDIANA EDAD
4                     Ismay, Mr. Joseph Bruce  49.0  ADULTOS MAYORES
...                                       ...   ...              ...
1305         Sage, Mrs. John (Annie Bullen)   NaN             None
1306                       Storey, Mr. Thomas  60.5  ADULTOS MAYORES
1307                    Baumann, Mr. John D   NaN             None
1308            Blackwell, Mr. Stephen Weart   NaN             None
1309                Baxter, Mr. Quigg Edmond  24.0          ADULTOS

[1310 rows x 3 columns]
```

FIN DEL LAB

LAB 11.03: Integración de datos con Python y pandas

En este Lab se utilizarán todas las técnicas aprendidas para realizar una integración de datos. A partir de un conjunto de archivo dispersos, se integrará un master de datos que pueda ser utilizado para trabajos de analítica de datos.

La cámara de bienes raíces del condado King County, en Washington, tiene las operaciones de venta de los años 2022 y 2023. Se desea hacer un trabajo de analítica para comparar la competencia que tienen tres agentes inmobiliarios: GINA JEANNOT, FRANK PAINTER – NEOHOMES, y SKYLINE PROPERTIES.

Dispones del siguiente esquema de datos, que deberás integrar:

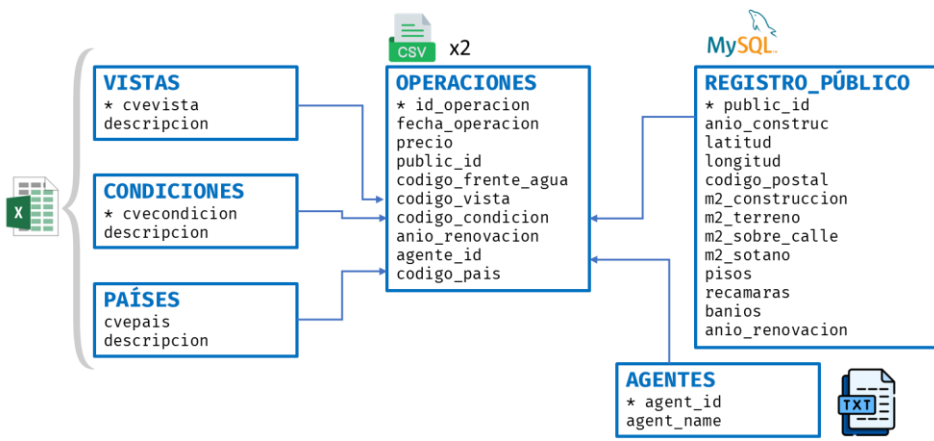


FIGURA 11.01: *Integración de master a partir de múltiples orígenes.*

Las tareas por hace son las siguientes:

1. Documentar la lista de fuentes.
2. Integrar un solo conjunto de datos a partir de dos.
3. Generar categóricos descriptivos para el año, año-mes, y mes de operación.
4. Recuperar los catálogos de diferentes fuentes.

5. Homologación de campos, tipos y nombres.
6. Integrar el master, asociando los catálogos y registro público, con operaciones.
7. Guardar el master en un archivo CSV.

Documentar la lista de fuentes

Las fuentes por utilizar en la integración son las siguientes:

Nombre corto:	**operaciones_2022**
Tipo de fuente:	Archivo CSV
Nombre de archivo:	**operaciones_2022.csv**
Permisos requeridos:	No aplica
Campos por recuperar:	**id_operacion (int)** **fecha_operacion (datetime)** **precio (float)** **public_id (int)** **codigo_frente_agua (int)** **codigo_vista (int)** **codigo_condicion (int)** **agente_id (int)** **codigo_pais (int)**
Filtros aplicables:	Solo incluir las operaciones de los agentes requeridos (**agente_id** = 2, 7, 9).
Generalidades:	Se recupera con encabezados.

Nombre corto:	**operaciones_2023**
Tipo de fuente:	Archivo CSV
Nombre de archivo:	**operaciones_2023.csv**
Permisos requeridos:	No aplica
Campos por recuperar:	**id_operacion (int)** **fecha_operacion (datetime)** **precio (float)** **public_id (int)** **codigo_frente_agua (int)** **codigo_vista (int)** **codigo_condicion (int)** **agente_id (int)** **codigo_pais (int)**
Filtros aplicables:	Solo incluir las operaciones de los agentes requeridos (**agente_id** = 2, 7, 9).
Generalidades:	Se recupera sin encabezados.

Nombre corto:	**vistas**
Tipo de fuente:	Archivo XLSX (Libro Excel)
Nombre de archivo:	**catalogos.xlsx**
Hoja electrónica:	**Categóricos**
Elemento:	**VISTAS** (Rango de celda)
Permisos requeridos:	No aplica
Campos por recuperar:	**cvevista** **descripcion**
Filtros aplicables:	Ninguno
Generalidades:	Ninguno

Nombre corto:	**condiciones**
Tipo de fuente:	Archivo XLSX (Libro Excel)
Nombre de archivo:	**catalogos.xlsx**
Hoja electrónica:	**Categóricos**
Elemento:	**CONDICIONES** (Rango de celda)
Permisos requeridos:	No aplica
Campos por recuperar:	**cvecondicion** **descripcion**
Filtros aplicables:	Ninguno
Generalidades:	Ninguno

Nombre corto:	**países**
Tipo de fuente:	Archivo XLSX (Libro Excel)
Nombre de archivo:	**catalogos.xlsx**
Hoja electrónica:	**Categóricos**
Elemento:	**A18:B28** (Rango de celda)
Permisos requeridos:	No aplica
Campos por recuperar:	**cvepais** **descripcion**
Filtros aplicables:	Ninguno
Generalidades:	Ninguno

Nombre corto:	**agentes**
Tipo de fuente:	Archivo TXT
Nombre de archivo:	**agentes_inmobiliarios.txt**
Permisos requeridos:	No aplica
Campos por recuperar:	**agent_id** **agent_name**
Filtros aplicables:	Ninguno
Generalidades:	El **agent_id** se encuentra de la columna 1 a la 10. El **agent_name** se encuentra de la columna 11 a la 40.

Nombre corto:	**registro_público**
Tipo de fuente:	Base de datos MySQL
Servidor:	**<nombre de servidor>**
Base de datos:	
Usuario:	
Password:	
Permisos requeridos:	**Basados en usuario**
Consulta:	**registro_publico**
Permisos requeridos:	No aplica
Campos por recuperar:	**public_id** **anio_construc** **latitud** **longitud** **codigo_postal** **m2_construccion** **m2_terreno** **m2_sobre_calle** **m2_sotano** **pisos** **recamaras** **banios** **anio_renovacion**
Filtros aplicables:	Ninguno
Generalidades:	Ninguno

Integrar un solo conjunto de datos a partir de dos

En algunas ocasiones se tienen que integrar datos que tienen la misma estructura, pero tienen datos diferentes por diferencia en la temporalidad, o algún filtro. En nuestro caso, tenemos dos archivos que contienen la misma estructura, pero un tiene datos de 2022, y otro de 2023, en unos DataFrame llamados **op_2022** y **op_2023**, respectivamente.

La secuencia de trabajo es la siguiente:

1. Se importan las librerías que han de ocuparse para el procesamiento de datos (**pandas**) y el trabajo con fechas (**numpy** y **datetime**).
2. Se almacena en variables la liga de acceso a datos RAW en GitHub.
3. Como no se requieren todas las columnas contenidas en los archivos, se enumeran las columnas de interés en una lista.

4. Como queremos importar los datos especificando el tipo de dato que nos interesa que tengan en nuestro DataFrame, se especifican los tipos de dato deseados, en un diccionario.
5. Se cargan los datos usando el método **`read_csv()`** de pandas.
6. Se muestra la lista de campos correspondiente a los datos recuperados.
7. Como solo se requieren datos de ciertos agentes inmobiliarios, se realiza un filtro de filas, dejado solo las operaciones de Gina Jeannot, Frank Painter, y Skyline.
8. Se normalizan los tipos de datos de los datos recuperados, con el fin de que solo se tengan como datos de valor aquellos que ameriten cálculos.
9. Se concatenan los DataFrames, para tener un solo conjunto de datos.

```
# Se importan las librerías requeridas para la
# integración.
from numpy import datetime64
import pandas as pd
import datetime as dt

# Se declaran variables con las ligas de acceso a
# los datos en GitHub.

# Operaciones 2022
url_2022='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
operaciones_2022.csv'

# Operaciones 2023
url_2023='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
operaciones_2023.csv'

# Se define una lista que enumera las columnas
# de interés.
columnas_requeridas=['id_operacion', 'fecha_operacion',
    'precio', 'public_id', 'codigo_frente_agua',
    'codigo_vista', 'codigo_condicion',
    'agente_id', 'codigo_pais'
]
```

```
# Se define un diccionario que enumera los tipos
# de datos esperados para las columnas.
tipos_requeridos={
    'id_operacion':int,
    'fecha_operacion':str,
    'precio':float,
    'public_id':int,
    'codigo_frente_agua':int,
    'codigo_vista':int,
    'codigo_condicion':int,
    'agente_id':int,
    'codigo_pais':int
}

# Se lee el CSV con los datos de 2022, recuperando
# las columnas de interés, con el tipo de dato
# deseado.
op_2022=pd.read_csv(url_2022,
                    usecols=columnas_requeridas,
                    dtype=tipos_requeridos
                    )

# Se muestran los datos y la volumetría de filas.
op_2022.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id_operacion        21613 non-null  int64
 1   fecha_operacion     21613 non-null  object
 2   precio              21613 non-null  float64
 3   public_id           21613 non-null  int64
 4   codigo_frente_agua  21613 non-null  int64
 5   codigo_vista        21613 non-null  int64
 6   codigo_condicion    21613 non-null  int64
 7   agente_id           21613 non-null  int64
 8   codigo_pais         21613 non-null  int64
dtypes: float64(1), int64(7), object(1)
memory usage: 1.5+ MB
```

```
# Se filtran los datos, para que solamente queden
# las filas de los agentes sujetos a análisis.
op_2022=op_2022[op_2022['agente_id'].isin([2,7,9])]
```

```
# Se muestran los datos y la volumetría de filas.
op_2022.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5226 entries, 3 to 21612
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id_operacion        5226 non-null   int64
 1   fecha_operacion     5226 non-null   object
 2   precio              5226 non-null   float64
 3   public_id           5226 non-null   int64
 4   codigo_frente_agua  5226 non-null   int64
 5   codigo_vista        5226 non-null   int64
 6   codigo_condicion    5226 non-null   int64
 7   agente_id           5226 non-null   int64
 8   codigo_pais         5226 non-null   int64
dtypes: float64(1), int64(7), object(1)
memory usage: 408.3+ KB
```

```
# Se modifican los tipos de datos de las columnas
# para mejorar su tratamiento.

# id_operacion se convierte a cadena (str / object)
op_2022['id_operacion']=op_2022['id_operacion'].astype(str)

# fecha_opercion se convierte a fecha, con formato DD/MM/AAAA.
op_2022['fecha_operacion']=pd.to_datetime(
    op_2022['fecha_operacion'], format='%d/%m/%Y')

# public_id se convierte a cadena (str / object)
op_2022['public_id']=op_2022['public_id'].astype(str)

# codigo_frente_agua se convierte a cadena (str / object)
op_2022['codigo_frente_agua']=op_2022['codigo_frente_agua'].
      astype(str)

# codigo_vista se convierte a cadena (str / object)
op_2022['codigo_vista']=op_2022['codigo_vista'].astype(str)

# codigo_condicion se convierte a cadena (str / object)
op_2022['codigo_condicion']=op_2022['codigo_condicion'].
      astype(str)

# agente_id  se convierte a cadena (str / object)
op_2022['agente_id']=op_2022['agente_id'].astype(str)

# codigo_pais se convierte a cadena (str / object)
op_2022['codigo_pais']=op_2022['codigo_pais'].astype(str)
```

```
# Se muestran los datos y la volumetría de filas.
op_2022.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5226 entries, 3 to 21612
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id_operacion        5226 non-null   object
 1   fecha_operacion     5226 non-null   datetime64[ns]
 2   precio              5226 non-null   float64
 3   public_id           5226 non-null   object
 4   codigo_frente_agua  5226 non-null   object
 5   codigo_vista        5226 non-null   object
 6   codigo_condicion    5226 non-null   object
 7   agente_id           5226 non-null   object
 8   codigo_pais         5226 non-null   object
dtypes: datetime64[ns](1), float64(1), object(7)
memory usage: 408.3+ KB
```

```
# Se repite el proceso para los datos de 2023.
op_2023=pd.read_csv(url_2023,
                    usecols=columnas_requeridas,
                    dtype=tipos_requeridos
                    )

op_2023=op_2023[op_2023['agente_id'].isin([2,7,9])]
op_2023['id_operacion']=op_2023['id_operacion'].astype(str)
op_2023['fecha_operacion']=pd.to_datetime(
    op_2023['fecha_operacion'], format='%d/%m/%Y')
op_2023['public_id']=op_2023['public_id'].astype(str)
op_2023['codigo_frente_agua']=op_2023['codigo_frente_agua'].
      astype(str)
op_2023['codigo_vista']=op_2023['codigo_vista'].astype(str)
op_2023['codigo_condicion']=op_2023['codigo_condicion'].
      astype(str)
op_2023['agente_id']=op_2023['agente_id'].astype(str)
op_2023['codigo_pais']=op_2023['codigo_pais'].astype(str)

# Se muestran los datos y la volumetría de filas.
op_2023.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1665 entries, 2 to 6934
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id_operacion        1665 non-null   object
 1   fecha_operacion     1665 non-null   datetime64[ns]
 2   precio              1665 non-null   float64
```

```
 3   public_id           1665 non-null   object
 4   codigo_frente_agua  1665 non-null   object
 5   codigo_vista        1665 non-null   object
 6   codigo_condicion    1665 non-null   object
 7   agente_id           1665 non-null   object
 8   codigo_pais         1665 non-null   object
dtypes: datetime64[ns](1), float64(1), object(7)
memory usage: 130.1+ KB
```

```
# Se muestra el número de filas de cada uno de
# los DataFrames creados.
print(len(op_2022))
print(len(op_2023))

# Se concatenan (unen) las filas de los dos
# DataFrames, generando un nuevo índice,
# omitiendo a los índices originales.
operaciones=pd.concat([op_2022, op_2023],
                      ignore_index=True)

# Se imprime el número de filas de las operaciones
# integradas.
print(len(operaciones))
```

```
5226
1665
6891
```

Generar categóricos descriptivos para el año, año-mes, y mes de operación

A partir de un dato fecha/hora, se generan descriptivos categóricos que serán útiles al momento de hacer analítica, con etiquetas útiles para series de tiempo. Se requiere una llamada **año**, que contendrá el año en formato cadena, **año_mes**, que muestre el año y el número de mes, en formato cadena, y **mes**, que muestre el número de mes, más el nombre completo del mes, en mayúsculas, en formato cadena.

```
# Se generan categóricos descriptivos para el
# año (AAAA), el año con el mes (AAAA-MM), y el
# mes con el nombre, en mayúsculas.
operaciones['año'] = operaciones[
    'fecha_operacion'].dt.strftime('%Y')
```

```
operaciones['año_mes'] = operaciones[
    'fecha_operacion'].dt.strftime('%Y-%m')
operaciones['mes'] = operaciones[
    'fecha_operacion'].dt.strftime('%m-%B').str.upper()

# Se imprime una tabla de frecuencia de cada
# uno de los categóricos generados.
print(operaciones['año'].value_counts(),'\n')
print(operaciones['año_mes'].value_counts(),'\n')
print(operaciones['mes'].value_counts(),'\n')
```

```
2022    3561
2023    3330
Name: año, dtype: int64

2023-04    1040
2023-03     942
2023-02     580
2022-06     554
2022-08     499
2022-07     487
2023-01     464
2022-09     450
2022-10     432
2022-05     414
2022-11     372
2022-12     335
2023-05     304
2022-04      18
Name: año_mes, dtype: int64

04-APRIL        1058
03-MARCH         942
05-MAY           718
02-FEBRUARY      580
06-JUNE          554
08-AUGUST        499
07-JULY          487
01-JANUARY       464
09-SEPTEMBER     450
10-OCTOBER       432
11-NOVEMBER      372
12-DECEMBER      335
Name: mes, dtype: int64
```

Recuperar los catálogos de diferentes fuentes

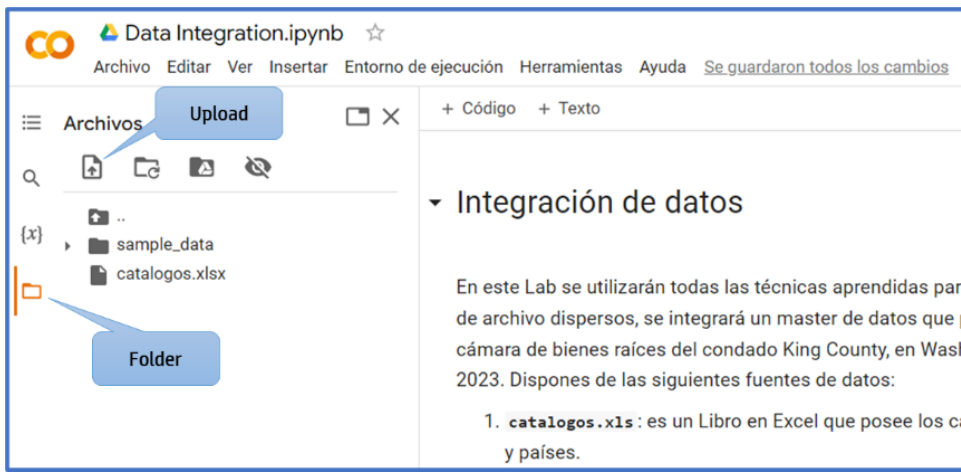
Se recopilan los categóricos descriptivos a partir de diferentes, que será útil al momento de integrar el master.

Para el caso de los catálogos de vistas, condiciones de las viviendas y países, los datos se encuentran en un Libro de Excel llamado **`catalogos.xlsx`**; los catálogos se encuentran en una misma Hoja llamada **`Categóricos`**.

El archivo catalogos.xlsx se debe descargar de la carpeta data del repositorio GitHub que acompaña al libro.

`https://github.com/AprendaPracticando/AnaliticaPythonR1/tree/main/data`

Descarga el archivo del repositorio, y cárgalo en tu ambiente de **Google Colab**. Para hacerlo, debes hacer clic en el ícono de ***Folder*** que aparece en la extrema izquierda de **Google Colab**, y luego, hacer clic en el ícono ***Upload*** (subir archivo) de **Google Colab**.



Las vistas están en el Rango de celdas **`A2:B7`**, las condiciones están en el Rango de celdas **`A10:B15`**, y los países se encuentran en el Rango de celdas **`A18:B28`**. Se leerán usando el método de pandas **`read_excel()`**, y se guarda el resultado en los DataFrame **`vistas`**, **`condiciones`** y **`países`**.

```
# Se recuperan los catálogos desde Excel

# Recuperación desde el Rango de celdas
# A2:B7, en la Hoja Catálogos, para extraer
# las vistas
vistas=pd.read_excel('catalogos.xlsx',
                     sheet_name='Categóricos',
                     usecols='A:B', skiprows=1,
                     nrows=5)

# Muestra lo recuperado.
print(vistas,'\n')

# Recuperación desde el Rango de celdas
# A10:B15, en la Hoja Catálogos, para extraer
# las condiciones.
condiciones=pd.read_excel('catalogos.xlsx',
                     sheet_name='Categóricos',
                     usecols='A:B', skiprows=9,
                     nrows=5)

# Muestra lo recuperado.
print(condiciones,'\n')

# Recuperación desde el Rango de celdas
# A18:B28, en la Hoja Catálogos, para extraer
# los países.
países=pd.read_excel('catalogos.xlsx',
                     sheet_name='Categóricos',
                     usecols='A:B', skiprows=17,
                     nrows=10)

# Muestra lo recuperado.
print(países,'\n')
```

```
   cvevista          descripcion
0         0          E-SIN VISTA
1         1     D-VISTA ESTÁNDAR
2         2      C-VISTA ABIERTA
3         3   B-VISTA PANORÁMICA
4         4  A-VISTA EXCEPCIONAL

   cvecondicion              descripcion
0             1       E-MALAS CONDICIONES
1             2   D-CONDICIONES REGULARES
2             3      C-BUENAS CONDICIONES
3             4      B-BUENAS CONDICIONES
4             5  A-EXCELENTES CONDICIONES

   cvepais     descripcion
```

```
0        1  ESTADOS UNIDOS
1        2          MEXICO
2        3     PUERTO RICO
3        4          CANADA
4        5         ALEMANIA
5        6       INGLATERRA
6        7         COLOMBIA
7        8           PANAMA
8        9            CHINA
9       10            COREA
```

El caso del catálogo de agentes es distinto. Los datos se encuentran en un archivo textual, que en la primera fila contiene el nombre de las columnas, y a partir de la segunda línea son datos.

La primera columna corresponde al id del agente, y la segunda columna es el nombre del agente. El id del agente está desde la posición 1 a la 10, mientras que el nombre del agente está de la posición 11 a la 40.

Para recuperar el contenido se utiliza el método **`read_fwf()`** de pandas, a partir de un archivo llamado agentes_inmobiliarios.txt, que se encuentra en **GitHub** y que puede ser recuperado en formato RAW, a partir de una liga de internet (*URL*).

```
# Liga para recuperar el archivo TXT en modo RAW, desde GitHub
url_txt='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
agentes_inmobiliarios.txt'

# Se recupera la información del archivo, dividiendo los campos
# terminando el primero en la columna 10, y el segundo en la
# columna 40, infiriendo si existe o no encabezados.
# Encontrará encabezados porque la primera fila contiene sólo
# etiquetas, pero de la fila 2 en adelante, se encuentran
# números en la primera columna.
# Se usa read_fwf() (fixed width formatted)
agentes=pd.read_fwf(url_txt,
                    widths=[10,40],
                    header='infer')

# Se muestra lo recuperado
print(agentes)
```

```
    agent_id               agent_name
0          1               WINDERMERE
1          2             GINA JEANNOT
2          3        MELANIE ANTONUCCI
3          4    CARRIE FOLEY- COMPASS
4          5             RACHEL ADLER
5          6       TEAMUP AT COLDWELL
6          7       SKYLINE PROPERTIES
7          8                 FLYHOMES
8          9  FRANK PAINTER - NEOHOMES
9         10           CHAMPMAN HOMES
10        11                    OWNER
```

Para recuperar los datos del registro público, se debe acceder a una base de datos MySQL. Para poder acceder a los datos, es necesario que instale el conector a base de datos MySQL de Python, ejecutando la siguiente línea.

```
!pip install mysql-connector-python
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/
colab-wheels/public/simple/
Collecting mysql-connector-python
  Downloading mysql_connector_python-8.0.32-cp39-cp39-manylinux1_x86_64.
whl  (23.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23.5/23.5 MB 53.3 MB/s eta 0:00:00
Requirement already satisfied: protobuf<=3.20.3,>=3.11.0 in
/usr/local/lib/python3.9/dist-packages (from mysql-connector-python) (3.20.3)
Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-8.0.32
```

Instalado el paquete, debes asegurarte de que dispones de una base de datos MySQL que contenga los datos. Escapa al alcance del libro decirte lo que tienes que hacer para que los datos estén disponibles. Puedes hacer lo siguiente, para disponer de la base de datos requerida:

1. Instalar tu instancia de MySQL.
2. Crear una base de datos vacía.
3. Ejecutar el siguiente estatuto SQL, para crear la tabla **`registro_publico`**:

```
CREATE TABLE registro_publico(
    public_id CHAR(8),
    anio_construc CHAR(4),
    latitud FLOAT,
    longitud FLOAT,
    codigo_postal CHAR(5),
    m2_construccion FLOAT,
    m2_terreno FLOAT,
    m2_sobre_calle FLOAT,
    m2_sotano FLOAT,
    pisos INT,
    recamaras INT,
    banios FLOAT,
    anio_renovacion CHAR(4),
    PRIMARY KEY (public_id)
)
```

4. Descargar el archivo **`registro_publico.csv`**, que está en el folder data del repositorio de datos en GitHub que acompaña al libro.
5. Importar los datos del CSV, para que se carguen en la tabla que has creado en MySQL.
6. Obtener la información de servidor, usuario y password que te permitan acceder a los datos desde tu programa.

Se debe importar el conector de datos a MySQL, establecer conexión con la base de datos proporcionando las credenciales de acceso que apliquen, y crear el DataFrame llamado registro_publico, donde se carguen los datos que se encuentren en la tabla registro_publico que se encuentra en la base de datos. Es importante aclarar que el DataFrame y la tabla no tienen por qué llamarse igual, aunque en este caso sí lo hacen.

```
import mysql.connector

# Establecer la conexión con la base de datos
mydb = mysql.connector.connect(
  host="MYSQL5043.site4now.net",
  user="9c7dd4_king",
  password="P@ssw0rd",
  database="db_9c7dd4_king"
)
```

```
# Definir la consulta SQL
query = "SELECT * FROM registro_publico;"

# Leer los datos desde MySQL y cargarlos en un DataFrame
registro_publico = pd.read_sql(query, con=mydb, index_col=None)

mydb.close()

registro_publico.drop(index=0)
```

Homologación de campos, tipos y nombres

Se homologan los campos, tipos y nombres, para poder establecer las relaciones entre datos. Este trabajo es laborioso, pero esencial para que todo funcione de maravilla.

```
# El campo de coincidencia en operaciones se llama
# codigo_vistas, mientras que en vista se llama cvevista.
# El campo de coincidencia en operaciones es de
# tipo object, y en vistas es int64.
# El campo descriptivo en vista se llama descripción
# al igual que en otros catálogos.

# Se cambian los nombres de columna de vista, a) Para
# homologación con operaciones, y b) Difereciarlo de otras
# descripciones en otros catálogos.
vistas.rename(columns={'cvevista':'codigo_vista',
                       'descripcion':'vista'}, inplace=True)

# Se cambia el tipo de dato en vista, para homologarlo en
# operaciones.
vistas['codigo_vista']=vistas['codigo_vista'].astype(str)

# Se ve el resultado
vistas.dtypes
```

```
código_vista    object
vista           object
dtype: object
```

```
# El campo de coincidencia en operaciones se llama
# codigo_condicion, mientras que en condiciones se llama
# cvecondicion.
# El campo de coincidencia en operaciones es de
# tipo object, y en condiciones es int64.
# El campo descriptivo en condiciones se llama descripción
# al igual que en otros catálogos.

# Se cambian los nombres de columna de condiciones, a) Para
# homologación con operaciones, y b) Diferenciarlo de otras
# descripciones en otros catálogos.
condiciones.rename(columns={'cvecondicion':'codigo_condicion',
            'descripcion':'condición'}, inplace=True)

# Se cambia el tipo de dato en vista, para homologarlo en
# operaciones.
condiciones['codigo_condicion']=
      condiciones['codigo_condicion'].astype(str)

# Se ve el resultado
condiciones.dtypes
```

```
código_condición    object
condición           object
dtype: object
```

```
# El campo de coincidencia en operaciones se llama
# codigo_pais, mientras que en países se llama
# cvepais.
# El campo de coincidencia en operaciones es de
# tipo object, y en países es int64.
# El campo descriptivo en países se llama descripción
# al igual que en otros catálogos.

# Se cambian los nombres de columna de países, a) Para
# homologación con operaciones, y b) Diferenciarlo de otras
# descripciones en otros catálogos.
países.rename(columns={'cvepais':'codigo_pais',
                       'descripcion':'país'}, inplace=True)

# Se cambia el tipo de dato en vista, para homologarlo en
# operaciones.
países['codigo_pais']=países['codigo_pais'].astype(str)

# Se ve el resultado
países.dtypes
```

```
código_país    object
país           object
dtype: object
```

```
# El campo de coincidencia en operaciones se llama
# agente_id, mientras que en agentes se llama
# agent_id.
# El campo de coincidencia en operaciones es de
# tipo object, y en agentes es int64.

# Se cambian los nombres de columna de países, a) Para
# homologación con operaciones.
agentes.rename(columns={'agent_id':'agente_id',
                        'agent_name':'agente'}, inplace=True)

# Se cambia el tipo de dato en vista, para homologarlo en
# operaciones.
agentes['agente_id']=agentes['agente_id'].astype(str)

# Se ve el resultado
agentes.dtypes
```

```
agente_id    object
agente       object
dtype: object
```

Integrar el master, asociando los catálogos y registro público, con operaciones

Habiendo homologado los nombres de columna y sus tipos, podemos integrar el master sin problemas. Escogemos la tabla que constituye el centro de los datos. En este caso, se trata de operaciones, porque es tabla débil en todos los casos, y todas las tablas del modelo fluyen hacia dicha tabla.

```
# Asociando Vistas
operaciones=operaciones.merge(vistas,
                  on='codigo_vista',
                  how='inner')
```

```
# Asociando Condiciones
operaciones=operaciones.merge(condiciones,
                  on='codigo_condicion',
                  how='inner')

# Asociando País
operaciones=operaciones.merge(países,
                  on='codigo_pais',
                  how='inner')

# Asociando Agentes
operaciones=operaciones.merge(agentes,
                  on='agente_id',
                  how='inner')

# Asociando Registro público
operaciones=operaciones.merge(registro_publico,
                  on='public_id',
                  how='inner')

# Ver el resultado
operaciones.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6891 entries, 0 to 6890
Data columns (total 28 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id_operación        6891 non-null   object
 1   fecha_operación     6891 non-null   datetime64[ns]
 2   precio              6891 non-null   float64
 3   public_id           6891 non-null   object
 4   código_frente_agua  6891 non-null   object
 5   código_vista        6891 non-null   object
 6   código_condición    6891 non-null   object
 7   agente_id           6891 non-null   object
 8   código_país         6891 non-null   object
 9   año                 6891 non-null   object
 10  año_mes             6891 non-null   object
 11  mes                 6891 non-null   object
 12  vista               6891 non-null   object
 13  condición           6891 non-null   object
 14  país                6891 non-null   object
 15  agente              6891 non-null   object
 16  anio_construc       6891 non-null   object
 17  latitud             6891 non-null   float64
 18  longitud            6891 non-null   float64
 19  codigo_postal       6891 non-null   object
 20  m2_construccion     6891 non-null   float64
 21  m2_terreno          6891 non-null   float64
 22  m2_sobre_calle      6891 non-null   float64
 23  m2_sotano           6891 non-null   float64
 24  pisos               6891 non-null   int64
```

```
 25  recamaras           6891 non-null   int64
 26  banios              6891 non-null   float64
 27  anio_renovacion     6891 non-null   object
dtypes: datetime64[ns](1), float64(8), int64(2), object(17)
memory usage: 1.5+ MB
```

Guardar el master en un archivo CSV

Ya que se tiene todo el master integrado, es buena idea generar un archivo CSV a partir del mismo, con la finalidad de disponer de un único origen de datos sobre el cual trabajar, sin depender de la conectividad con las múltiples fuentes. Si no nos podemos conectar con la base de datos, o si no podemos enlazarnos a **Github**, no hay problema, porque al tener el master podemos hacer trabajos de analítica a partir del archivo generado.

```
# Se guarda el DataFrame integrado, y se guarda en un
# archivo CSV, con encabezados, sin incluir el índice
# como columna.
operaciones.to_csv('operaciones_master.csv',
                   header=True, index=False)

# Revisar en Google Colab la integración del archivo.
```

Las variantes de una integración pueden ser muchas. Algunas requieren más trabajo de homologación y limpieza, pero a grandes rasgos es lo mismo.

FIN DEL LAB

CAPÍTULO 12:

Tratamiento de datos ausentes (missing data)

LAB 12.01: Tratamiento de datos ausentes

En este Lab se desea generar, paso a paso, un categórico de intervalo.

Las tareas por realizar son:

1. Definir la estrategia de manejo de atípicos del caso.
2. Revisión de existencia de datos ausentes.
3. Estrategia de eliminación de filas con ausentes. **`value_counts()`**.
4. Establecer manualmente los límites de intervalo (**`bins`**).
5. Cambiar intervalos a semi-cerrado a la derecha **`[x,y)`** (**`right`**).
6. Definir las etiquetas de intervalo (**`labels`**).
7. Generación de categórico de intervalo usando UDF's.

```
# Datos base
import pandas as pd
tipos_correctos={
    'id_persona':object,
    'clave_sobrevivencia':object,
    'clase_viaje':object
}
personas_titanic_csv='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
personas_titanic_v5.csv'
titanic = pd.read_csv(personas_titanic_csv,
      dtype=tipos_correctos)

titanic['id_persona']=titanic['id_persona'].str.split('.').
      str.get(0)
titanic['clave_sobrevivencia']=titanic['clave_sobrevivencia'].
      str.split('.').str.get(0)
titanic['clase_viaje']=titanic['clase_viaje'].str.split('.').
      str.get(0)
```

Revisar la existencia de ausentes y la suficiencia de datos

Recordemos el objetivo de análisis:

OBJETIVO DE ANÁLISIS: Considerando la información histórica contenida en la enciclopedia británica, relativa a personas que iban en el TITANIC y la suerte que tuvieron, queremos hacer un análisis exploratorio histórico que nos permita saber cómo afectaron a la sobrevivencia de las personas, aspectos como la clase de pasajero, sexo, rango de edad, y si la persona sola o acompañada.

Para efecto de cubrir el objetivo de análisis y nuestras hipótesis, las columnas que permanecen como requeridas son las siguientes:

- **sobrevivencia**
- **clase**
- **sexo**
- **rango_edad**
- **acompañada**

```
# Revisamos la disponibilidad de los datos, usando info()
# Para comprobar si los campos requeridos cumplen con el mínimo
# requerido por la muestra estadística
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1310 entries, 0 to 1309
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   id_persona           1310 non-null   object
 1   clave_sobrevivencia  1307 non-null   object
 2   nombre               1310 non-null   object
 3   sexo                 1307 non-null   object
 4   clase_viaje          1307 non-null   object
 5   id_bote              486 non-null    object
 6   id_cuerpo            121 non-null    float64
 7   edad                 1046 non-null   float64
 8   cantidad_parientes   1310 non-null   int64
 9   cantidad_familiares  1310 non-null   int64
 10  acompañantes         1310 non-null   int64
 11  rango_edad           1046 non-null   object
 12  sobrevivencia        1307 non-null   object
 13  clase                1307 non-null   object
 14  acompañada           1310 non-null   object
dtypes: float64(2), int64(3), object(10)
memory usage: 153.6+ KB
```

Podemos concluir que todos los campos requeridos tienen más observaciones que las indicadas en el tamaño de la muestra (511), por lo cual, las operaciones realizadas sobre estos datos pueden tener validez estadística.

Campo	Requeridos	Disponibles	¿Son suficientes?
`sobrevivencia`	511	1307	✓
`clase_viaje`	511	1307	✓
`sexo`	511	1307	✓
`rango_edad`	511	1046	✓
`acompañada`	511	1310	✓

Eliminación de filas con datos ausentes

Podemos revisar si eliminar los registros que presenten datos ausentes nos puede servir.

```
# Por comodidad, generamos una lista con los nombres de los
# campos requeridos por el objetivo de análisis.
requeridos=['sobrevivencia','clase_viaje','sexo',
      'rango_edad','acompañada']

# Verificamos cuántas filas tenemos, sin faltantes en
# ningún campo requerido.
sin_ausentes=titanic.dropna(subset=requeridos)

print(len(sin_ausentes))
```

```
1040
```

Antes de eliminar las filas que tengan vacíos o nulos en los campos requeridos, tenemos 1,040 filas, que es muy por encima de la muestra estadística, que es 511.

Podemos afirmar que, en este caso, podemos darnos el lujo de eliminar cualquier fila que tenga un ausente, y no comprometer a la muestra estadística.

Corrección de MAR usando requeridos indirectos y UDF

Si los datos fueran escasos y cada fila fuera importante, podríamos intentar inferir los datos ausentes, a partir de los datos requeridos indirectos.

Pongamos el ejemplo de sobrevivencia: identificamos todos los datos ausentes en **`sobrevivencia`**, que como sabemos, es primordial por tratarse de la variable dependiente.

Primero, revisamos en qué filas tenemos ausentes en **`sobrevivencia`**:

```
# Identifica los ausentes del campo sobrevivencia.
titanic[['nombre','clave_sobrevivencia',
         'sobrevivencia','id_bote',
         'id_cuerpo']][(titanic['sobrevivencia'].isnull())]
```

```
                                nombre clave_sobrevivencia sobrevivencia  \
108               Artagaveytia, Mr. Ramon                NaN           NaN
272   Allison, Mr. Hudson Joshua Creighton                NaN           NaN
1308          Blackwell, Mr. Stephen Weart                NaN           NaN

     id_bote  id_cuerpo
108      NaN       22.0
272      NaN      135.0
1308     NaN        NaN
```

Como podemos ver, hay tres filas con datos ausentes, pero sabemos también que a partir de **`id_bote`** y **`id_cuerpo`** podemos inferir **`sobrevivencia`**:

1. Si **`id_bote`** no es nulo, quiere decir que la persona se subió a un bote salvavidas, y vivió.
2. Si **`id_cuerpo`** no es nulo, quiere decir que le asignaron un número de cuerpo, porque murió.

Elaboramos una función definida por el usuario para hacer el cálculo, y actualizamos **`sobrevivencia`** y **`clave_sobrevivencia`**.

```
# Función que infiere el valor de sobreviencia a partir de
# id_bote o id_cuerpo.
def corrige_sobrevivencia(fila):
  # El valor actual de sobrevivencia se asigna a una variable.
  etiqueta=fila['sobrevivencia']
  # Si id_cuerpo no es nulo, la etiqueta pasa a MURIÓ
  if (not pd.isnull(fila['id_cuerpo'])):
    etiqueta='MURIÓ'
  # Si id_bote no es nulo, la etiqueta pasa a VIVIÓ
  if (not pd.isnull(fila['id_bote'])):
    etiqueta='VIVIÓ'
  # Se retorna la etiqueta.
  return etiqueta

def corrige_clave_sobrevivencia(fila):
  # El valor actual de sobrevivencia se asigna a una variable.
  etiqueta=fila['clave_sobrevivencia']
  # Validamos que la etiqueta no esté vacía o sea nula
  if (not pd.isnull(fila['clave_sobrevivencia'])):
    if (fila['sobrevivencia']=='MURIÓ'):
      etiqueta='0'
    else:
      etiqueta='1'
  # Se retorna la etiqueta.
  return etiqueta

# Se aplica la función definida por el usuario, a cada
# fila del DataSet, actualizando sobrevivencia con el
# valor retornado.
titanic['sobrevivencia']=titanic.
      apply(corrige_sobrevivencia,axis=1)
titanic['clave_sobrevivencia']=titanic.
      apply(corrige_clave_sobrevivencia,axis=1)

# Identifica los ausentes del campo sobrevivencia.
# Sigue habiendo uno, porque no hay información ni en
# cuerpo ni en bote, y no hay otra manera de saber si
# vivió o murió.
# Muestra las columnas nombre, clave_sobrevivencia,
# sobrevivencia, id_bote y id_cuerpo, cuando sobrevivencia
# sea nulo.
titanic[['nombre','clave_sobrevivencia',
         'sobrevivencia','id_bote',
         'id_cuerpo']][(titanic['sobrevivencia'].isnull())]
```

```
# Este dato es MNAR (missing not at random),
# pues sistemáticamente, en todos los casos, si tenemos
# bote o cuerpo, podemos inferir sobrevivencia.
```

```
                          nombre clave_sobrevivencia sobrevivencia id_bote  \
1308  Blackwell, Mr. Stephen Weart                NaN           NaN     NaN

      id_cuerpo
1308        NaN
```

Corrección de MAR por asignación directa

El campo **sexo** es ***MAR*** (***missing at random***), porque podemos inferir su valor, pero no de manera sistemática en todos los casos.

Si analizamos el valor de **nombre**, tenemos amplias posibilidades de poder inferir **sexo**, pero no de manera indubitable ni en todos los casos.

Identifiquemos los ausentes en **sexo**:

```
# Identifica los ausentes del campo sexo.
titanic[['nombre','sexo']][(titanic['sexo'].isnull())]
```

```
                        nombre sexo
148     Dick, Mr. Albert Adrian  NaN
237  Douglas, Mr. Walter Donald  NaN
299                Alexis Evans  NaN
```

Las filas con el índice 148 y 237 tienen información suficiente para concluir que los pasajeros eran hombres.

La fila con el índice 299 no tiene forma de inferir el sexo de la persona, porque no incluye título, y Alexis es un nombre que aplica tanto para hombres como para mujeres.

Se actualizan los registros que se pueden corregir, de manera específica, usando **at[]**.

```
# Se le asigna el valor 'HOMBRE' a la columna
# sexo, a las filas con índice 148 y 237, usando
# at[]
titanic.at[148,'sexo']="HOMBRE"
titanic.at[237,'sexo']="HOMBRE"

# Identifica los ausentes del campo sexo.
titanic[['nombre','sexo']][(titanic['sexo'].isnull())]
```

```
           nombre sexo
299  Alexis Evans  NaN
```

FIN DEL LAB

CAPÍTULO 13:

Tratamiento de datos atípicos (Outliers)

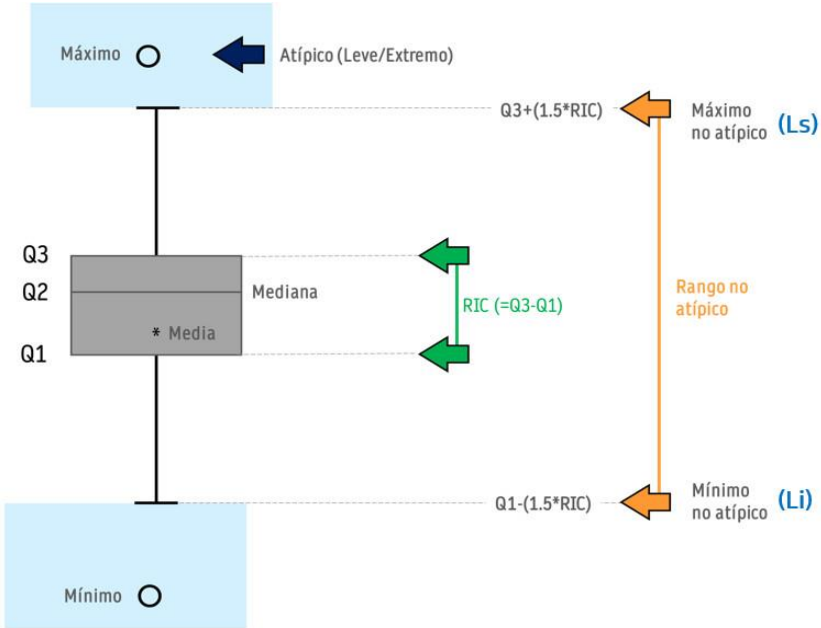


FIGURA 13.01: *Box-Plot.*

LAB 13.01: Tratamiento de datos atípicos

En este Lab se desea desarrollar la estrategia para el tratamiento de datos atípicos.

Imagina que tenemos un DataFrame con las estaturas y los pesos de un conjunto de personas, y que deseas calcular el mínimo no atípico, el máximo no atípico, ver las observaciones con atípicos y con atípicos extremos. Se tienen 4 filas con valores atípicos: 2 son atípicos, y 2 atípicos extremos.

En este Lab se demostrará el uso de diversas técnicas para el manejo de datos atípicos.

Las tareas por realizar son las siguientes:

1. Crear un DataFrame con atípicos y atípicos extremos.
2. Calcular el mínimo no atípico, máximo no atípico, y RIC.
3. Eliminar datos atípicos.
4. Truncar datos atípicos.
5. Suavizar atípicos usando transformación logarítmica.
6. Verificar normalidad usando la prueba Shapiro-Wilk.
7. Verificar normalidad usando histogramas.
8. Manejar atípicos usando la técnica de binning.
9. Manejar atípicos usando técnica de imputación.

Crear un DataFrame con atípicos y atípicos extremos

Genera un DataFrame que deliberadamente tenga datos atípicos.

Se trabajará específicamente con la columna **peso_kg**.

```
import pandas as pd

# Se genera un DataFrame con datos atípicos y atípicos
# extremos.
datos={
    'estatura_cm':[160,172,180,170,160,169,164,178,174,173,
         166,163,171,180,174,169,160,166,177,175,166,165,179,
         162,173,171,162,161,171,180],
    'peso_kg':[67.80,66.24,88.00,47.20,69.00,65.55,72.96,
         79.56,84.36,69.35,71.28,71.82,81.65,104.83,68.08,
         73.14,53.40,42.05,70.07,67.50,62.70,63.70,81.37,
         62.00,75.19,145.20,65.72,57.95,133.90,76.00]
}

df=pd.DataFrame(datos)

print(df)
```

```
    estatura_cm  peso_kg
0           160    67.80
1           172    66.24
2           180    88.00
3           170    47.20
4           160    69.00
5           169    65.55
6           164    72.96
7           178    79.56
8           174    84.36
9           173    69.35
10          166    71.28
11          163    71.82
12          171    81.65
13          180   104.83
14          174    68.08
15          169    73.14
16          160    53.40
17          166    42.05
18          177    70.07
19          175    67.50
20          166    62.70
21          165    63.70
22          179    81.37
23          162    62.00
24          173    75.19
25          171   145.20
26          162    65.72
27          161    57.95
28          171   133.90
29          180    76.00
```

Calcular el mínimo no atípico, máximo no atípico, y RIC

```
# Se calculan el primer y tercer cuartil, el rango
# intercuartílico y los máximos y mínimos no atípicos,
# en su versión regular y extrema.
Q1=df['peso_kg'].quantile(0.25)
Q3=df['peso_kg'].quantile(0.75)

RIC=(Q3-Q1)

Li=Q1-(1.5*RIC)
Ls=Q3+(1.5*RIC)
LiE=Q1-(3.0*RIC)
LsE=Q3+(3.0*RIC)
```

Eliminar datos atípicos

Genera un nuevo DataFrame llamado **`sin_atipicos`**, donde apliques la estrategia de eliminación de atípicos.

El DataFrame **`sin_atipicos`** solo contendrá filas con valores típicos.

Solo habrá filas donde **`peso_kg`** sea mayor o igual al mínimo no atípico, y menor o igual al máximo no atípico.

El resultado debe contener 26 filas.

```
# Se genera un DataFrame que contenga solo las filas donde
# peso tenga valores típicos.
sin_atipicos = df[(df['peso_kg']>=Li) & (df['peso_kg']<=Ls)]

print(sin_atipicos.shape,'\n')
print(sin_atipicos)
```

```
(26, 2)

    estatura_cm  peso_kg
0           160    67.80
1           172    66.24
2           180    88.00
3           170    47.20
4           160    69.00
5           169    65.55
6           164    72.96
7           178    79.56
8           174    84.36
9           173    69.35
```

```
10          166    71.28
11          163    71.82
12          171    81.65
14          174    68.08
15          169    73.14
16          160    53.40
18          177    70.07
19          175    67.50
20          166    62.70
21          165    63.70
22          179    81.37
23          162    62.00
24          173    75.19
26          162    65.72
27          161    57.95
29          180    76.00
```

Genera un nuevo DataFrame llamado **`sin_atípicos_extremos`**, donde apliques la estrategia de eliminación de atípicos.

El DataFrame **`sin_atípicos_extremos`** solo contendrá filas con valores típicos y atípicos que no sean extremos.

Solo habrá filas donde **`peso_kg`** sea mayor o igual al mínimo no atípico extremo, y menor o igual al máximo no atípico extremo.

El resultado debe contener 28 filas.

```
# Se genera un DataFrame que contenga solo las filas donde
# peso tenga valores típicos y atípicos no extremos.
sin_atípicos_extremos = df[(df['peso_kg']>=LiE) &
      (df['peso_kg']<=LsE)]

print(sin_atípicos_extremos.shape,'\n')
print(sin_atípicos_extremos)
```

```
(28, 2)

    estatura_cm  peso_kg
0           160    67.80
1           172    66.24
2           180    88.00
3           170    47.20
4           160    69.00
5           169    65.55
6           164    72.96
7           178    79.56
8           174    84.36
9           173    69.35
10          166    71.28
11          163    71.82
```

```
12          171    81.65
13          180   104.83
14          174    68.08
15          169    73.14
16          160    53.40
17          166    42.05
18          177    70.07
19          175    67.50
20          166    62.70
21          165    63.70
22          179    81.37
23          162    62.00
24          173    75.19
26          162    65.72
27          161    57.95
29          180    76.00
```

Truncar datos atípicos

Genera una nueva columna llamada **`truncado`**, donde apliques la estrategia de truncado de atípicos.

Los valores mayores a 100 se truncarán con un valor de **`peso_kg`** igual a 100.

Los valores menores a 50 se truncarán con un valor de **`peso_kg`** igual a 50.

Revisa los registros donde se haya realizado truncado.

```
# Se realiza el truncado de datos. Lo que esté por abajo
# de 50 se ajusta a 50, y todo lo que esté por encima de
# 100, se ajusta a 100.
df['truncado']=df['peso_kg'].clip(lower=50,upper=100)

# Se muestran las filas que fueron truncadas.
df[['peso_kg','truncado']][(df['peso_kg'] != df['truncado'])]
```

```
    peso_kg  truncado
3     47.20      50.0
13   104.83     100.0
17    42.05      50.0
25   145.20     100.0
28   133.90     100.0
```

Suavizar atípicos usando transformación logarítmica

Genera una nueva columna llamada **`logaritmo`**, donde apliques la estrategia de ***transformación logarítmica de atípicos*** usando logaritmos.

Transforma logarítmicamente el valor de **`peso_kg`** y genera la columna **`logaritmo`** con el valor de la transformación.

Muestra los resultados de la transformación.

```
# Se importa la librería pandas, porque se requiere para
# el trabajo con logaritmos.
import numpy as np

# Se genera un cálculo logarítmico sobre la columna con
# atípicos. El valor de logaritmo es más suavizado, llevando el
valor de peso_kg más cercano a la normalidad.
df['logaritmo']=np.log(df['peso_kg'])

# Se muestran las filas que fueron truncadas.
print(df[['peso_kg','logaritmo']])
```

```
    peso_kg  logaritmo
0     67.80   4.216562
1     66.24   4.193285
2     88.00   4.477337
3     47.20   3.854394
4     69.00   4.234107
5     65.55   4.182813
6     72.96   4.289911
7     79.56   4.376511
8     84.36   4.435093
9     69.35   4.239166
10    71.28   4.266616
11    71.82   4.274163
12    81.65   4.402442
13   104.83   4.652340
14    68.08   4.220683
15    73.14   4.292375
16    53.40   3.977811
17    42.05   3.738859
18    70.07   4.249495
19    67.50   4.212128
20    62.70   4.138361
21    63.70   4.154185
22    81.37   4.399007
23    62.00   4.127134
24    75.19   4.320018
25   145.20   4.978112
…
```

Verificar normalidad usando la prueba Shapiro-Wilk

Utiliza la prueba ***Shapiro-Wilk*** para comprobar que la nueva columna ha suavizado las diferencias entre valores, y es más cercana a una distribución normal.

La prueba de normalidad estadística llamada ***Shapiro-Wilk***, que prueba la hipótesis nula de que los datos provienen de una distribución normal.

Si el valor **p** de la prueba es menor que un nivel de significancia predeterminado (como 0.05), se rechaza la hipótesis nula y se concluye que los datos no siguen una distribución normal.

```
# Se importa la librería scipy, porque se requiere para
# el trabajo con la pruea Shapiro-Wilk.
from scipy.stats import shapiro

# Realizar la prueba de Shapiro-Wilk, sobre peso_kg y logaritmo
# para comprobar que la transformación hizo tender los datos
# hacia la normalidad.
stat, p_peso = shapiro(df['peso_kg'])
stat, p_logaritmo = shapiro(df['logaritmo'])

# Imprimir el valor p y la conclusión
print(f'Valor p para peso_kg: {p_peso:.8f}')
print(f'Valor p para logaritmo: {p_logaritmo:.8f}')
```

```
Valor p para peso_kg: 0.00005744
Valor p para logaritmo: 0.01252854
```

Dados estos resultados, se puede concluir que la variable **`logaritmo`** es más cercana a una distribución normal que la variable **`peso_kg`**.

Esto se debe a que el valor **p** obtenido para la variable **`logaritmo`** es mayor que el nivel de significancia estándar de 0.05, lo que significa que no hay suficiente evidencia para rechazar la hipótesis nula de que los datos de la variable **`logaritmo`** provienen de una distribución normal.

Por otro lado, el valor **p** obtenido para la variable **`peso_kg`** es mucho menor que el nivel de significancia de 0.05, lo que sugiere que la hipótesis nula de normalidad puede ser rechazada en favor de una hipótesis alternativa.

Es importante tener en cuenta que los valores **p** no proporcionan una medida directa de qué tan cerca está una variable de una distribución normal, sino que indican la probabilidad de obtener los datos observados o datos más extremos bajo la hipótesis nula de que los datos provienen de una distribución normal. Por lo tanto, un valor de **p** menor indica que es menos probable que los datos provengan de una distribución normal.

Suavizar atípicos usando raíz cuadrada

También se puede suavizar los datos usando una ***transformación de raíz cuadrada de atípicos***.

```
# Se genera un cálculo de raíz cuadrada sobre la columna
# con atípicos. El valor de logaritmo es más suavizado,
# llevando el valor de peso_kg más cercano a la normalidad.
df['raíz']=np.sqrt(df['peso_kg'])

# Se muestran las filas que fueron truncadas.
print(df[['peso_kg','raíz']])
```

```
    peso_kg       raíz
0     67.80   8.234076
1     66.24   8.138796
2     88.00   9.380832
3     47.20   6.870226
4     69.00   8.306624
5     65.55   8.096295
6     72.96   8.541663
7     79.56   8.919641
8     84.36   9.184770
9     69.35   8.327665
10    71.28   8.442748
11    71.82   8.474668
12    81.65   9.036039
13   104.83  10.238652
14    68.08   8.251061
15    73.14   8.552193
16    53.40   7.307530
17    42.05   6.484597
18    70.07   8.370783
19    67.50   8.215838
20    62.70   7.918333
21    63.70   7.981228
22    81.37   9.020532
23    62.00   7.874008
24    75.19   8.671217
25   145.20  12.049896
…
```

```
# Realizar la prueba de Shapiro-Wilk, sobre peso_kg y raíz
# para comprobar que la transformación hizo tender los datos
# hacia la normalidad.
stat, p_raíz = shapiro(df['raíz'])

# Imprimir el valor p y la conclusión
print(f'Valor p para peso_kg: {p_peso:.8f}')
print(f'Valor p para raíz: {p_raíz:.8f}')
```

```
Valor p para peso_kg: 0.00005744
Valor p para raíz: 0.00099591
```

Qué transformación es más efectiva, ¿la logarítmica, o la raíz cuadrada?

En general, cuanto más grande sea el valor **p** obtenido de la prueba, mayor será la evidencia en contra de la hipótesis alternativa de que los datos no provienen de una distribución normal.

Es importante mencionar que Shapiro-Wilk no es una prueba que revise grados de normalidad. No podemos argumentar, usando solo Shapiro-Wilk, el grado en que es mejor **`logaritmo`** que **`raíz`**.

Lo que sí podemos decir es que los valores de **p** obtenidos para ambas variables (**`raíz`**, **`peso_kg`**) son muy pequeños, lo que sugiere que ambas muestras no provienen de una distribución normal.

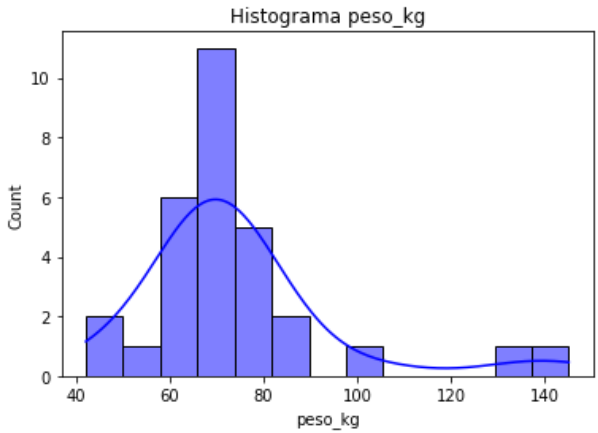
En este caso, sí podemos afirmar que **`logaritmo`** nos es más útil, pero sí los resultados hubieran sido diferentes, es decir, que **`raíz`** también fuera lo suficientemente alto como para suponer que provienen de una distribución normal, no sería claro cuál de las dos transformaciones nos es más útil.

Verificar normalidad usando histogramas

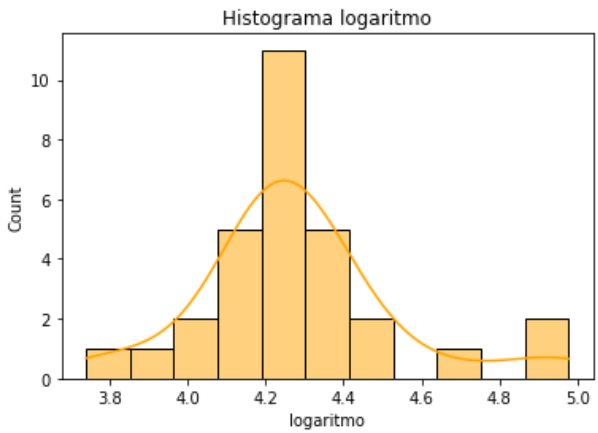
Grafica histogramas de **`peso_kg`**, **`logaritmo`** y **`raíz`**, para observar visualmente cuál tiene más a la curva normal.

Podemos generar una distribución de frecuencias, y revisar visualmente qué gráfica se ajusta más a una distribución normal. Aquí hay que ver las cimas, y lo pronunciado de estas, así como la centralidad.

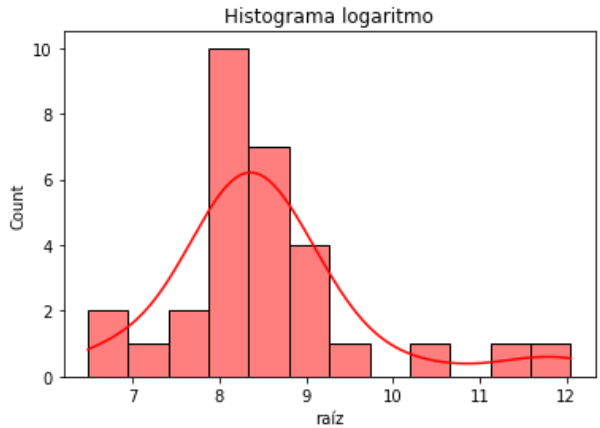
```
# Se importa la librerías par graficación.
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm

# Histograma de peso_kg
sns.histplot(df, x='peso_kg', color='blue',
      alpha=0.5, kde=True)
plt.title('Histograma peso_kg')
plt.show()
```




```
# Histogama de logaritmo
sns.histplot(df, x='logaritmo', color='orange',
      alpha=0.5, kde=True)
plt.title('Histograma logaritmo')
plt.show()
```



```
# Histogama de raíz
sns.histplot(df, x='raíz', color='red',
      alpha=0.5, kde=True)
plt.title('Histograma logaritmo')
plt.show()
```



Podemos comprobar que la columna **`logaritmo`** es la que más se ajusta a la curva normal.

Manejar atípicos usando técnica de *binning*

Genera una nueva columna llamada **`estado`**, donde apliques la estrategia de ***binning*** de atípicos usando **`cut()`**.

Establece los límites de las clases a partir de la información del mínimo, máximo, mínimo no atípico y máximo no atípico.

Muestra el resultado, y muestra una tabla de frecuencias para cada etiqueta del categórico.

```
# Determinamos el mínimo y el máximo.
mínimo=df['peso_kg'].min()
máximo=df['peso_kg'].max()

# Se establecen como límites el mínimo,
# Li, Ls, y el máximo.
limites=[mínimo,Li,Ls,máximo]
```

```
# Se asignan etiquetas para los atípicos arriba y abajo,
# y los típicos.
etiquetas=['ATIPICO BAJO','TÍPICO','ATÍPICO ALTO']

# Se generan las clases de intervalo, considerando los
# límites establecidos, con las etiquetas establecidas,
# e incluyendo el valor menor, para que no se excluye.
df['estado']=pd.cut(df['peso_kg'],
                    bins=limites,
                    labels=etiquetas,
                    include_lowest=True)

print(df[['peso_kg','estado']],'\n')
print(df['estado'].value_counts())
```

```
    peso_kg        estado
0     67.80        TÍPICO
1     66.24        TÍPICO
2     88.00        TÍPICO
3     47.20        TÍPICO
4     69.00        TÍPICO
5     65.55        TÍPICO
6     72.96        TÍPICO
7     79.56        TÍPICO
8     84.36        TÍPICO
9     69.35        TÍPICO
10    71.28        TÍPICO
11    71.82        TÍPICO
12    81.65        TÍPICO
13   104.83  ATÍPICO ALTO
14    68.08        TÍPICO
15    73.14        TÍPICO
16    53.40        TÍPICO
17    42.05  ATIPICO BAJO
18    70.07        TÍPICO
19    67.50        TÍPICO
20    62.70        TÍPICO
21    63.70        TÍPICO
22    81.37        TÍPICO
23    62.00        TÍPICO
24    75.19        TÍPICO
25   145.20  ATÍPICO ALTO
26    65.72        TÍPICO
27    57.95        TÍPICO
28   133.90  ATÍPICO ALTO
29    76.00        TÍPICO

TÍPICO          26
ATÍPICO ALTO     3
ATIPICO BAJO     1
Name: estado, dtype: int64
```

Este programa funciona si hay atípicos arriba y abajo.

Si no hay atípicos, la lógica se debe cambiar, pues el mínimo estará por encima de **`Li`**, y el máximo estaría por debajo de **`Ls`**, lo que produciría errores.

Manejar atípicos usando técnica de imputación

Genera una nueva columna llamada **`valor_imputado`**, donde apliques la estrategia de imputación de atípicos.

Genera la mediana del **`peso_kg`** sin incluir atípicos. Este valor aplicará como valor imputado.

Remplaza los valores atípicos por el valor imputado.

Muestra el resultado.

```
# Se calcula la mediana de peso_kg, sin incluir datos atípicos.
mediana=df.loc[(df['peso_kg']>=Li) &
               (df['peso_kg']<=Ls),'peso_kg'].median()

print("El valor de la mediana (valor imputado) es",
      f"{mediana:.2f}", '\n')

# Se genera una función definida por el usuario, donde se
# retorne el valor imputado, en lugar
def imputación(fila):
  valor=fila['peso_kg']
  if (fila['peso_kg']<Li or fila['peso_kg']>Ls):
    valor=mediana
  return valor

df['valor_imputado']=df.apply(imputación,axis=1)

print(df)
```

```
El valor de la mediana (valor imputado) es 69.17

    estatura_cm  peso_kg  truncado  logaritmo       raíz        estado  \
0           160    67.80     67.80   4.216562   8.234076        TÍPICO
1           172    66.24     66.24   4.193285   8.138796        TÍPICO
2           180    88.00     88.00   4.477337   9.380832        TÍPICO
3           170    47.20     50.00   3.854394   6.870226        TÍPICO
4           160    69.00     69.00   4.234107   8.306624        TÍPICO
5           169    65.55     65.55   4.182813   8.096295        TÍPICO
6           164    72.96     72.96   4.289911   8.541663        TÍPICO
7           178    79.56     79.56   4.376511   8.919641        TÍPICO
8           174    84.36     84.36   4.435093   9.184770        TÍPICO
9           173    69.35     69.35   4.239166   8.327665        TÍPICO
```

```
10          166    71.28     71.28   4.266616   8.442748        TÍPICO
11          163    71.82     71.82   4.274163   8.474668        TÍPICO
12          171    81.65     81.65   4.402442   9.036039        TÍPICO
13          180   104.83    100.00   4.652340  10.238652  ATÍPICO ALTO
14          174    68.08     68.08   4.220683   8.251061        TÍPICO
15          169    73.14     73.14   4.292375   8.552193        TÍPICO
16          160    53.40     53.40   3.977811   7.307530        TÍPICO
17          166    42.05     50.00   3.738859   6.484597  ATIPICO BAJO
18          177    70.07     70.07   4.249495   8.370783        TÍPICO
19          175    67.50     67.50   4.212128   8.215838        TÍPICO
20          166    62.70     62.70   4.138361   7.918333        TÍPICO
21          165    63.70     63.70   4.154185   7.981228        TÍPICO
22          179    81.37     81.37   4.399007   9.020532        TÍPICO
23          162    62.00     62.00   4.127134   7.874008        TÍPICO
24          173    75.19     75.19   4.320018   8.671217        TÍPICO
25          171   145.20    100.00   4.978112  12.049896  ATÍPICO ALTO
26          162    65.72     65.72   4.185403   8.106787        TÍPICO
27          161    57.95     57.95   4.059581   7.612490        TÍPICO
28          171   133.90    100.00   4.897093  11.571517  ATÍPICO ALTO
29          180    76.00     76.00   4.330733   8.717798        TÍPICO

    valor_imputado
0           67.800
1           66.240
2           88.000
3           47.200
4           69.000
5           65.550
6           72.960
7           79.560
8           84.360
9           69.350
10          71.280
11          71.820
12          81.650
13          69.175
14          68.080
15          73.140
16          53.400
17          69.175
18          70.070
19          67.500
20          62.700
21          63.700
22          81.370
23          62.000
24          75.190
25          69.175
26          65.720
27          57.950
28          69.175
29          76.000
```

FIN DEL LAB

LAB 13.02: Tratamiento de datos atípicos y ausentes para el Titanic

En este Lab se desea eliminar las filas con atípicos respecto a la edad de las personas que viajaban en el Titanic.

Las tareas por realizar son:

1. Determinar el mínimo no atípico y máximo no atípico de la columna **edad**.

```
# Datos base
import pandas as pd
tipos_correctos={
    'id_persona':object,
    'clave_sobrevivencia':object,
    'clase_viaje':object
}

personas_titanic_csv='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
personas_titanic_v5.csv'

titanic = pd.read_csv(personas_titanic_csv,
      dtype=tipos_correctos)

titanic['id_persona']=titanic['id_persona'].str.split('.').
      str.get(0)
titanic['clave_sobrevivencia']=titanic['clave_sobrevivencia'].
      str.split('.').str.get(0)
titanic['clase_viaje']=titanic['clase_viaje'].str.split('.').
      str.get(0)

print('Número de filas del conjunto de datos: ', len(titanic))
```

```
Número de filas del conjunto de datos:  1310
```

Tratar los datos ausentes por eliminación

Dado que la muestra estadística es de 511 y tenemos bastante más datos que esos, se decide ejecutar como estrategia de tratamiento de ausentes la eliminación de las filas que los contengan.

```
# Por comodidad, se enumeran en una lista los campos
# requeridos, que son a fin de cuentas los que pueden
# considerarse con datos ausentes.
requeridos=['sobrevivencia','clase_viaje','sexo',
      'rango_edad','acompañada']

# Se eliminan aqullas filas donde alguno de los campos
# requeridos tenga datos ausentes.
print('Filas antes de eliminación de ausentes:',
      len(titanic))

titanic.dropna(subset=requeridos, inplace=True)

print('Filas después de eliminación de ausentes:',
      len(titanic))
```

```
Filas antes de eliminación de ausentes: 1310
Filas después de eliminación de ausentes: 1040
```

```
# Se verifica que se tienen 1040 filas con todos los datos
# requeridos disponibles.
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1040 entries, 0 to 1306
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   id_persona           1040 non-null   object
 1   clave_sobrevivencia  1040 non-null   object
 2   nombre               1040 non-null   object
 3   sexo                 1040 non-null   object
 4   clase_viaje          1040 non-null   object
 5   id_bote              415 non-null    object
 6   id_cuerpo            117 non-null    float64
 7   edad                 1040 non-null   float64
 8   cantidad_parientes   1040 non-null   int64
 9   cantidad_familiares  1040 non-null   int64
 10  acompañantes         1040 non-null   int64
 11  rango_edad           1040 non-null   object
 12  sobrevivencia        1040 non-null   object
```

```
 13  clase                1040 non-null   object
 14  acompañada           1040 non-null   object
dtypes: float64(2), int64(3), object(10)
memory usage: 130.0+ KB
```

Tratar los datos atípicos por eliminación

Dado que la muestra estadística es de 511 y tenemos bastante más datos que esos, se decide ejecutar como estrategia de tratamiento de datos atípicos la eliminación de las filas que los contengan.

```
# Se calcula el mínimo no atípico (Li) y máximo no atípico (Ls)
# del campo edad.
Q1_edad=titanic['edad'].quantile(0.25)
Q3_edad=titanic['edad'].quantile(0.75)
RIC_edad=(Q3_edad-Q1_edad)
Li_edad=Q1_edad-(1.5*RIC_edad)
Ls_edad=Q3_edad+(1.5*RIC_edad)

print('El mínimo no atípico de la edad es: ', Li_edad)
print('El máximo no atípico de la edad es: ', Ls_edad)
```

```
El mínimo no atípico de la edad es:  -6.0
El máximo no atípico de la edad es:  66.0
```

```
# Eliminación de filas con atípicos.
print('Filas antes de quitar atípicos:', len(titanic))

titanic=titanic[(titanic['edad']>=Li_edad) &
      (titanic['edad']<=Ls_edad)]

print('Filas antes de quitar atípicos:', len(titanic))
```

```
Filas antes de quitar atípicos: 1040
Filas antes de quitar atípicos: 1032
```

Eliminar campos no requeridos

Ya no tiene sentido que mantengamos en el conjunto de datos a los requeridos indirectos, por lo cual, se dejan solo los campos requeridos.

```
# Se actualiza titanic, dejándolo solo con los campos
# requeridos.
titanic=titanic[requeridos]

# Se verifica el resultado, mostrando los campos que
# permanecen.
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1032 entries, 0 to 1306
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   sobrevivencia  1032 non-null   object
 1   clase_viaje    1032 non-null   object
 2   sexo           1032 non-null   object
 3   rango_edad     1032 non-null   object
 4   acompañada     1032 non-null   object
dtypes: object(5)
memory usage: 48.4+ KB
```

```
# Se muestran los datos que nos permiten hacer trabajos
# de analítica, de acuerdo con el objetivo de análisis
# establecido.
titanic
```

```
     sobrevivencia clase_viaje    sexo       rango_edad  acompañada
0            MURIÓ           1  HOMBRE     MEDIANA EDAD        SOLA
3            MURIÓ           1  HOMBRE     MEDIANA EDAD        SOLA
4            VIVIÓ           1  HOMBRE  ADULTOS MAYORES        SOLA
6            MURIÓ           1  HOMBRE     MEDIANA EDAD        SOLA
7            MURIÓ           1  HOMBRE          ADULTOS        SOLA
...            ...         ...     ...              ...         ...
1288         VIVIÓ           3  HOMBRE          ADULTOS        SOLA
1293         VIVIÓ           3  HOMBRE          ADULTOS        SOLA
1294         MURIÓ           3  HOMBRE          ADULTOS        SOLA
1296         MURIÓ           3  HOMBRE          JOVENES  ACOMPAÑADA
1306         MURIÓ           3  HOMBRE  ADULTOS MAYORES        SOLA

[1032 rows x 5 columns]
```

Guardado de archivo final, en formato CSV y Excel

El conjunto de datos ya está listo para su procesamiento con técnicas de analítica. Se guarda en formato CSV (**`titanic_final.csv`**) y Microsoft Excel (**`titanic_final.xlsx`**).

```
# Se guarda el DataFrame en un archivo CSV, sin incluir índice.
titanic.to_csv('titanic_final.csv', index=False)

# Se guarda el DataFrame en un archivo XLSX, sin incluir
# índice, en una hoja llamada datos.
titanic.to_excel('titanic_final.xlsx',
      index=False, sheet_name='datos')
```

Revisa en **Google Colab**, haciendo clic en el ícono de carpeta que está en la lateral izquierda, si se generaron los archivos.

FIN DEL LAB

CAPÍTULO 14:

Muestreo aleatorio simple y muestreo estratificado

LAB 14.01: Muestra aleatoria simple y muestra estratificada

En este Lab se utilizarán todas las técnicas aprendidas para realizar el muestreo aleatorio simple y el muestreo estratificado, sobre los datos de la cámara de bienes raíces del condado King County, en Washington; se considerarán las operaciones de venta de los años 2022 y 2023 de los agentes inmobiliarios GINA JEANNOT, FRANK PAINTER – NEOHOMES, y SKYLINE PROPERTIES, que ya han sido limpiados y tratados en otros ejercicios. En otras palabras, ya disponemos de los datos limpios, sin atípicos y ausentes.

Las tareas por realizar son:

1. Recuperar los datos previamente limpiados y tratados.
2. Determinar el tamaño de la muestra estadística.
3. Determinar la muestra aleatoria simple.
4. Determinar si la muestra es representativa.
5. Generación de la muestra estratificada.
6. Verificar que hubo mejora en la representatividad de los estratos.

Recuperar los datos previamente limpiados y tratados

Lo primero que hacemos es recuperar los datos limpios y tratados, que se encuentran en **GitHub**.

```
import pandas as pd

# Se almacena en una variable la liga de acceso a los
# datos master que se encuentran en Github
url_master='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
operaciones_master.csv'
```

```
# Se enumeran los tipos de datos correctos para los datos.
tipos_esperados={
  'id_operacion':str,
  'fecha_operacion':object,
  'precio':float,
  'public_id':str,
  'codigo_frente_agua':str,
  'codigo_vista':str,
  'codigo_condicion':str,
  'agente_id':str,
  'codigo_pais':str,
  'año':str,
  'año_mes':str,
  'mes':str,
  'vista':str,
  'condición':str,
  'país':str,
  'agente':str,
  'anio_construc':str,
  'latitud':str,
  'longitud':str,
  'codigo_postal':str,
  'm2_construccion':float,
  'm2_terreno':float,
  'm2_sobre_calle':float,
  'm2_sotano':float,
  'pisos':float,
  'recamaras':float,
  'banios':float,
  'anio_renovacion':str
}

# Se leen los datos y se cargan en un DataFrame
operaciones_master=pd.read_csv(url_master,
      dtype=tipos_esperados)

# La columna fecha_operacion se recuperó como
# object, y se transforma a datetime
operaciones_master['fecha_operacion']=pd.to_datetime(
    operaciones_master['fecha_operacion'])

# Se muestra el resultado
operaciones_master.shape
```

```
(6891, 28)
```

Determinar el tamaño de la muestra estadística

Como podemos comprobar, tenemos 6891 filas, con 28 registros.

Suponiendo que queremos un 99% de confianza con un margen de error del 5%, el tamaño de la muestra se calcularía así.

```
# Declara las variables, cuidando que sean del tipo correcto.
N=len(operaciones_master)
p=0.50
q=1-p
E=0.05
Z=2.576

# Se codifica la fórmula, para calcular el tamaño de la
# muestra (n), y muestra el resultado.
# Toma en cuenta la propiedad conmutativa 'Cuando
# multiplicamos, el orden de los factores no afecta
# al producto'.
n=int((Z**2*p*q*N)/((N*E**2)+(Z**2*p*q)))

print(f'El tamaño de la muestra es {n}')
```

```
El tamaño de la muestra es 605
```

Determinar la muestra aleatoria simple

Entonces, el tamaño de la muestra es 605.

Para generar una muestra aleatoria simple, se utiliza el método **`sample()`**

```
# Se calcula la muestra aleatoria simple, considerando
# un tamaño de la muestra de 605.
muestra_aleatoria_simple=operaciones_master.sample(n=605)
```

Ahora, el DataFrame **`muestra_aleatoria_simple`** contiene 605 observaciones, que representan la muestra aleatoria simple de la población.

Determinar si la muestra es representativa

Vamos a suponer que deseamos que la muestra esté estratificada por el tipo de condición en que se encuentra la propiedad, es decir, por la columna **`condición`**.

Dado que la muestra estratificada pretende reproducir la representatividad que cada estrato tiene, lo primero que debemos determinar es precisamente cuál es la representatividad por reproducir.

Esto lo logramos conociendo la frecuencia relativa que tiene la característica de estratificación en la población. No hay más representatividad más precisa que esa.

```
# Se define cuál es la columna de referencia para
# la estratificación.
columna='condición'

# Se muestra la tabla de frecuencias absolutas y relativas.
original=operaciones_master[columna].value_counts()
for condicion, fi in original.items():
  hi=fi/len(operaciones_master)
  print(f"{condicion:30s} {fi:10d} {hi:0.4%}")
```

```
C-BUENAS CONDICIONES                4460 64.7221%
B-BUENAS CONDICIONES                1844 26.7595%
A-EXCELENTES CONDICIONES             508 7.3719%
D-CONDICIONES REGULARES               67 0.9723%
E-MALAS CONDICIONES                   12 0.1741%
```

La Serie pandas llamada **`original`** contiene las frecuencias absolutas para la característica de estratificación, a partir de la población.

Posteriormente, determinamos las frecuencias absolutas y relativas para la característica de estratificación, a partir de la muestra aleatoria simple que ya hemos generado.

```
muestra=muestra_aleatoria_simple[columna].value_counts()

for condicion, fi in muestra.items():
  hi=fi/len(muestra_aleatoria_simple)
  print(f"{condicion:30s} {fi:10d} {hi:0.4%}")
```

```
C-BUENAS CONDICIONES                387 63.9669%
B-BUENAS CONDICIONES                170 28.0992%
A-EXCELENTES CONDICIONES             42 6.9421%
D-CONDICIONES REGULARES               5 0.8264%
E-MALAS CONDICIONES                   1 0.1653%
```

La Serie pandas llamada **muestra** contiene ahora las frecuencias absolutas para la característica de estratificación, a partir de la muestra.

En este caso, podemos darnos cuenta de que hay discrepancias, pero no son muy significativas; desde luego, habrá casos en donde puede presentarse un mayor problema.

Generación de la muestra estratificada

Si queremos estratificar, de tal manera que la representatividad en una población y en una muestra sean lo más pequeñas posibles, podemos generar muestras aleatorias sobre los estratos, en la proporción que les corresponde a cada uno.

```
from pandas.core.internals import concat
import math

# Se especifica la característica de estratificación.
columna='condición'

# Toma en cuenta que hi, la frecuencia relativa,
# representa la proporción de cada condición
# en la población.
original=operaciones_master[columna].value_counts()
filas=len(operaciones_master)
tamaño_muestra=605

# Se genera un DataFrame con la estructura del DataFrame
# base. Se genera vacío, con la finalidad de irle agregando
# las filas producto del muestreo.
muestra_estratificada=pd.DataFrame(
      columns=operaciones_master.columns)
```

```
# Se lee de forma secuencial la tabla de frecuencias,
# para determinar cuántas observaciones se deben
# tomar de cada estrato.
for categoria, fi in original.items():
  hi=fi/filas
  # Generamos un conjunto de datos que solo contenga
  # las filas del segmento.
  segmento=operaciones_master[(
      operaciones_master[columna]==categoria)]

  # Se genera una muestra aleatoria simple, sobre
  # las filas correspondientes al estrato, usando
  # la frecuencia relativa (hi) como proporción.
  elementos_representativos=math.ceil(tamaño_muestra*hi)
  muestra=segmento.sample(n=elementos_representativos)

  # Se agregan las filas seleccionadas a la muestra
  # estratificada. Se irán acumulando las filas
  # representativas de cada estrato.
  muestra_estratificada=pd.concat(
      [muestra_estratificada,muestra],
      ignore_index=True)

  # Se muestran las filas que cumplen con el criterio,
  # las filas que constituyen la muestra del segmento,
  # las filas que se acumularán a la muestra estratificada,
  # y las filas acumuladas de la muestra estratificada.
  # Aquí se aprecia cómo va creciendo la muestra.
  print(filas, len(segmento), len(muestra),
        len(muestra_estratificada))

# Se muestra la forma final que tiene la muestra estratificada.
print(muestra_estratificada.shape)
```

```
6891 4460 392 392
6891 1844 162 554
6891 508 45 599
6891 67 6 605
6891 12 2 607
(607, 28)
```

Ahora, determinamos la tabla de frecuencia de la muestra estratificada, para ver si la representatividad mejoró.

```
# Se genera la tabla de frecuencias con la muestra
# estratificada, para comprobar que se acerca más
# a la representatividad deseada.
# Es muy poco probable que coincidan exactamente,
# debido a que las muestras aleatorias seleccionan
# diferentes filas cada vez, y el tamaño de observaciones
# a tomar por segmento se redondean.
muestra_est=muestra_estratificada[columna].value_counts()

for condicion, fi in muestra.items():
  hi=fi/len(muestra_aleatoria_simple)
  print(f"{condicion:30s} {fi:10d} {hi:0.4%}")
```

```
C-BUENAS CONDICIONES                392 64.7934%
B-BUENAS CONDICIONES                162 26.7769%
A-EXCELENTES CONDICIONES             45 7.4380%
D-CONDICIONES REGULARES               6 0.9917%
E-MALAS CONDICIONES                   2 0.3306%
```

La Serie pandas llamada **muestra_est** contiene ahora las frecuencias absolutas para la característica de estratificación, a partir de la muestra.

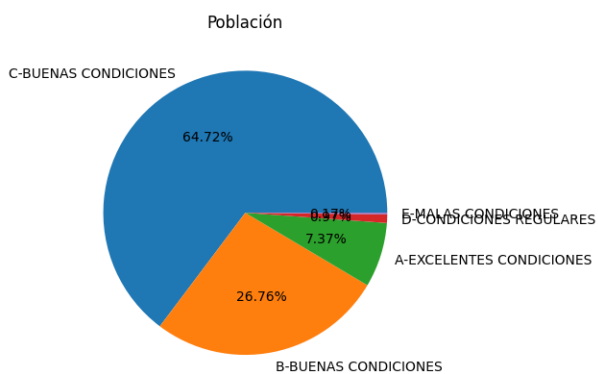
Verificar que hubo mejora en representatividad de los estratos

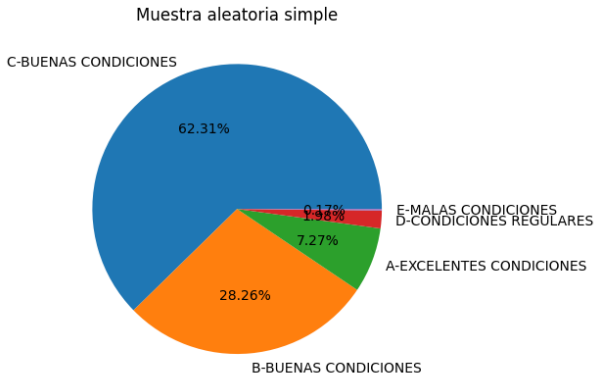
Ahora verificaremos visualmente cuál de las dos muestras, aleatoria simple, o estratificada, representa más acertadamente a los estratos en la población.

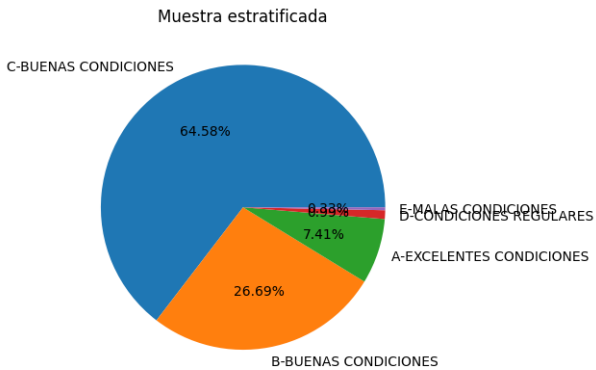
Esta es la distribución de segmentos para la población, usado como característica de estratificación a **condición**.

Para esta parte, usaremos la librería **matplotlib**. Generaremos 3 gráficas de sectores, a partir de Series pandas que contienen las tablas de frecuencias correspondiente a la característica de estratificación, para la población (**original**), para la muestra aleatoria simple (**muestra**) y para la muestra estratificada (**muestra_est**).

```
# Librería para poder graficar.
import matplotlib.pyplot as plt

# Creamos la gráfica de sectores, tomando la serie
# original, que contiene la tabla de frecuencias de
# la característica de estratificación, a partir de
# la población.
plt.pie(original, labels=original.index, autopct='%1.2f%%')
plt.title('Población')

# Se genera una nueva gráfica.
plt.figure()
# Creamos la gráfica de sectores, tomando la serie
# muestra, que contiene la tabla de frecuencias de
# la característica de estratificación, a partir de
# la muestra aleatoria simple.
plt.pie(muestra, labels=muestra.index, autopct='%1.2f%%')
plt.title('Muestra aleatoria simple')

# Se genera una nueva gráfica.
plt.figure()
# Creamos la gráfica de sectores, tomando la serie
# muestra, que contiene la tabla de frecuencias de
# la característica de estratificación, a partir de
# la muestra aleatoria simple.
plt.pie(muestra_est, labels=muestra_est.index,
      autopct='%1.2f%%')
plt.title('Muestra estratificada')

# Se muestran los gráficos
plt.show()
```







Como puede observarse, la muestra estratificada tiene una representatividad más cercana a la de la población, que la que presenta la muestra aleatoria simple.

FIN DEL LAB

CAPÍTULO 15:

Serialización JSON y Pickle

LAB 15.01: Serialización de un DataFrame usando Pickle

En este Lab se utilizarán las técnicas de serialización y deserialización, para comprobar que se puede usar Pickle para el transporte de información sin pérdida o corrupción de contenidos.

Las tareas por realizar son:

1. Recuperar los datos previamente limpiados y tratados.
2. Serializar a pickle y almacenar en un archivo.
3. Leer desde un archivo pickle y deserializar.

Recuperar los datos previamente limpiados y tratados

Lo primero que hacemos es recuperar los datos limpios y tratados, que se encuentran en **GitHub**.

```
import pandas as pd

# Se almacena en una variable la liga de acceso a los
# datos master que se encuentran en Github
url_master='https://raw.githubusercontent.com/
AprendaPracticando/AnaliticaPythonR1/main/data/
operaciones_master.csv'

# Se leen los datos y se cargan en un DataFrame
origen=pd.read_csv(url_master,
      dtype=tipos_esperados)

# Se muestra el resultado
origen.shape
```

```
(6891, 28)
```

Serializar a pickle y almacenar en un archivo

Se serializa a pickle el DataFrame, y se almacena en un archivo de extensión **`.pickle`**

```
# Librería para poder usar pickle
import pickle

# Se abrirá un archivo llamado datos.pickle, en modo
# write (w) binary (b) en donde, si existe lo remplaza
# y si no existe lo crea (+). El archivo tendrá un
# apuntador llamado f.
with open("datos.pickle","wb+") as f:
    # Se serializa usando pickle el contenido del objeto
    # llamado origen, y se guarda en el archivo.
    pickle.dump(origen,f)

# El archivo ya debe existir en el ambiente.
```

Leer desde un archivo pickle y deserializar

Se lee el contenido de un archivo binario de extensión **`.pickle`**, que contiene un objeto serializado usando pickle. Se deserializa, y se obtiene el objeto original.

```
# Se abrirá un archivo llamado datos.pickle, en modo
# read (r) binary (b). El archivo tendrá un
# apuntador llamado f.
with open("datos.pickle","rb") as f:
    destino=pickle.load(f)

# Se comprueba que el nuevo objeto es un DataFrame, con
# el contenido original.
destino.shape
```

```
(6891, 28)
```

FIN DEL LAB

APRENDA EDICIONES

SAN FÉLIX 5432, DESPACHO “D”

VISTA SOL, GUADALUPE, NUEVO LEÓN, MÉXICO

FECHA DE IMPRESIÓN: 21/ABRIL/2023