Clase 6: Visualización de información

FM849 - Programación Científica para Proyectos de Inteligencia Artificial (IA)

7 de enero de 2026

Visualización de información

- La información puede almacenarse en diferentes formatos: **grafos**, **árboles**, **tablas**, **diccionarios**, entre otros.
- En este curso, nos enfocaremos en datos tabulares (tablas). Este es el formato más simple y utilizado.
- En la clase anterior, vimos cómo abrir un conjunto de datos tabular usando `pandas`. Ahora, la idea es visualizar estos datos.

Pero antes de eso, ¿por qué nos gustaria visualizar los datos?

- Como vimos en la clase de estadística, los conjuntos de datos pueden ser bastante complejos para analizar fila por fila.
- Las técnicas de visualización nos permiten encontrar patrones o tendencias en los datos.
- La visualización de información nos permite responder a preguntas asociadas a los datos.

matplotlib vs seaborn

En esta clase, trabajaremos con dos paquetes para visualizar datos.

- `matplotlib`: librería de uso general para construir todo tipo de gráficos.
  - Es muy flexible, pero requiere más código.
  - Recordar la importación: `import matplotlib.pyplot as plt`.
  - Las funciones de `matplotlib` reciben los vectores de datos a graficar como argumentos.
- `seaborn`: librería orientada al análisis de datos.
  - Está pensada para trabajar directamente con *DataFrames*.
  - Recordar la importación: `import seaborn as sns`.

matplotlib vs seaborn

En matplotlib, los gráficos se construyen indicando:

- Directamente los **datos a graficar** (arreglos, listas, u objetos de tipo pd.Series).
- No existe un argumento data que agrupe los datos.
- Los argumentos (x, y, etc.) corresponden a los **arreglos de valores** y no a los nombres de las columnas.

En seaborn, los gráficos se construyen indicando:

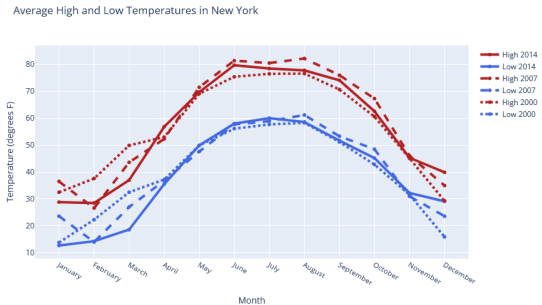
- data: el *DataFrame* que contiene los datos.
- El resto de los argumentos (x, y, hue, etc.) son los **nombres de las columnas** del *DataFrame* que se quieren graficar.

Esto permite crear gráficos de forma simple, clara y eficiente, sin necesidad de extraer manualmente los datos como ocurre habitualmente en matplotlib.

Gráficos de línea

- Los ejes x e y representan los valores que adoptan las variables analizadas.
- Se usan para analizar tendencias temporales y comparación de la evolución de variables.
- Las marcas son puntos que se conectan por líneas.
- La posición vertical expresa un valor cuantitativo, mientras la posición horizontal contiene las llaves ordenadas.

En la figura de la derecha, se indica la evolución de la temperatura en un año.



Ejemplo de gráfico de línea (matplotlib)

Con `plt.plot`:

```
>>> import matplotlib.pyplot as plt
>>> x = [0, 1, 2, 3, 4]
>>> y = [1, 3, 2, 5, 4]
>>> # (0, 1), (1, 3), ... (4, 4)
>>> plt.plot(x, y)
>>> plt.ylabel("y")
>>> plt.show()
```

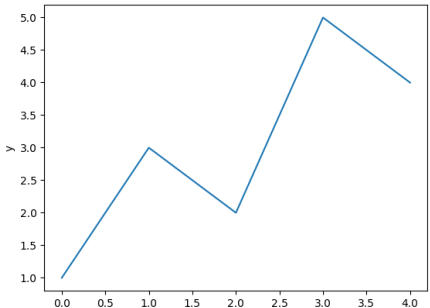


Figura 1: Gráfico de línea en `matplotlib`. Muestra los valores de la lista x (eje x) y lista y (eje y).

Más ejemplos los pueden revisar en este enlace.

Ejemplo de gráfico de línea (seaborn)

Con `sns.lineplot`:

```
>>> import seaborn as sns
>>> # DataFrame de vuelos
>>> df = sns.load_dataset("flights")
>>> sns.lineplot(data=df,
...              x="year",
...              y="passengers")
>>> plt.show()
```

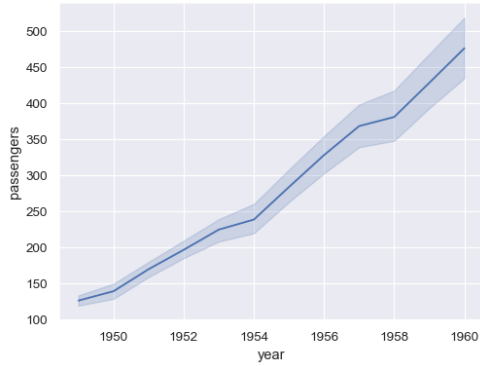


Figura 2: Gráfico de línea que muestra el número de vuelos (eje y) según el año (eje x). El área sombreada representa un intervalo de confianza.

Más ejemplos los pueden revisar en 🔗 este enlace.

Gráfico de dispersión

- El gráfico de dispersión muestra la relación entre dos variables numéricas.
- Permite detectar correlaciones, identificar patrones, clústeres y encontrar *outliers*.
- Los canales son las posiciones: horizontal y vertical. Los ejes x e y representan los valores que adoptan las variables analizadas.

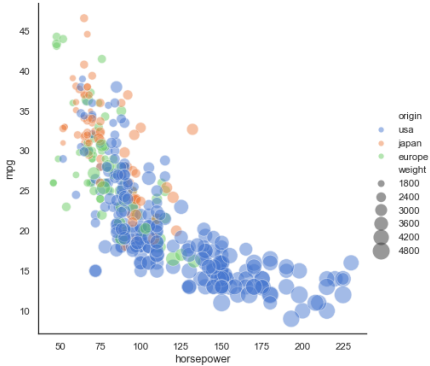


Figura 3: Gráfico de dispersión entre millas por galón (`MPG`, eje y) y caballos de fuerza (`horsepower`, eje x) en autos.

Ejemplo de gráfico de dispersión (matplotlib)

Con `plt.scatter`:

```
>>> import matplotlib.pyplot as plt
>>> x = [1, 2, 3, 4, 5]
>>> y = [2, 1, 3, 5, 4]
>>> # (1, 2), (2, 1), ... , (5, 4)
>>> plt.scatter(x, y)
>>> plt.ylabel("y")
>>> plt.xlabel("x")
>>> plt.show()
```

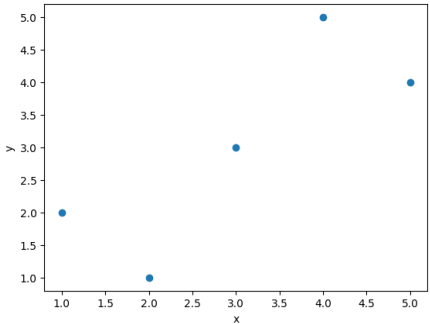


Figura 4: Gráfico de dispersión en `matplotlib`. Muestra los valores de la lista x (eje x) y lista y (eje y).

Más ejemplos los pueden revisar en 🔗 este enlace.

Ejemplo de gráfico de dispersión (seaborn)

Con `sns.scatterplot`:

```
>>> import seaborn as sns
>>> # DataFrame de propinas
>>> df = sns.load_dataset("tips")
>>> sns.scatterplot(data=df,
...                 x="total_bill",
...                 y="tip")
>>> plt.show()
```

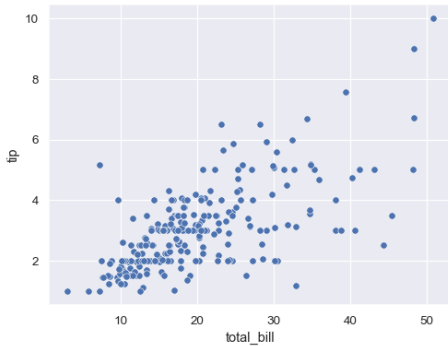


Figura 5: Gráfico de dispersión entre pago total (`total bill`, eje x) y propina (`tip`, eje y).

Más ejemplos los pueden revisar en este enlace.

Gráfico de barras

- Requiere 1 atributo categórico y 1 cuantitativo.
- Los canales que codifican información incluyen: el largo de la barra para expresar un valor cuantitativo y una separación en el espacio para representar otra categoría.
- Se puede usar para comparar categorías y encontrar casos extremos.

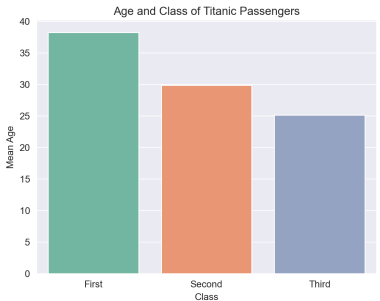


Figura 6: Edad promedio según la clase en el barco Titanic.

Ejemplo de gráfico de barras (matplotlib)

Con `plt.bar`:

```
>>> import matplotlib.pyplot as plt
>>> categorias = ["A", "B", "C", "D"]
>>> valores = [23, 45, 12, 30]
>>> plt.bar(categorias, valores)
>>> plt.ylabel("Valor")
>>> plt.xlabel("Categoría")
>>> plt.show()
```

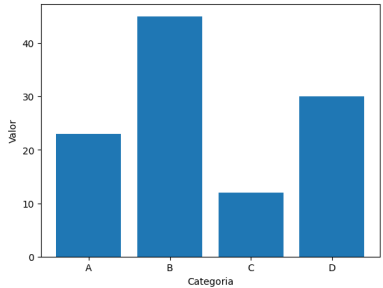


Figura 7: Gráfico de barras generado en `matplotlib`. Muestra valores (eje y) para cada categoria (eje x).

Más ejemplos los pueden revisar en 🔗 este enlace.

Ejemplo de gráfico de barras (seaborn)

Con `sns.barplot`:

```
>>> import seaborn as sns
>>> # DataFrame de pingüinos
>>> df = sns.load_dataset("penguins")
>>> sns.barplot(df,
                x="island",
                y="body_mass_g")
>>> plt.show()
```

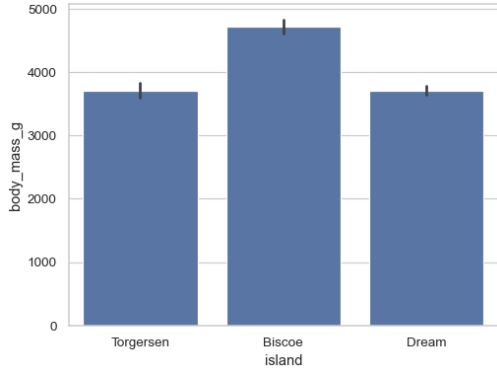


Figura 8: Gráfico de barras mostrando el peso promedio de pingüinos (variable numérica en eje y) en islas (variable categórica).

Más ejemplos los pueden revisar en este enlace.

Histograma

- Un histograma representa una variable cuantitativa agrupando los datos en intervalos.
- Cada barra corresponde a un rango de valores del eje horizontal.
- Permite analizar la forma de la distribución, su asimetría y concentración.
- El eje *y* muestra la frecuencia (número de observaciones) en cada intervalo.

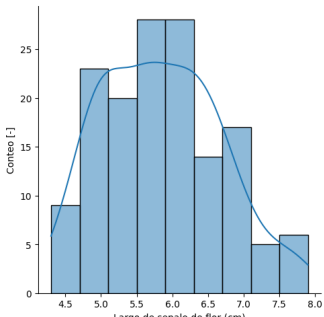


Figura 9: Histograma que muestra la distribución del largo del sépalo en el conjunto de datos Iris.

Ejemplo de histograma (matplotlib)

Con `plt.hist`:

```
>>> import matplotlib.pyplot as plt
>>> temperatura = [2, 33, 24, 25, 12, 20, 0, 23]
>>> plt.hist(temperatura)
>>> plt.xlabel("Temperatura [°C]")
>>> plt.ylabel("Conteo [-]")
>>> plt.show()
```

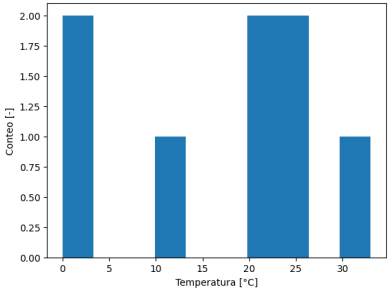


Figura 10: Histograma en `matplotlib`. Muestra la frecuencia (eje y) de veces que ocurren valores de temperatura (eje x).

Más ejemplos los pueden revisar en este enlace.

Ejemplo de histograma (seaborn)

Con `sns.histplot`:

```
>>> import seaborn as sns
>>> # DataFrame de pingüinos
>>> df = sns.load_dataset("penguins")
>>> sns.histplot(data=df,
...              x="flipper_length_mm")
>>> plt.show()
```

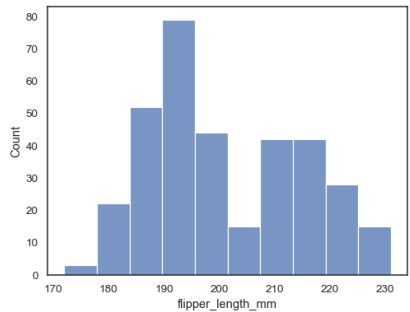


Figura 11: Histograma que muestra la frecuencia (eje y) de los distintos largos de aleta (`flipper_length_mm`) observados (eje x).

Más ejemplos los pueden revisar en 🔗 este enlace.

Boxplot

- El *boxplot* resume una distribución considerando 5 valores obtenidos de la serie de datos: mínimo, cuartil 1 (Q_1), cuartil 2 (Q_2: mediana), cuartil 3 (Q_3) y máximo.
- Se usa para comparar las distribuciones entre columnas de un *DataFrame* en función de sus cuartiles y *outliers*.
- Requiere un atributo categórico (eje x) y un atributo cuantitativo (eje y).
- El principal canal de codificación es el largo de la caja (rango intercuartil) y los puntos fuera de los límites (asociados a *outliers*).

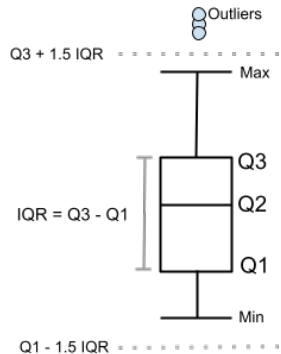


Figura 12: Componentes de *boxplot*. IQR es el rango intercuartil.

Ejemplo de *boxplot* en `matplotlib`

Con `plt.boxplot`:

```
>>> import matplotlib.pyplot as plt
>>> temperatura = [2, 33, 24, 25, 12] + \
>>> ...            [20, 0, 23, 50]
>>> plt.boxplot(temperatura)
>>> plt.ylabel("Temperatura")
>>> plt.show()
```

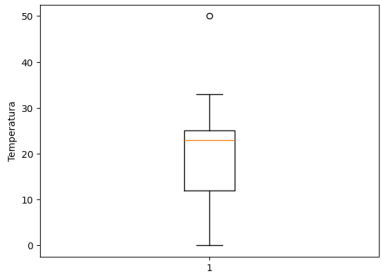


Figura 13: *Boxplot* de la variable temperatura (eje y), que muestra el valor mínimo, el máximo y los cuartiles primero (Q_1), segundo (Q_2: mediana) y tercero (Q_3).

Más ejemplos los pueden revisar en este enlace.

Ejemplo de *boxplot* en `seaborn`

Con `sns.boxplot`:

```
>>> import seaborn as sns
>>> df = sns.load_dataset('titanic')
>>> sns.boxplot(data=df, x="class", y="age")
>>> plt.ylabel("Edad")
>>> plt.xlabel("Clase")
>>> plt.show()
```

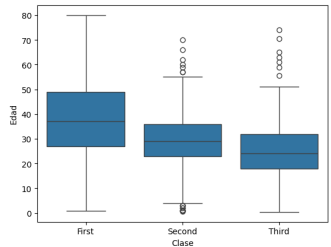


Figura 14: *Boxplot* de distribuciones de edades según clase en embarcación Titanic. Muestra los valores de edades mínimo, máximo y los cuartiles primero (Q_1), segundo (Q_2: mediana) y tercero (Q_3) separados por clase.

Más ejemplos los pueden revisar en este enlace.

Heatmaps

- Un *heatmap* es una visualización que representa valores numéricos mediante una escala de colores.
- Se utiliza para analizar patrones, concentraciones y relaciones en datos bidimensionales.
- Permite identificar rápidamente zonas de valores altos y bajos.
- Es comúnmente usado para visualizar matrices de correlación, distancias o frecuencias.
- El color es el principal canal de codificación del valor.

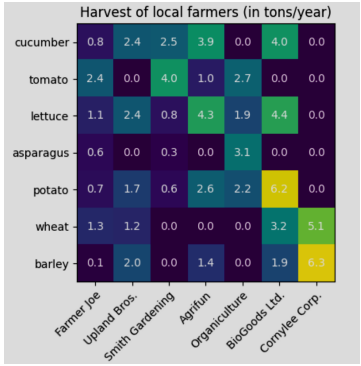


Figura 15: Ejemplo de *heatmap*. El eje x representa a diferentes granjeros; el eje y representa verduras. El color en la matriz representa la cantidad de verduras cosechadas (en toneladas por año) para cada combinación.

Heatmap con `matplotlib` (`imshow`)

Con `plt.imshow`:

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> data = np.random.rand(4, 4)
>>> plt.imshow(data, cmap="viridis")
>>> plt.colorbar(label="Valor")
>>> plt.xlabel("Indice columna")
>>> plt.ylabel("Indice fila")
>>> plt.title("Heatmap usando imshow")
>>> plt.show()
```

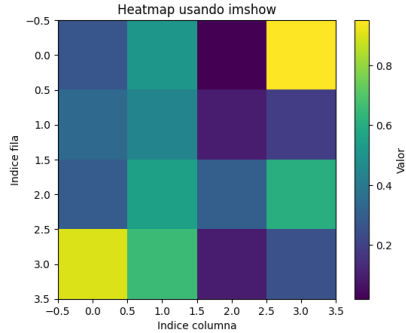


Figura 16: *Heatmap* generado con `imshow` en `matplotlib`. Los colores representan los valores numéricos de una matriz de 4×4 generada aleatoriamente, donde cada celda se codifica mediante una escala de color.

Más ejemplos los pueden revisar en este enlace.

Heatmap con `seaborn`

Con `sns.heatmap`:

```
>>> import seaborn as sns
>>> df = sns.load_dataset("iris")
>>> corr = df.corr(numeric_only=True)
>>> sns.heatmap(corr, annot=True,
...             cmap="coolwarm")
>>> plt.title("Matriz de correlación")
>>> plt.show()
```

Más ejemplos los pueden revisar en este enlace.

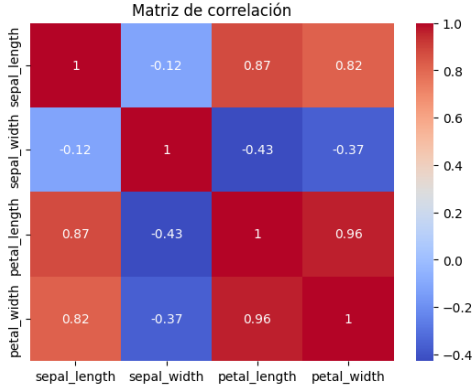


Figura 17: *Heatmap* de la matriz de correlación del dataset Iris, donde los colores indican la intensidad y el signo de la correlación lineal entre las variables numéricas, y los valores anotados corresponden a los coeficientes de correlación.

Ejemplo práctico

Exploraremos un conjunto de datos que tiene información sobre Pokémon en Google Colab, para aplicar lo aprendido en esta clase.

matplotlib (Resumen)

```
import matplotlib.pyplot as plt # Importación
```

Función	Uso	Ejemplo
`plot`	Gráfico de línea	`plt.plot(valores)`
`scatter`	Gráfico de puntos	`plt.scatter(valores_x, valores_y)`
`bar`	Gráfico de barras	`plt.bar(etiquetas, valores)`
`boxplot`	Gráfico de caja	`plt.boxplot(valores)`
`histogram`	Histograma	`plt.hist(valores)`
`imshow`	Gráfico *heatmap*	`plt.imshow(matriz)`

Tabla 1: Funciones en `matplotlib`.

seaborn (Resumen)

```
import seaborn as sns # Importación
```

- `lineplot`: gráfico de líneas entre dos columnas de un *DataFrame*.
  `sns.lineplot(data=dataframe, x="x", y="y")`
- `scatterplot`: gráfico de dispersión entre dos columnas de un *DataFrame*.
  `sns.scatterplot(data=dataframe, x="x", y="y")`
- `histplot`: histograma de una columna de un *DataFrame*.
  `sns.histplot(data=dataframe, x="x")`
- `boxplot`: comparación entre distribuciones de variables mirando sus cuartiles.
  `sns.boxplot(data=dataframe, x="grupo", y="valor")`
- `heatmap`: visualización matricial donde los colores representan los valores numéricos de una tabla, comúnmente usada para analizar relaciones entre variables (p. ej., matrices de correlación).
  `sns.heatmap(dataframe, annot=True)`

Referencias

Wes McKinney. *Python for Data Analysis*. O'Reilly Media, 3 edition, 2022.