# Módulo 4. Análisis e interpretación de datos

## Introducción

En el procesamiento de datos, una vez que hemos recolectado los datos y los organizamos, se hace necesario el análisis exploratorio de los mismos para obtener un conocimiento básico de los datos y de las relaciones entre ellos.

Luego, se realizará el análisis e interpretación de los datos para hacer la conversión de datos a información útil.

Python ofrece diversas funciones para explorar el *dataset* y las estudiaremos en este módulo.

Conoceremos las diversas herramientas para generar gráficos con Python, sus librerías y cómo abordar su interpretación.

Para comenzar, veamos un video que nos introduce en el análisis de *datasets*.

**Video 1. ¿Qué es el análisis exploratorio de datos?**

Fuente: Comunicación Numérica (14 de febrero de 2021). ¿Qué es el análisis exploratorio de datos? [archivo de video]. YouTube. https://bit.ly/3SVNsKh.

## Video inmersión

## Unidad 1 Tipos de datos y cómo analizarlos

### Tema 1. Análisis exploratorio de datos

El análisis exploratorio de datos (*exploratory data analysis*) es una forma de analizar datos definida por John W. Tukey (1915-2000), estadístico estadounidense nacido en New Bedford, Massachusetts y considerado uno de los pioneros de la ciencia de datos.

El EDA (*exploratory data analysis*) se refiere a un conjunto de técnicas estadísticas cuya finalidad es conseguir un conocimiento básico de los datos y de las posibles relaciones entre ellos.

> Es el tratamiento estadístico al que se someten las muestras recogidas durante un proceso de investigación en cualquier campo científico. . . Con el fin de optimizar y obtener una mayor rapidez y precisión en todo el proceso estadístico, se suele recurrir a distintos softwares con aplicaciones específicas para el tratamiento estadístico. (Rodríguez et al., 2020, p. 7)

La idea clave del análisis exploratorio de datos es que el primer paso (y el más importante en cualquier proyecto basado en datos) es la observación de los datos.

**Cuando tenemos un *dataset* en el proceso, deberíamos hacernos algunas preguntas claves como:**

- ¿Existe algún tipo de estructura en los datos?
- ¿Dónde están centrados?
- ¿Cuál es la dispersión?
- ¿Cuál es la relación entre ellos?
- ¿Cómo sintetizar y presentar la información?
- ¿Hay sesgo en los datos seleccionados?
- ¿Hay errores de codificación?
- ¿Hay datos atípicos? ¿qué hacer con ellos?
- ¿Hay datos faltantes? ¿Qué hacer para recuperarlos? ¿Tiene algún patrón?

**Usualmente, un EDA sigue una serie de pasos:**

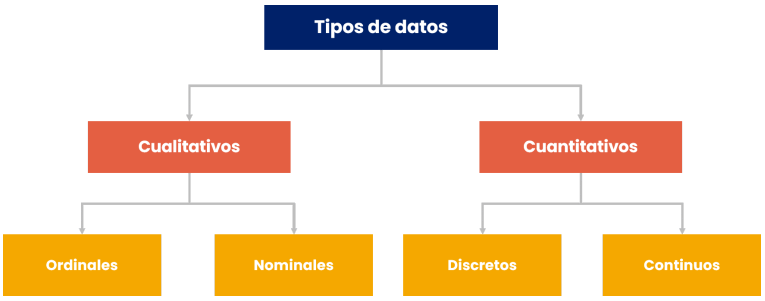
1. Familiarizarse con la naturaleza de los datos a analizar.
2. Realizar un examen gráfico y descriptivo de las variables.
3. Realizar un examen gráfico y descriptivo de la relación entre variables.
4. Preparar los datos para aplicar cualquier técnica.
5. Identificar datos atípicos y evaluar el impacto que estos provocan en el análisis.
6. Analizar el efecto de los datos faltantes y tomar decisiones.

Finalmente, para conectar lo anterior, la estadística descriptiva está formada por métodos gráficos y numéricos que se utilizan para resumir, procesar los datos y transformarlos en información.

Ahora bien, dependiendo del tipo de datos que tengamos para procesar y manipular, será el tipo de herramienta que debemos utilizar. Veamos los tipos de datos:

## Tema 2. Tipos de datos

**Figura 1: Tipos de datos**




Fuente: elaboración propia.

Cuando el estudio se centra en datos cualitativos, la mejor herramienta es una tabla de frecuencia que muestre un recuento de los datos. Esta tabla puede ser absoluta o relativa, siendo la última mucho más adecuada.

Los porcentajes, índices, razones y tasas son los valores más adecuados para referirse a las categorías de la variable. Veamos de qué se trata cada uno.

- **El porcentaje** es “una proporción que toma como referencia el número 100” (Real Academia Española, 2022, https://bit.ly/3JxxBP3). Es un número entre 0 y 100 que mide la proporción de un total. Por ejemplo, en un examen, un alumno contestó correctamente 90 preguntas de un total de 100. Entonces, el porcentaje de respuestas correctas es 90 sobre 100, es decir, 90 %. A su vez, el porcentaje de respuestas incorrectas es 10 sobre 100, es decir un 10 %.
- **Una razón** es “el cociente de dos números o, en general, de dos cantidades comparables entre sí” (Real Academia Española, 2022, https://bit.ly/3kXtceE). Por ejemplo, podemos calcular la razón de ventas (2000) por vendedor (10) dividiendo el número de productos totales vendidos por el número de vendedores. Es decir, 2000 en 10, obteniendo un total de 200 productos vendidos por vendedor.

- **Una tasa** es definida a partir de la fracción entre dos magnitudes, en general, relacionadas. Esto quiere decir que el denominador debe contener al numerador. Existen diferentes tasas: de mortalidad, de incidencias, de natalidad, de fertilidad, de interés, entre otras. Por ejemplo, una tasa de interés del 2 %, quiere decir que se cobrarán 2 pesos por cada 100 pesos prestados.
- **Un índice** es una expresión numérica de la relación entre dos cantidades. Un índice en estricto rigor es un valor que puede ser una proporción, razón, una tasa o un valor dentro de un contexto que permite indicar una situación.

**Ejemplo:** el gerente de la empresa necesita conocer en qué porcentaje aumentaron los ingresos de este año (1200) en relación con el anterior (1000). Entonces, el resultado sería, 1200 sobre 1000, multiplicado por 100, igual a 120. Es decir, los ingresos aumentaron en un 20 %.

Si los datos poseen un orden (**datos ordinales**), la tabla se ordena naturalmente. Por su parte, cuando los datos son **nominales**, es bueno usar un criterio sencillo para ordenarlos (orden alfabético, por ejemplo).

**Python proporciona una variedad de tipos de datos especializados, por ejemplo:**

- fechas y horas;
- matrices de tipo fijo (*fixed-type arrays*);
- colas de montículos. “En computación, un montículo (*heap* en inglés) es una estructura de datos del tipo árbol con información perteneciente a un conjunto ordenado” (Sensagent Corporation, 2013, https://bit.ly/3ZPnPwJ);
- colas de doble extremo (es una colección ordenada de datos que permite insertar y eliminar elementos por ambos extremos);
- enumeraciones;
- algunos tipos de datos integrados, como *dict, list, set y frozenset, tuple;*
- la clase *str* se utiliza para contener cadenas de caracteres Unicode;
- las clases *bytes* y *bytearray* se utilizan para contener datos binarios.

**Para investigar más acerca de los tipos de datos en Python, recomendamos ver la página de documentación: https://bit.ly/41URCWl.**

## Actividad de repaso

**Lea la siguiente afirmación y seleccione la opción correcta: "Es un número entre 0 y 100 que mide la proporción de un total".**

Razón.

Tasa.

Porcentaje.

**Justificación**

## Tema 3. Funciones para explorar el *dataset*

Un objetivo fundamental de un EDA (análisis exploratorio de datos) es **detectar valores atípicos y errores.** A la vez, se deben identificar diferentes **patrones en los datos**, ya que esto permite comprender mejor los datos antes de hacer suposiciones. Los resultados del EDA ayudan a las empresas a conocer a sus clientes, expandir su negocio y tomar mejores decisiones.

**Para entender mejor el EDA, veamos un ejemplo práctico. Usaremos un dataset de automóviles.** Te recomendamos que lo descargues directamente desde el siguiente enlace: https://bit.ly/3L5HT9Y.

**Importamos librerías y leemos el *dataset***

```
>>> import numpy as np
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
```

**Figura 2. Dataset (1)**

```
In [78]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
import seaborn as sns
auto=pd.read_csv(r'C:\Users\gonza\Downloads\Automobile_data.csv')
auto.head()
```

Out[78]:

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | engine-size | fuel-system | bore | stroke | compression-ratio | horsepo |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | |
| 1 | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | |
| 2 | 1 | ? | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 | ... | 152 | mpfi | 2.68 | 3.47 | 9.0 | |
| 3 | 2 | 164 | audi | gas | std | four | sedan | fwd | front | 99.8 | ... | 109 | mpfi | 3.19 | 3.4 | 10.0 | |
| 4 | 2 | 164 | audi | gas | std | four | sedan | 4wd | front | 99.4 | ... | 136 | mpfi | 3.19 | 3.4 | 8.0 | |

5 rows × 26 columns

Fuente: elaboración propia.

Podemos notar que, al leer el archivo desde el ordenador, la ruta que declaremos va a depender de cada uno. En nuestro caso, la ruta puede ser: “C:\Users\IPP\Downloads\Automobile_data.csv”.

Se agrega previamente el carácter “r” antes de la ruta para que pueda leer los “\” de la ruta.

Al cargarlo previamente, en este caso en Jupyter Notebook, podemos leerlo directamente y obtendremos el mismo resultado:

**Figura 3. Dataset (2)**

```
In [79]:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
import seaborn as sns
auto=pd.read_csv('Automobile_data.csv')
auto.head()
```

Out[79]:

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | engine-size | fuel-system | bore | stroke | compression-ratio | horsepo |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | |
| 1 | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | |
| 2 | 1 | ? | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 | ... | 152 | mpfi | 2.68 | 3.47 | 9.0 | |
| 3 | 2 | 164 | audi | gas | std | four | sedan | fwd | front | 99.8 | ... | 109 | mpfi | 3.19 | 3.4 | 10.0 | |
| 4 | 2 | 164 | audi | gas | std | four | sedan | 4wd | front | 99.4 | ... | 136 | mpfi | 3.19 | 3.4 | 8.0 | |

5 rows × 26 columns

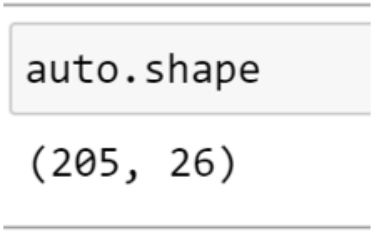
Fuente: elaboración propia.

Con la función “head()” podemos visualizar por pantalla las primeras cinco filas del *dataframe* a menos que indiquemos el número de filas. Por ejemplo: head(10), donde mostraremos por pantalla las primeras 10 filas.

Veamos de qué tamaño es el *dataset*. Recordemos que se puede revisar con la función

“*shape*”:

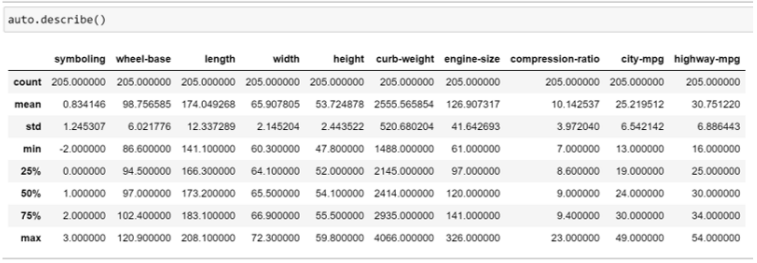
**Figura 4. *Dataset* (3)**




Fuente: elaboración propia.

Luego, podemos crear una vista general de los valores con **la función “*describe*()”. Lo que nos entregará conteo, media, desviación estándar, min, cuartiles Q1, Q2, Q3 y valor máximo.**

**Figura 5. *Dataset* (4)**



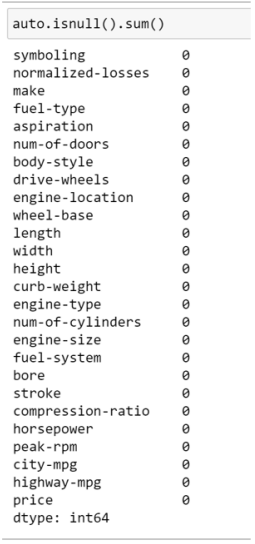
```
auto.describe()
```

| | symboling | wheel-base | length | width | height | curb-weight | engine-size | compression-ratio | city-mpg | highway-mpg |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 | 205.000000 |
| mean | 0.834146 | 98.756585 | 174.049268 | 65.907805 | 53.724878 | 2555.565854 | 126.907317 | 10.142537 | 25.219512 | 30.751220 |
| std | 1.245307 | 6.021776 | 12.337289 | 2.145204 | 2.443522 | 520.680204 | 41.642693 | 3.972040 | 6.542142 | 6.886443 |
| min | -2.000000 | 86.600000 | 141.100000 | 60.300000 | 47.800000 | 1488.000000 | 61.000000 | 7.000000 | 13.000000 | 16.000000 |
| 25% | 0.000000 | 94.500000 | 166.300000 | 64.100000 | 52.000000 | 2145.000000 | 97.000000 | 8.600000 | 19.000000 | 25.000000 |
| 50% | 1.000000 | 97.000000 | 173.200000 | 65.500000 | 54.100000 | 2414.000000 | 120.000000 | 9.000000 | 24.000000 | 30.000000 |
| 75% | 2.000000 | 102.400000 | 183.100000 | 66.900000 | 55.500000 | 2935.000000 | 141.000000 | 9.400000 | 30.000000 | 34.000000 |
| max | 3.000000 | 120.900000 | 208.100000 | 72.300000 | 59.800000 | 4066.000000 | 326.000000 | 23.000000 | 49.000000 | 54.000000 |

Fuente: elaboración propia.

Ahora, revisemos los valores faltantes en nuestro *dataset*.

**Figura 6. *Dataset* (5)**



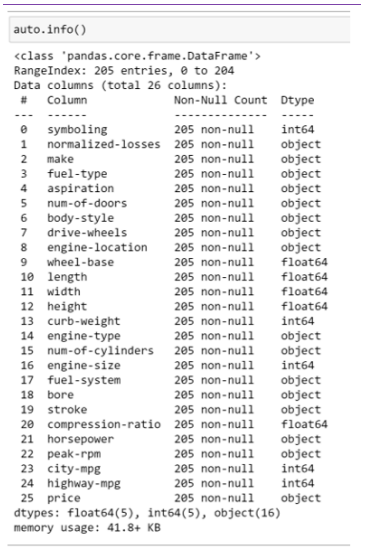
```
auto.isnull().sum()
```

```
symboling           0
normalized-losses   0
make                0
fuel-type           0
aspiration          0
num-of-doors        0
body-style          0
drive-wheels        0
engine-location     0
wheel-base          0
length              0
width               0
height              0
curb-weight         0
engine-type         0
num-of-cylinders    0
engine-size         0
fuel-system         0
bore                0
stroke              0
compression-ratio   0
horsepower          0
peak-rpm            0
city-mpg            0
highway-mpg         0
price               0
dtype: int64
```

Fuente: elaboración propia.

Esto indica que no tenemos ningún valor faltante en las 26 columnas. Puede suceder que encontremos algunos valores “?” en la variable “normalized-losses”. Por esa razón es importante revisar los tipos de datos en cada atributo (variable).

**Figura 7. *Dataset* (6)**



```
auto.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   symboling          205 non-null    int64
 1   normalized-losses  205 non-null    object
 2   make               205 non-null    object
 3   fuel-type          205 non-null    object
 4   aspiration         205 non-null    object
 5   num-of-doors       205 non-null    object
 6   body-style         205 non-null    object
 7   drive-wheels       205 non-null    object
 8   engine-location    205 non-null    object
 9   wheel-base         205 non-null    float64
 10  length             205 non-null    float64
 11  width              205 non-null    float64
 12  height             205 non-null    float64
 13  curb-weight        205 non-null    int64
 14  engine-type        205 non-null    object
 15  num-of-cylinders   205 non-null    object
 16  engine-size        205 non-null    int64
 17  fuel-system        205 non-null    object
 18  bore               205 non-null    object
 19  stroke             205 non-null    object
 20  compression-ratio  205 non-null    float64
 21  horsepower         205 non-null    object
 22  peak-rpm           205 non-null    object
 23  city-mpg           205 non-null    int64
 24  highway-mpg        205 non-null    int64
 25  price              205 non-null    object
dtypes: float64(5), int64(5), object(16)
memory usage: 41.8+ KB
```

Fuente: elaboración propia.

Con eso vemos que hay columnas que son del tipo “*object*” cuando deberían ser del tipo “*integer*” o “*float*” como son los casos: *normalized-losses*, *bore, stroke, horsepower, peak-rpm y price*. Esto se debe al símbolo “?”.

Una alternativa sería reemplazar los valores “?” por valor “*nan*” (es decir, nulo o vacío) y luego trabajar esos valores nulos reemplazándolos con la media o mediana (esto se puede lograr en caso de que sean pocos en relación al total).

Aunque no forma parte del alcance de este curso, veamos cómo se resolvería esta situación:

**Figura 8. *Dataset* (7)**

```
for col in auto.columns:
    auto[col].replace({'?':np.nan},inplace=True)
auto.isnull().sum()
```

```
symboling            0
normalized-losses    41
make                 0
fuel-type            0
aspiration           0
num-of-doors         2
body-style           0
drive-wheels         0
engine-location      0
wheel-base           0
length               0
width                0
height               0
curb-weight          0
engine-type          0
num-of-cylinders     0
engine-size          0
fuel-system          0
bore                 4
stroke               4
compression-ratio    0
horsepower           2
peak-rpm             2
city-mpg             0
highway-mpg          0
price                4
dtype: int64
```

Fuente: elaboración propia.

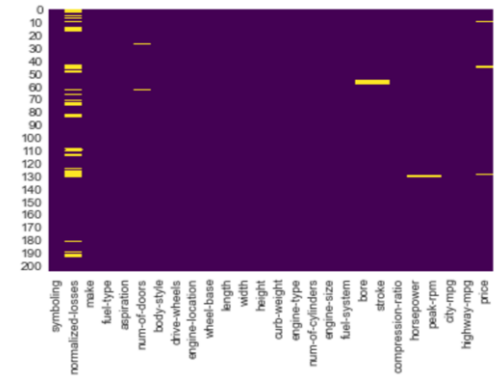
Reemplazamos el signo “?” por valor nulo y, luego, hacemos un recuento. Concluimos que, efectivamente, las columnas mencionadas tenían en alguna fila el signo “?”. Si quisiéramos visualizar la distribución de valores nulos, podemos usar un mapa de calor.

> Los “*heatmaps*” o mapas de calor son una representación gráfica de datos donde los valores se representan mediante colores. . . utilizan una gama de colores que va desde los más cálidos, como el rojo, el naranja y el amarillo, hasta colores como el verde y el azul en el extremo más frío del espectro. (Basurto, 2022, https://bit.ly/3T04aIb)

**Figura 9: Mapa de calor**

```
sns.heatmap(auto.isnull(),cbar=False,cmap='viridis')
```

```
<AxesSubplot:>
```




Fuente: elaboración propia.

Finalmente, podríamos rellenar los espacios vacíos para las variables numéricas con el valor de la media:

**Figura 10: Rellenar los espacios vacíos**

```
num_col = ['normalized-losses', 'bore',  'stroke', 'horsepower', 'peak-rpm','price']
for col in num_col:
    auto[col]=pd.to_numeric(auto[col])
    auto[col].fillna(auto[col].mean(), inplace=True)
auto.head()
```

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | engine-size | fuel-system | bore | stroke | compression-ratio | horsepo |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 122.0 | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | 1 |
| 1 | 3 | 122.0 | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | 1 |
| 2 | 1 | 122.0 | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 | ... | 152 | mpfi | 2.68 | 3.47 | 9.0 | 1 |
| 3 | 2 | 164.0 | audi | gas | std | four | sedan | fwd | front | 99.8 | ... | 109 | mpfi | 3.19 | 3.40 | 10.0 | 1 |
| 4 | 2 | 164.0 | audi | gas | std | four | sedan | 4wd | front | 99.4 | ... | 136 | mpfi | 3.19 | 3.40 | 8.0 | 1 |

5 rows × 26 columns

Fuente: elaboración propia.

## Tema 4. Funciones básicas de matemáticas

Python es muy útil para resolver problemas matemáticos. “Incluye tipos incorporados para manejar números enteros y números de coma flotante, que son adecuados para las matemáticas básicas que pueden aparecer en una aplicación promedio” (Rico Schmidt, s.f., https://bit.ly/3FbgQGz). La biblioteca estándar incluye módulos para necesidades más avanzadas.

Los números de coma flotante incorporados de Python son lo suficientemente precisos para la mayoría de las solicitudes con requerimientos matemáticos, pero **cuando se necesitan representaciones más precisas de valores no enteros, los módulos decimal y *fractions*** son más adecuados.

- **“El módulo decimal** implementa aritmética decimal de coma fija y flotante usando el modelo familiar para la mayoría de las personas” (Rico Schmidt, s.f., https://bit.ly/3Zxzalk). Una instancia de decimal puede representar cualquier número exactamente, redondear hacia arriba o hacia abajo, y aplicar un límite al número de dígitos significativos. Ver documentación adicional en https://bit.ly/428Lri7.
- **El módulo *fractions*** implementa operaciones numéricas para trabajar con números racionales. Ver documentación adicional en: https://bit.ly/3J3xl8T.
- **El módulo *random*** incluye un generador de números pseudoaleatorios de distribución uniforme, así como funciones para simular muchas distribuciones comunes no uniformes. Ver documentación adicional en https://bit.ly/3L5OWiW.
- **El módulo *math*** contiene implementaciones rápidas de funciones matemáticas avanzadas tales como logaritmos y funciones trigonométricas. Ver documentación

adicional en https://bit.ly/3mDsSC1.

- **“El módulo statistics** implementa muchas fórmulas estadísticas comunes [la media, la mediana, el modo, varianza, etc.] para cálculos eficientes usando varios tipos numéricos de Python [int, float, Decimal, y Fracción]” (Rico Schmidt, s.f., https://bit.ly/3ZCHksY). Ver documentación en https://bit.ly/3SVWwyN.

En Python podemos hacer operaciones en una columna de datos. Como ejemplo, **multipliquemos todos los valores de peso por 2**. Un uso más útil podría ser normalizar los datos con la media, área o algún otro valor calculado de nuestros datos.

**Figura 11: Operaciones en una columna de datos**

```python
# Multiplicar todos los valores de peso por 2
surveys_df['weight']*2
```

Fuente: elaboración propia.

Podemos graficar rápida y fácilmente los datos usando Pandas, y aplicar estadísticas descriptivas.

**Figura 12: Crear gráfico**

```python
# Aseguremonos de que las imágenes aparezcan insertadas en iPython Notebook
%matplotlib inline
# Creaemos una gráfica de barras
species_counts.plot(kind='bar');
```

Fuente: elaboración propia.

También podemos realizar conteos de especie por sitio, por ejemplo, ver cuántos animales fueron capturados por sitio:

**Figura 13: Conteos de especie por sitio**

```python
total_count = surveys_df.groupby('plot_id')['record_id'].nunique()
# También grafiquemos eso
total_count.plot(kind='bar');
```

Fuente: elaboración propia.

## Actividad de repaso

**¿Qué módulo incluye un generador de números pseudoaleatorios de distribución uniforme?**

- Módulo decimal.
- Módulo random.
- Módulo fractions.
- Módulo statistics.
- Módulo math.

**Justificación**

# Unidad 2: Gráficos y su interpretación

## Tema 1. Librerías de visualización de Python

Existen diversas librerías de visualización asociadas a Python que se están aplicando en diversas áreas de vanguardia:

- *Machine learning.*
- Cálculo numérico.
- Análisis de datos.
- Aprendizaje automático “es un subconjunto de la inteligencia artificial que permite que permite a una máquina aprender de datos anteriores sin tener que diseñarla explícitamente” (Equipo AEFOL, 2022, https://bit.ly/3ZtyPQT).
- *Deep learning* “es una técnica de aprendizaje automático basada en el modelo de red neuronal: se apilan decenas o incluso cientos de capas de neuronas para

aportar mayor complejidad al establecimiento de reglas" (DataScientest, 2022, https://bit.ly/3Zv5mWJ).

- Inteligencia artificial explicable (se refiere a técnicas en la aplicación de la inteligencia artificial por las cuales el ser humano es capaz de comprender las decisiones y predicciones realizadas por la inteligencia artificial).
- Procesamiento del lenguaje natural.

## Librerías de Python para visualización

Una de las fases del proceso de *machine learning* más importantes es entender el problema que vamos a resolver. Una forma que tenemos de mejorar nuestra comprensión del problema es entender mejor los datos. La visualización de datos nos ayuda a entender mejor tanto los datos como el problema.

Así mismo, la visualización de datos será también muy útil para comprender los resultados y analizar los errores. Aunque hay muchas librerías en Python para la visualización de datos, nos vamos a concentrar en Matplotlib, Seaborn y Bokeh por el momento. (Martínez Heras, 2020, https://bit.ly/3L6JEDN)

Las librerías de Python que vamos a conocer en este tema son gratuitas.

**Matplotlib**

Matplotlib es la librería gráfica de Python estándar y la más conocida. Puedes usar Matplotlib para generar gráficos de calidad necesaria para publicarlas tanto en papel como digitalmente.

Con Matplotlib puedes crear muchos tipos de gráficos: series temporales, histogramas, espectros de potencia, diagramas de barras, diagramas de errores, etc.

[Más adelante, en este módulo, estudiaremos más detenidamente lo que Matplotlib es capaz de realizar]

**Seaborn**

Seaborn es una librería gráfica basada en Matplotlib, especializada en la visualización de datos estadísticos. Se caracteriza por ofrecer un interfaz de alto

nivel para crear gráficos estadísticos visualmente atractivos e informativos.

Seaborn considera la visualización como un aspecto fundamental a la hora de explorar y entender los datos. Se integra muy bien con la librería de manipulación de datos pandas.

**Bokeh**

Bokeh es una librería para visualizar datos de forma interactiva en un navegador web. Con Bokeh podemos crear gráficos versátiles, elegantes e interactivos. Los desarrolladores de Bokeh buscan un buen rendimiento con gran cantidad de datos, incluso con datos que vayan llegando en tiempo real.

. . .

## Librerías de Python para cálculo numérico y análisis de datos

Otra de las fases del proceso de *machine learning* que más tiempo consume es la preparación de datos y el cálculo de atributos relevantes o características (*features*). NumPy, SciPy y Pandas son las librerías de Python ideales para análisis de datos y computación numérica.

Seguramente también nos enfrentaremos a problemas que no requieren el uso de aprendizaje automático sino sólo el análisis de datos. Por supuesto, también podemos usar estas librerías en estos casos.

**NumPy**

NumPy proporciona una estructura de datos universal que posibilita el análisis de datos y el intercambio de datos entre distintos algoritmos. Las estructuras de datos que implementa son vectores multidimensionales y matrices con capacidad para gran cantidad de datos.

Además, esta librería proporciona funciones matemáticas de alto nivel que operan en estas estructuras de datos. Para aprender más, sigue el tutorial de NumPy (en inglés).

**SciPy**

SciPy proporciona rutinas numéricas eficientes fáciles de usar y opera en las

mismas estructuras de datos proporcionadas por NumPy. Por ejemplo, con SciPy puedes realizar: integración numérica, optimización, interpolación, transformadas de Fourier, álgebra lineal, estadística, etc.

. . .

**Pandas**

Pandas es una de las librerías de Python más útiles para los científicos de datos. Las estructuras de datos principales en Pandas son *Series* para datos en una dimensión y *DataFrame* para datos en dos dimensiones.

Estas son las estructuras de datos más usadas en muchos campos tales como finanzas, estadística, ciencias sociales y muchas áreas de ingeniería. Pandas destaca por lo fácil y flexible que hace la manipulación de datos y el análisis de datos.

Para aprender más, puedes mirar la documentación de Pandas (https://pandas.pydata.org/).

**Numba**

Numba es un compilador para Python diseñado especialmente para acelerar las funciones numéricas, generando código máquina optimizado a partir de código Python.

Numba traduce funciones escritas en Python a código máquina optimizado a la hora de ejecutarse; es decir, da la capacidad de compilar el código tan pronto como se ejecuta (es un compilador *Just-in-Time*).

Los algoritmos numéricos compilados con Numba pueden alcanzar velocidades de ejecución tan altas como las de C o FORTRAN.

Si es necesario optimizar la velocidad del código, no tenemos por qué compilar código por separado, ni necesitamos tener el compilador de C/C++ instalado. Basta aplicar uno de los decoradores de Numba a nuestra función de Python y Numba hará el resto.

Para saber más, puedes consultar la documentación de Numba (https://numba.pydata.org/)

## Librerías de Python para *machine learning*

Existen librerías de aprendizaje automático, Scikit-Learn es la más utilizada.

**Scikit-learn**

Scikit-learn es una librería de Python para *machine learning* y análisis de datos. Está basada en NumPy, SciPy y Matplotlib. Las ventajas principales de Scikit-learn son su facilidad de uso y la gran cantidad de técnicas de aprendizaje automático que implementa.

Con Scikit-learn podemos realizar aprendizaje supervisado y no supervisado. Podemos usarlo para resolver problemas tanto de clasificación y como de regresión.

Es muy fácil de usar porque tiene una interfaz simple y muy consistente, fácil de utilizar. Te das cuenta que el interfaz es consistente cuando puedes cambiar de técnica de *machine learning* cambiando solo una línea de código.

Otro punto a favor de scikit-learn es que los valores de los hiper-parámetros tienen unos valores por defecto adecuados para la mayoría de los casos.

**Estas son algunas de las técnicas de aprendizaje automático que podemos usar con Scikit-learn:**

- regresión lineal y polinómica;
- regresión logística;
- máquinas de vectores de soporte;
- árboles de decisión;
- bosques aleatorios (*random forests*);
- agrupamiento (*clustering*);
- modelos basados en instancias;
- clasificadores *bayesianos*;
- reducción de dimensionalidad;
- detección de anomalías;
- etc.

Para aprender más, puedes mirar la documentación de scikit-learn (https://scikit-learn.org/stable/).

---

## Librerías de Python para *deep learning*

Aunque el *deep learning* se engloba dentro del *machine learning*, colocamos las librerías de Python para aprendizaje profundo al mismo nivel, ya que últimamente el *deep learning* es el mayor responsable de la reciente popularidad del *machine learning.*

### TensorFlow

TensorFlow es una librería de Python, desarrollada por Google, para realizar cálculos numéricos mediante diagramas de flujo de datos.

En vez de codificar un programa, codificaremos un grafo. Los nodos de este grafo serán operaciones matemáticas y las aristas representan los tensores (matrices de datos multidimensionales).

Con esta computación basada en grafos, TensorFlow puede usarse para *deep learning* y otras aplicaciones de cálculo científico.

Si te estás preguntando por qué necesitamos diseñar un grafo en vez de un programa, es por la flexibilidad de ejecución de TensorFlow. Por ejemplo, el grafo que representa la red neuronal profunda y sus datos, se podrá ejecutar en una o varias CPU o GPU en un PC, en un servidor o en un móvil.

Para aprender más, puedes visitar la página de TensorFlow: https://www.tensorflow.org y ver allí un Tutorial: https://www.tensorflow.org/tutorials (en inglés).

### Keras

Keras es un interfaz de alto nivel para trabajar con redes neuronales. El interfaz de Keras es mucho más fácil de usar que el de TensorFlow. Esta facilidad de uso es su principal característica.

Con Keras es muy fácil comprobar si nuestras ideas tendrán buenos resultados rápidamente. Keras utiliza otras librerías de *deep learning* (TensorFlow, CNTK o Theano) de forma transparente para hacer el trabajo que le digamos.

Para aprender más, puedes mirar la documentación de Keras: https://keras.io/ (en inglés).

**PyTorch**

PyTorch es una librería de Python, desarrollada por Facebook, que permite el cálculo numérico eficiente en CPU y GPU.

Puedes pensar en PyTorch como una librería que te la las capacidades de NumPy en una GPU. En otras palabras, si tu tarjeta gráfica tiene un procesador gráfico (por ejemplo, una NVIDIA moderna), tu código se puede ejecutar unas ¡10 – 20 veces más rápido!

El aprendizaje profundo (*deep learning*) usa cálculos matriciales y de derivadas masivos y paralelizables en GPU. Por eso, PyTorch también se especializa en *deep learning.*

Para aprender más, puedes mirar la página de PyTorch y su documentación: https://pytorch.org/

---

Librerías de Python para IA explicable

**SHAP**

SHAP es una librería para realizar inteligencia artificial explicable (XAI por sus siglas en inglés *explainable artificial intelligence*). Utiliza cálculos del campo de la teoría de juegos para averiguar qué variables tienen más influencia en las predicciones de las técnicas de *machine learning.*

SHAP permite entender cómo se toman las decisiones en modelos de caja negra (*random forest* o redes neuronales). Puedes obtener explicaciones tanto para predicciones individuales como de forma global. Su API es bastante fácil de usar.

---

Librerías de Python para procesamiento de lenguaje natural

Algunas de las librerías que hemos visto se pueden usar también para algunas de las fases del procesamiento del lenguaje natural. Por ejemplo, Scikit-learn puede usarse para calcular frecuencias normalizadas de los términos que aparecen en documentos.

Las librerías de *deep learning* y Scikit-learn también permiten construir modelos de *machine learning* con datos de texto, una vez estos se hayan convertido a un formato estándar.

En este apartado, vamos a ver las librerías que están principalmente dedicadas al procesamiento del lenguaje natural.

**NLTK: *natural language toolkit***

NLTK es una de las librerías más antiguas en Python para procesamiento de lenguaje natural. Sigue siendo muy útil para tareas de preprocesado de texto tales como la *tokenización*, lematización, exclusión de palabras irrelevantes, etc. NLTK también se usa mucho como herramienta de estudio y enseñanza de procesamiento del lenguaje.

Para aprender más de NLTK, puedes visitar la página: https://www.nltk.org/

**Gensim**

Gensim es una librería para el procesamiento de lenguaje natural creada por Radim Řehůřek. El punto fuerte de Gensim es el modelado de temas. Es decir, puede identificar automáticamente de que tratan un conjunto de documentos.

Además, Gensim es útil para construir o importar representaciones de vectores distribuidas tales como *word2vec*. También podemos usar Gensim para analizar la semejanza entre documentos, lo que es muy útil cuando realizamos búsquedas.

Para aprender más, puedes ver la página: https://radimrehurek.com/gensim/

**spaCy**

SpaCy es la librería de procesamiento natural más rápida que existe. Está diseñada para usarse en aplicaciones reales y extraer información relevante. SpaCy también es muy útil para preparar texto para otras tareas de aprendizaje automático. Por ejemplo, podemos preparar los datos para usarlos con TensorFlow, PyTorch, Scikit-learn, Gensim, etc.

Con SpaCy también vamos a poder construir modelos lingüísticos estadísticos sofisticados para muchos de los problemas de procesamiento de lenguaje natural.

Para saber más, mira la documentación de SpaCy (en inglés).

**Jupyter Notebook**

Jupyter Notebook es una aplicación web para crear documentos que contienen código, ecuaciones, visualizaciones y texto. Puedes usar Jupyter Notebooks para limpiar datos, transformarlos, realizar simulaciones numéricas, modelos estadísticos, visualizaciones de datos, *machine learning* y mucho más.

A efectos prácticos es como una consola interactiva de Python en un navegador que permite la ejecución de código Python, visualización de datos y gráficos, y documentar lo que estés haciendo.

Jupyter no es en realidad una librería de Python. Sin embargo, ya que estamos viendo cuáles son las herramientas que más usa un científico de datos, la lista no estaría completa sin Jupyter.

[Jupyter Notebook es un entorno web que va a facilitarnos mucho el trabajo. Con Jupyter podemos probar nuestras ideas y ver los resultados de forma muy intuitiva, a la vez que lo documentamos].

. . .

**Anaconda**

Anaconda es una distribución de Python para Cálculo numérico, análisis de datos y *machine learning*. Contiene las librerías más usadas por los científicos de datos. Además, hace muy fácil la instalación de otras librerías que puedas necesitar.

Con Anaconda también es posible crear varios entornos de trabajo si estás trabajando en varios proyectos. Esto puede ser útil, por ejemplo, si uno de los proyectos necesita Python 3 y el otro Python 2. O si estás trabajando en un proyecto que necesita unas librerías específicas o que tengan una versión específica.

A no ser que tengas que trabajar con aplicaciones antiguas, lo recomendable es usar la distribución de Anaconda con Python 3. (Martínez Heras, 2020, https://bit.ly/3L6JEDN)

Podemos ver la documentación de Anaconda en su página: https://anaconda.org/.

Resumen

“Hemos visto las mejores librerías de Python para:

- visualización;
- cálculo numérico;
- análisis de datos;
- manipulación de datos;
- *machine learning;*
- *deep learning;*
- inteligencia artificial explicable;
- procesamiento de lenguaje natural” (Martínez Heras, 2020, https://bit.ly/3L6JEDN).

Finalmente, la forma más fácil de instalar todas estas librerías es mediante Anaconda. Anaconda va a instalar muchas de estas librerías y el resto podrás instalarlas manualmente cuando las necesites.

## Actividad de repaso

**¿Cuál de las librerías estudiadas es un compilador para Python diseñado especialmente para acelerar las funciones numéricas, generando código de máquina optimizado a partir de código Python?**

- Numba.
- SciPy.
- NumPy.
- Bokeh.
- Seaborn.
- TensorFlow.

## Tema 2. Matplotlib: consejos generales

Matplotlib es una librería de visualización de datos multiplataforma que se fundamenta en las matrices de NumPy. Se trata de una librería de Python especializada en la creación de gráficos en dos dimensiones.

Permite crear y personalizar los tipos de gráficos más comunes:

- Diagramas de barras.
- Histograma.
- Diagramas de sectores.
- Diagramas de caja y bigotes (*boxplot*).
- Diagramas de violín.
- Diagramas de dispersión o puntos.
- Diagramas de líneas.
- Diagramas de áreas.
- Diagramas de contorno.
- Mapas de color.
- Además de combinaciones entre ellos" (Aprende con Alf, 2020, https://bit.ly/3L7B4EV)

Antes de empezar a hacer visualizaciones y gráficas con la librería Matplotlib es necesario que resolvamos temas generales con algunos consejos que nos ayudarán a sacar el mayor provecho de estos recursos.

### a. Cómo importar Matplotlib

Debemos importar la librería con: import matplotlib.mpl y, además, podemos asignarle una abreviación:

**Figura 14: Importar librería**

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

Fuente: elaboración propia.

Usaremos la interfaz *plt* con mayor frecuencia en los distintos ejemplos y ejercicios.

### b. Estilos de configuración

Con la función plt.style podemos elegir estilos estéticos apropiados para nuestros gráficos. Podemos partir con el estilo clásico.

**Figura 15: Elección de estilos**

```
plt.style.use('classic')
```

Fuente: elaboración propia.

A medida que necesitemos un cambio de estilo, podemos actualizarlo.

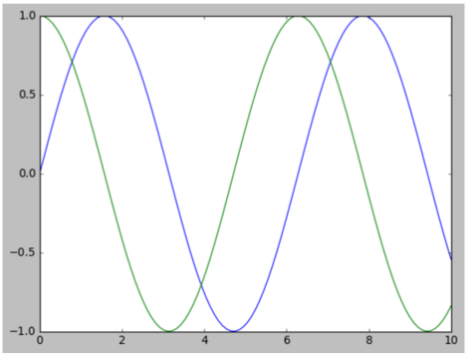
### c. Visualización de los gráficos

Para poder visualizar un gráfico, debemos usar la función plt.show(). plt.show(). que nos permiten iniciar un ciclo de eventos. Busca todos los objetos de figuran actualmente activos y abre una o más ventanas interactivas que muestran su(s) figura(s). Entonces, por ejemplo, podríamos tener un archivo llamado myplot.py que contenga lo siguiente:

**Figura 16. Visualización de gráficos**

```
# ------- file: myplot.py ------
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.show()
```




Fuente: elaboración propia.

### d. Gráfico de línea simple

La visualización de una sola función y = f(x) es probablemente el gráfico más simple de todos. Primero, configuraremos el cuaderno para trazar e importar las funciones que usaremos:

**Figura 17: Gráfico de línea simple**

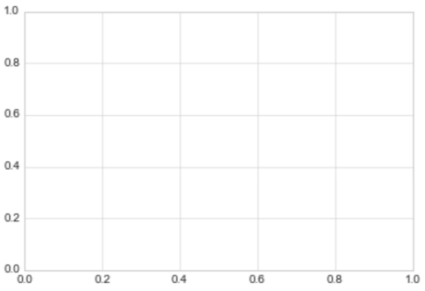
```
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
```

Fuente: elaboración propia.

Para todos los diagramas de Matplotlib comenzaremos creando una figura y sus ejes. En su forma más simple se pueden realizar de la siguiente manera:

**Figura 18: Creación de figura**

```
fig = plt.figure()
ax = plt.axes()
```




Fuente: elaboración propia.

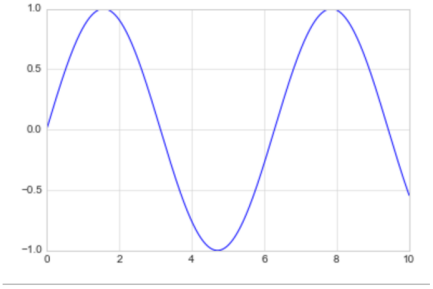
**En Matplotlib la figura (una instancia de la clase plt.Figure) se puede considerar como un contenedor único que posee todos los objetos que representan ejes, gráficos, textos y etiquetas.**

Los ejes (una instancia de la clase plt.Axes) son los que vemos arriba: un **cuadro delimitador con marcas y etiquetas** que, eventualmente, contendrá los elementos que conformarán nuestra visualización.

Una vez creado el eje, podemos usar la función ax.plot para trazar algunos datos. Comencemos con una sinusoide simple:

**Figura 19: Sinusoide**

```
fig = plt.figure()
ax = plt.axes()
x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
```

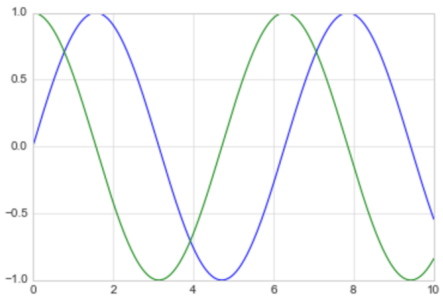



Fuente: elaboración propia.

Si queremos crear una sola figura con varias líneas, podemos llamar a la función de trazado varias veces:

**Figura 20: Figura con varias líneas**

```
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x));
```




Fuente: elaboración propia.

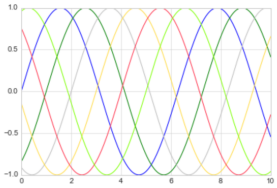
### e. Cómo ajustar el gráfico: colores y estilos de línea

El primer ajuste que quizás desees realizar en un gráfico es controlar los colores y estilos de las líneas. La función plt.plot() toma argumentos adicionales que se pueden usar para especificarlos.

Para ajustar el color, se puede utilizar la palabra clave “color” que acepta un argumento de cadena que representa a cualquiera de ellos. El color se puede especificar de varias maneras:

**Figura 21: Especificar color**

```
plt.plot(x, np.sin(x - 0), color='blue') # especificar el color por el nombre
plt.plot(x, np.sin(x - 1), color='g') # código corto del color (rgbcmyk)
plt.plot(x, np.sin(x - 2), color='0.75') # grado de grises entre 0 y 1
plt.plot(x, np.sin(x - 3), color='#FFDD44') # código HEX (RRGGBB desde 00 a FF)
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # tupla RGB, valores 0 y 1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # todos los nombres de colores HTML soportados
```

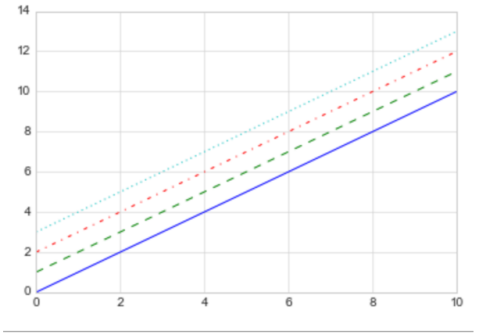


Fuente: elaboración propia.

De la misma manera, se puede definir el estilo de línea con la palabra clave “estilo de línea”:

**Figura 22. Estilo de línea**

```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');
```

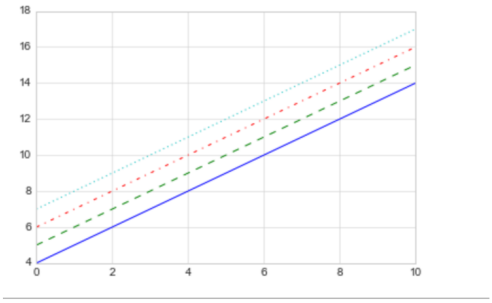



Fuente: elaboración propia.

Para abreviarlo, se pueden usar los siguientes códigos:

**Figura 23: Abreviaciones**

```
plt.plot(x, x + 4, linestyle='-') # línea sólida
plt.plot(x, x + 5, linestyle='--') # línea discontinua
plt.plot(x, x + 6, linestyle='-.') # línea guión punto
plt.plot(x, x + 7, linestyle=':'); # línea punteada
```




Fuente: elaboración propia.

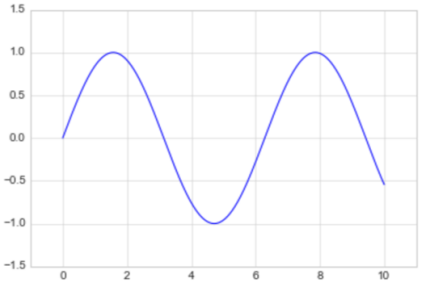
### f. Cómo ajustar el gráfico: límite de los ejes

Matplotlib hace un trabajo bastante bueno al elegir los límites de ejes predeterminados para su gráfico, pero no siempre escoge la mejor versión. Siempre es bueno revisar los límites de los ejes.

**En caso de necesitar ajustes, podemos usar las funciones plt.xlim() y plt.ylim():**

**Figura 24: Ajustar el gráfico**

```
plt.plot(x, np.sin(x))
plt.xlim(-1, 11)
plt.ylim(-1.5, 1.5);
```

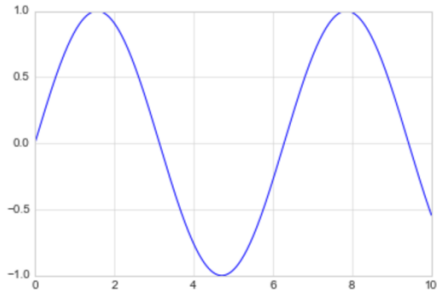



Fuente: elaboración propia.

Se ve mucho mejor si lo comparamos con el ejemplo inicial:

### Figura 25: Gráfico ajustado

```
fig = plt.figure()
ax = plt.axes()
x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x));
```




Fuente: elaboración propia.

Podríamos querer invertir el orden de los ejes. Esto se logra al invertir el orden de los argumentos de la función anterior:

### Figura 26: Invertir el orden de los ejes

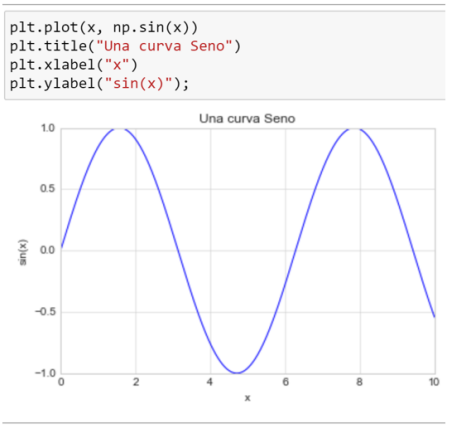
```
plt.plot(x, np.sin(x))
plt.xlim(10, 0)
plt.ylim(1.2, -1.2);
```


Fuente: elaboración propia.

### g. Etiquetar los gráficos

Finalmente, revisaremos el etiquetado de los gráficos: sus títulos, etiquetas de eje y leyendas simples. Los títulos y las etiquetas de los ejes son las etiquetas más simples y existen métodos que se pueden usar para establecerlas rápidamente.

**Figura 27: Revisar etiquetas y títulos**



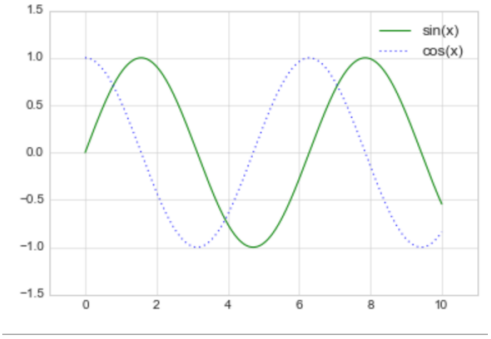

Fuente: elaboración propia.

Adicionalmente, podemos ajustar la posición, el tamaño y el estilo de estas etiquetas mediante argumentos opcionales para la función.

Cuando se muestran varias líneas dentro de un solo eje, puede ser útil crear una leyenda de trazado que etiquete cada tipo de línea. Matplotlib tiene una forma integrada de crear rápidamente una leyenda de este tipo. Se hace a través del **método plt.legend().**

**Figura 28: Método plt.legend()**

```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.xlim(-1, 11)
plt.ylim(-1.5, 1.5)
plt.legend();
```




Fuente: elaboración propia.

A continuación, veremos un video que indica buenas prácticas al momento de utilizar Matplotlib.

### Video 2: Introducción a Matplotlib en Python

Fuente: The PyCoach en Español (2 de abril de 2021). Introducción a Matplotlib en Python | Como hacer gráficos con Python | Curso completo de Matplotlib [archivo de video]. YouTube. https://bit.ly/3yqQQmV.

## Actividad de repaso

### ¿Para qué se utiliza la función ax.plot en Matplotlib?

- Para ajustar los límites de los ejes.
- Para crear una leyenda.
- Para trazar algunos datos.
- Para definir el estilo de línea.

**Justificación**

## Tema 3. Tipos de gráficos

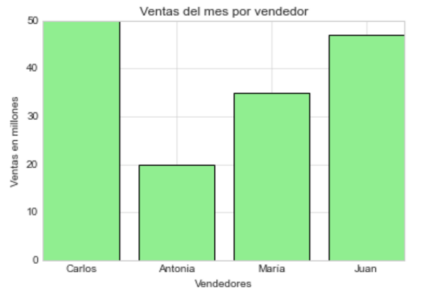
### a. Gráfico de barras discreto

Resume la frecuencia absoluta o relativa en datos numéricos discretos.

## Figura 29. Gráfico de barras discreto

```
import matplotlib.pyplot as plt

eje_x = ['Carlos', 'Antonia', 'María', 'Juan']
eje_y = [50,20,35,47]
plt.bar(eje_x, eje_y, color="lightgreen")
plt.ylabel('Ventas en millones')
plt.xlabel('Vendedores')
plt.title('Ventas del mes por vendedor')
plt.show()
```



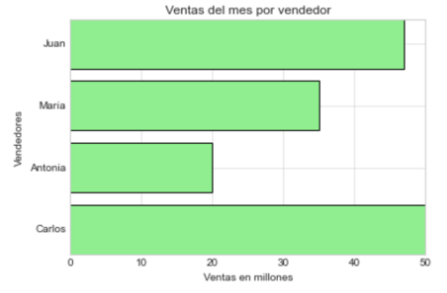

Fuente: elaboración propia.

En el ejemplo mostramos la función plt.show() en un gráfico de barras que resume la frecuencia absoluta de ventas del mes para cada vendedor (Carlos, Antonia, María y Juan). Este gráfico se podría invertir para visualizarlo horizontalmente. Para eso cambiamos “plt.bar” por “plt.hbar” y ajustamos las etiquetas de cada eje:

## Figura 30. Gráfico invertido

```
import matplotlib.pyplot as plt

eje_x = ['Carlos', 'Antonia', 'María', 'Juan']
eje_y = [50,20,35,47]
plt.barh(eje_x, eje_y, color="lightgreen")
plt.xlabel('Ventas en millones')
plt.ylabel('Vendedores')
plt.title('Ventas del mes por vendedor')
plt.show()
```




Fuente: elaboración propia.

Existe la opción de agrupar y apilar las barras. Por ejemplo, las barras agrupadas sirven para comparar categorías a lo largo del tiempo.

## Figura 31. Barras agrupadas

```
import numpy as np
import matplotlib.pyplot as plt

serie_1 = [406, 387, 442, 457, 485]
serie_2 = [421, 453, 435, 478, 512]

numero_de_grupos = len(serie_1)
indice_barras = np.arange(numero_de_grupos)
ancho_barras =0.35

plt.bar(indice_barras, serie_1, width=ancho_barras, label='Hombres')
plt.bar(indice_barras + ancho_barras, serie_2, width=ancho_barras, label='Mujeres')
plt.legend(loc='best')
## Se colocan los indicadores en el eje x
plt.xticks(indice_barras + ancho_barras, ('2017', '2018', '2019', '2020','2021'))

plt.ylabel('Numero de habitantes')
plt.xlabel('Año')
plt.title('Numero de habitantes por genero')

plt.show()
```


Fuente: elaboración propia.

En el ejemplo, podemos identificar la evolución que ha tenido el número de habitantes en un lugar específico separado por género: hombres y mujeres.

Los gráficos de barras apiladas se usan para mostrar, por ejemplo, la distribución de categorías.

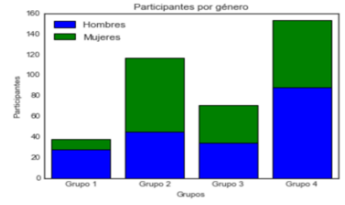
**Figura 32. Gráfico de barras apiladas**

```
import numpy as np
import matplotlib.pyplot as plt
grupos = ['Grupo 1', 'Grupo 2', 'Grupo 3', 'Grupo 4']
hombres = [28,45,34,88]
mujeres = [10,72,37,65]
indice = np.arange(len(grupos))

## Se crean las primeras barras
plt.bar(indice, hombres, label='Hombres')
## Se crean las segundas barras y se apilan sobre las primeras
plt.bar(indice, mujeres, label='Mujeres',  bottom=hombres)

plt.xticks(indice, grupos)
plt.ylabel("Participantes")
plt.xlabel("Grupos")
plt.title('Participantes por género')
plt.legend(loc = 2)

plt.show()
```




Fuente: elaboración propia.

En este ejemplo podemos ver la proporción entre hombres y mujeres para cada grupo: grupo 1, grupo 2, grupo 3 y grupo 4.

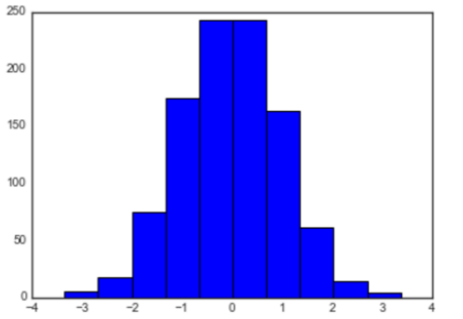
### b. Histograma

Este gráfico ilustra la frecuencia absoluta o relativa en datos numéricos continuos. Un histograma simple puede ser un excelente primer paso para comprender un conjunto de

datos.

**Figura 33. Histograma**

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
data = np.random.randn(1000)
plt.hist(data);
```

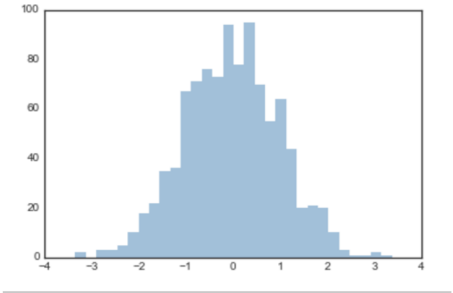



Fuente: elaboración propia.

Al igual que los otros tipos de gráficos, la **función hist()** tiene muchas opciones para ajustar el cálculo y la visualización. Veamos una versión del histograma con más atributos:

**Figura 34. Histograma con más atributos**

```
plt.hist(data, bins=30, alpha=0.5,
histtype='stepfilled', color='steelblue',
edgecolor='none');
```

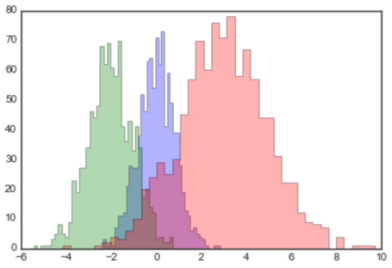



Fuente: elaboración propia.

El atributo “alpha” indica el nivel de transparencia para las barras del histograma. Mientras más cerca del cero (0) se encuentre, más transparente será. Esta visualización permite comparar distintas distribuciones de variables en un *dataset*. Se puede usar una combinación de histtype='stepfilled' junto con algo de transparencia alfa (“alpha”).

**Figura 35. Uso de histtype='stepfilled'**

```
x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)
kwargs = dict(histtype='stepfilled', alpha=0.3, bins=40)
plt.hist(x1, **kwargs)
plt.hist(x2, **kwargs)
plt.hist(x3, **kwargs);
```



Fuente: elaboración propia.

### c. Diagrama de caja

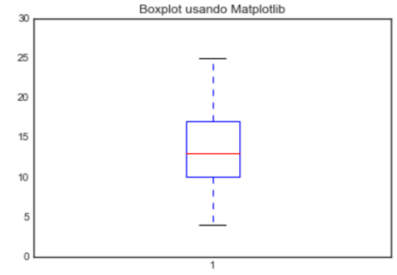
También conocido como *boxplot* o “caja y bigotes”. Refleja la dispersión de los datos a partir de los cuartiles. Un cuartil corresponde a la ubicación del dato según ciertas divisiones. **Los cuartiles son divisiones de los datos en cuatro partes:** el primer cuartil (Q1) representa hasta un 25 %. El segundo cuartil (Q2) corresponde al 50 % (igual a la mediana). El tercer cuartil (Q3) es la ubicación del dato en la posición del 75 %. También podemos encontrar quintiles al dividir en 5 grupos, deciles al dividir en 10 grupos y percentiles al dividir en 100.

**Figura 36. Diagrama de caja**

```
import matplotlib.pyplot as plt

x=[4,5,6,8,9,10,10,11,11,12,13,14,15,15,15,17,18,19,22,23,25]

plt.boxplot(x)
plt.title("Boxplot usando Matplotlib")
plt.ylim(0, 30)
plt.show()
```



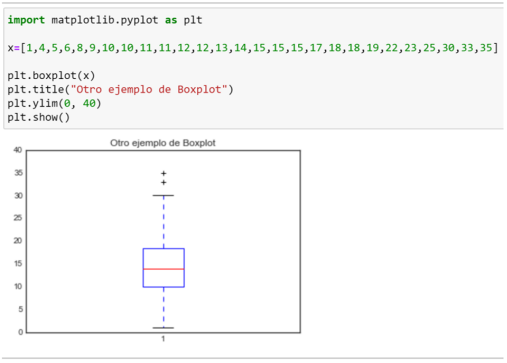

Fuente: elaboración propia.

En el ejemplo, retorna un *boxplot* a partir de los datos dados. Descomponemos el resultado:

- La caja se extiende desde el primer cuartil (Q1) hasta el tercer cuartil (Q3).
- La línea horizontal dentro de la caja representa la mediana de los datos.
- Los bigotes en la gráfica de caja se extienden desde el valor Q3 hasta el valor máximo de los datos y desde el valor mínimo de los datos hasta el Q1 de los datos.
- El valor mínimo de los datos está determinado por el valor de Q1-1.5*(Q3-Q1) mientras que el valor máximo de los datos está determinado por la fórmula Q3+1.5*(Q3-Q1) (Joshi, 2021, https://bit.ly/41YeeWg).

**Veamos otro ejemplo:**

**Figura 37. Otro ejemplo de *boxplot***




Fuente: elaboración propia.

Retorna el diagrama de caja de los datos dados x. Podemos observar dos valores atípicos en la parte superior del diagrama, representados por cruces en el diagrama. Un punto de datos se representa como un **atípico** si su valor es menor que Q1-1.5*(Q3-Q1) o mayor que Q3+ 1.5*(Q3-Q1).

Un **valor atípico** es una observación extrañamente grande o pequeña. Los valores atípicos pueden tener un efecto desproporcionado en los resultados estadísticos (por ejemplo, en la media), lo que puede conducir a interpretaciones engañosas. Por eso es importante identificarlos cuando hacemos un análisis de datos exploratorio para tomar decisiones que nos permitan mejorar los *datasets*.

También podemos usar el gráfico *boxplot* para comparar resultados de distintos grupos:

**Figura 38. Comparación de grupos con *boxplot***

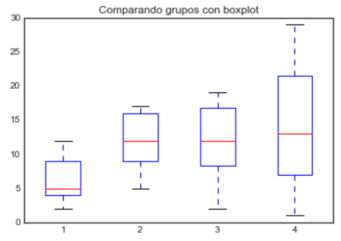
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(100)

data_a =np.random.randint(2,15, size=15)
data_b =np.random.randint(5,18, size=20)
data_c =np.random.randint(2,20, size=30)
data_d =np.random.randint(1,30, size=40)

data_2d=[data_a,data_b,data_c,data_d]

plt.boxplot(data_2d)
plt.title("Comparando grupos con boxplot")
plt.show()
```




Fuente: elaboración propia.

## Actividad de repaso

### ¿Cuál de las siguientes afirmaciones es correcta?

Los valores atípicos deben ignorarse en los resultados estadísticos.

Los valores atípicos pueden tener un efecto desproporcionado en los resultados estadísticos.

**Justificación**

## Tema 4. Interpretación de datos y gráficos

## Análisis desde los gráficos

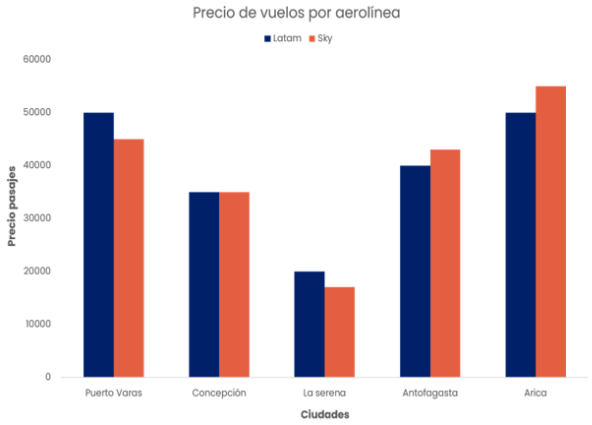
El análisis de datos desde los gráficos es una pieza fundamental en el análisis exploratorio, ya que no se trata solamente de graficar, sino que lo importante es sacar buenas conclusiones que permitan apoyar la toma de decisiones.

### a. Gráfico de barras discreto

Para el caso del gráfico de barras discreto, veamos el siguiente ejemplo: la siguiente

figura expone los costos de viajar en dos aerolíneas diferentes, desde un mismo aeropuerto, hacia ciudades de Chile.

**Figura 39. Gráfico de precios de vuelo por aerolínea**




Fuente: elaboración propia.

**¿Cómo se interpretan estos datos? ¿Qué conclusiones podemos obtener luego de observar el gráfico?** Las barras nos permiten comparar rápidamente cantidades o sumas y el valor que aporta este gráfico es la comparación. Si logramos mostrar de forma fácil una comparación a través de las barras (independiente que estas sean verticales u horizontales), entonces podremos comprender mejor algún fenómeno.

Podemos usar la información declarada para responder preguntas de negocio simples:

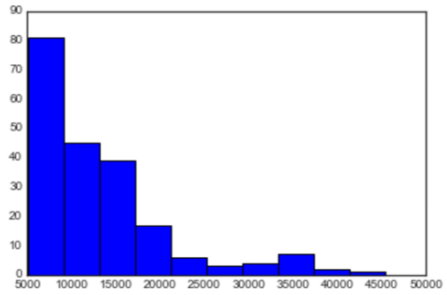
- **¿Qué ciudad tiene el mismo costo de viaje independiente de la aerolínea? Respuesta: Concepción.** Se puede observar que ambas barras son iguales.
- **Para viajar a Puerto Varas ¿qué aerolínea conviene en términos de costo? Respuesta:** SKY.

### b. Histograma

Como vimos, los histogramas miden un hecho mediante una distribución de los datos. Se deben elaborar con variables medibles tales como peso, temperatura, tiempo, etc. Es decir, datos continuos. Volvamos al ejemplo del *dataset* de autos:

**Figura 40. Histograma de precio de automóviles**

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
plt.hist(auto["price"]);
```

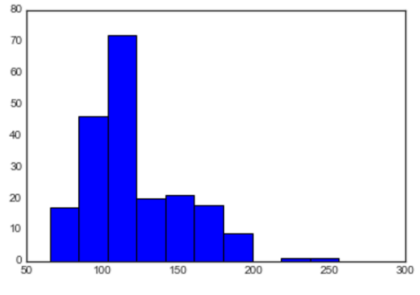



Fuente: elaboración propia.

Al generar un histograma del precio de los autos se observa una fuerte concentración de los datos en vehículos de menor precio (probablemente, más del 80 % en esta categoría) y solo unos pocos que superan los 25 000 dólares.

**Figura 41. Gráfico de la variable *normalized-losses***

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')
plt.hist(auto["normalized-losses"]);
```



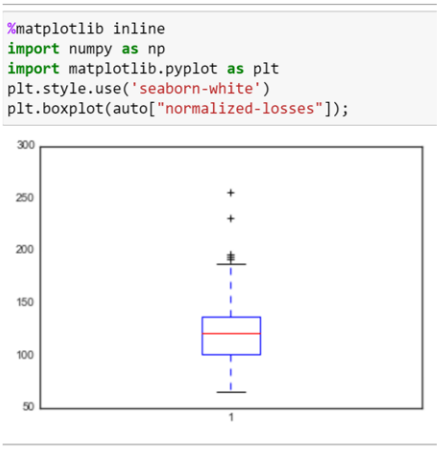

Fuente: elaboración propia.

En el caso de la **variable *normalized-losses***, se observa una distribución algo más normalizada, aunque con valor atípico.

### c. Diagrama de caja

Finalmente, los diagramas de caja o *boxplot* nos permiten **identificar valores atípicos** y comparar distribuciones, además de conocer de manera simple cómo se distribuye el 50 % de los valores centrales.

**Figura 42. *Boxplot* del precio de automóviles**




Fuente: elaboración propia.

## Cierre

En este módulo hemos estudiado los tipos de datos y cómo analizarlos, comenzando con el análisis exploratorio de datos. Para conocer más acerca de este tema, les dejamos una publicación que amplía estos contenidos:

> Fuente: Sotaquirá, M. (2021). ¿Cómo hacer el Análisis Exploratorio de Datos? - Guía paso a paso. Codificando Bits. https://bit.ly/41VG3yq.

Hemos aprendido diversas herramientas para generar gráficos con Python, sus librerías y cómo abordar su interpretación.

En el siguiente enlace les dejamos una simulación de un mercado económico en el que hay una población de actores, cada uno de los cuales tiene un nivel de riqueza diferente. Este caso de ejemplo se encuentra en inglés, pero es muy interesante porque aplica diversas herramientas estudiadas en este módulo: https://bit.ly/3SXz0kU.

**¡Felicidades! Hemos llegado al final del recorrido de los módulos. Ahora es momento de poner todos los conocimientos adquiridos en práctica.**

# Video de habilidades

**¿Cómo podemos importar la librería Matplotlib en Python?**

Con la función plt.style()

Con la función plt.legend()

Con la función import matplotlib.mpl

Justificación

## ¿Cuál es la función que nos permite iniciar un ciclo de eventos para visualizar los gráficos?

plt.xlim()

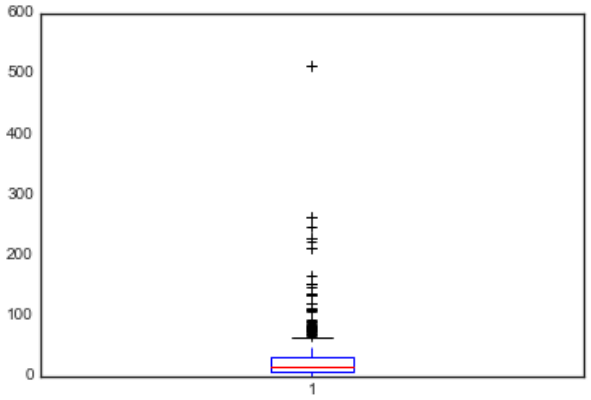
plt.show()

plt.style()

Justificación

¿Qué representa la línea horizontal dentro de la caja en el siguiente diagrama?

**Figura 43: actividad**




Fuente: elaboración propia.

La media de los datos.

La mediana de los datos.

El valor máximo de los datos.

Justificación

**¿Qué es un valor atípico en un diagrama de caja?**

Una observación extrañamente grande o pequeña.

Un valor que está dentro del rango del 50% de los datos.

Un valor que está en el rango del 25% de los datos.

Justificación

**¿Qué efecto pueden tener los valores atípicos en los resultados estadísticos?**

Ningún efecto.

Solo afectan a los resultados de la mediana.

Pueden conducir a interpretaciones engañosas.

Justificación

# Glosario

# Referencias

**Aprende con Alf** (2020). *La librería Matplotlib*. Aprende con ALF. https://aprendeconalf.es/docencia/python/manual/matplotlib/.

**Basurto, K.** (17 de febrero de 2022). Mapas de Calor: Qué son y por qué utilizarlos. *Hiberus blog*. https://www.hiberus.com/crecemos-contigo/mapas-de-calor-que-son-y-por-que-utilizarlos/.

**DataScientest** (19 de abril de 2022). *Deep Learning o Aprendizaje profundo: ¿qué es?*

DataScientest. https://datascientest.com/es/deep-learning-definicion.

**Equipo AEFOL** (1 de agosto de 2022). *¿Cuál es la diferencia entre la IA y el aprendizaje automático?* ELearning actual. https://elearningactual.com/cual-es-la-diferencia-entre-la-ia-y-el-aprendizaje-automatico/.

**Joshi, S.** (2021). *Matplotlib Boxplot Python.* DelffStack. https://www.delftstack.com/es/howto/matplotlib/matplotlib-boxplot-python/.

**Real Academia Española** (2022). *Porcentaje*. Real Academia Española. https://dle.rae.es/porcentaje?m=form.

**Real Academia Española** (2022). *Razón*. Real Academia Española. https://dle.rae.es/raz%C3%B3n?m=form.

**Rico Schmidt, E.** (s.f.). *Decimal — Matemáticas de coma fija y flotante.* Ernesto Rico Schmidt. https://rico-schmidt.name/pymotw-3/decimal/index.html.

**Rico Schmidt, E.** (s.f.). *Statistics — Cálculos estadísticos*. Ernesto Rico Schmidt. https://rico-schmidt.name/pymotw-3/statistics/index.html.

**Rico Schmidt, E.** (s.f.). *Matemáticas*. Ernesto Rico Schmidt. https://rico-schmidt.name/pymotw-3/numeric.html.

**Sensagent Corporation** (2013). *Montículo (informática)*. Sensagent Corporation. http://diccionario.sensagent.com/Mont%c3%adculo%20(inform%c3%a1tica)/es-es/.

**Rodríguez, M., Espinoza, S., Huerta, M. y Tapia Silva, A.** (2020). *Análisis exploratorio de datos en SPSS.* Ediciones UCM.

**The PyCoach en Español** (2 de abril de 2021). *Introducción a Matplotlib en Python* |

*Como hacer gráficos con Python | Curso completo de Matplotlib* [archivo de video]. YouTube. https://bit.ly/3yqQQmV.