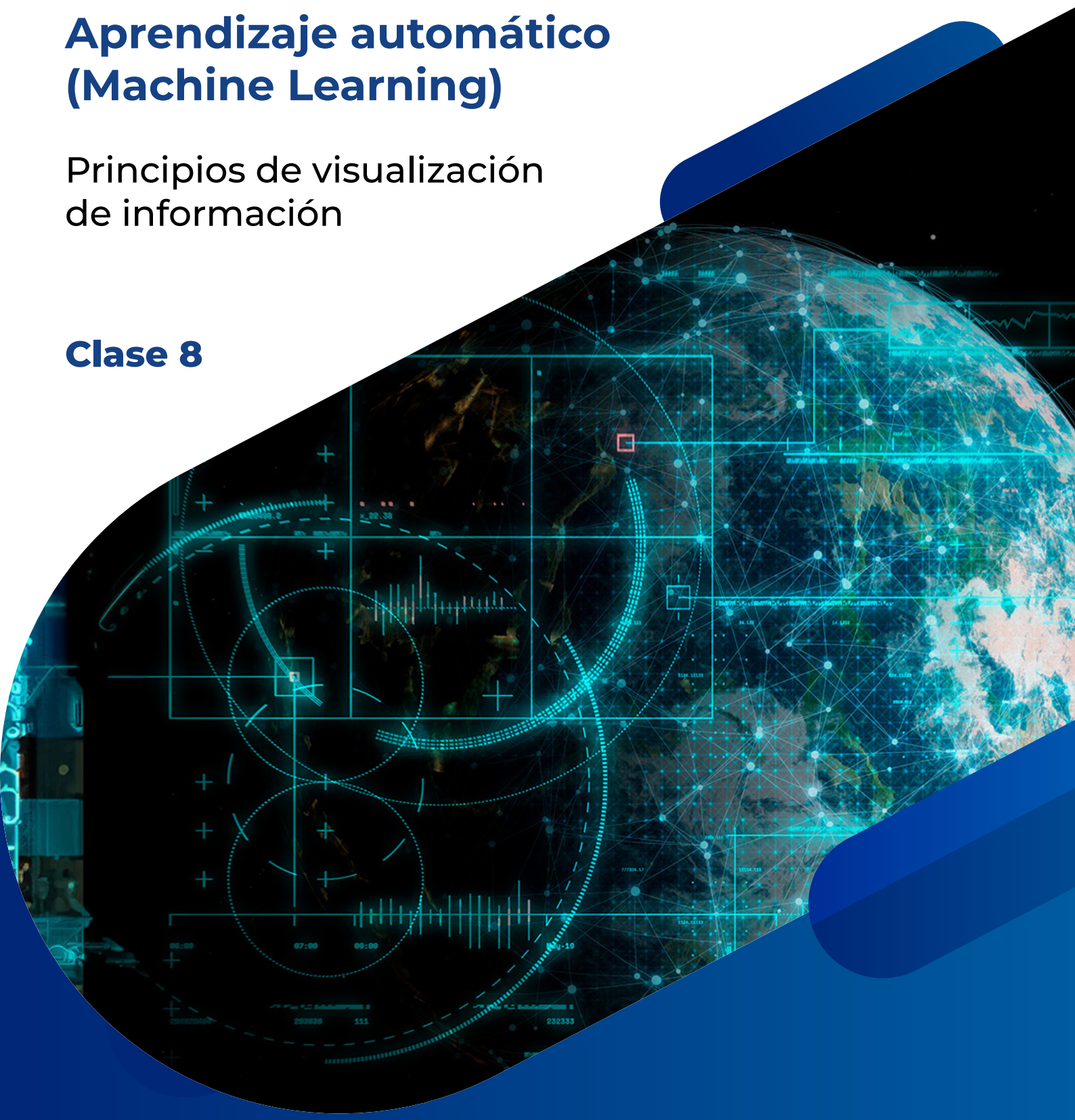
PUCE Virtual
EDUCACIÓN FLEXIBLE

Aprendizaje automático (Machine Learning)

Principios de visualización de información

Clase 8



MAESTRÍA EN
SISTEMAS DE INFORMACIÓN
Mención Data Science

La excelencia no se improvisa

INTRODUCCIÓN

La **visualización de datos** es una disciplina esencial para la comprensión y comunicación efectiva de **información compleja**. Sin embargo, es crucial entender que no todos los gráficos son igual de útiles o informativos. Los llamados ***gráficos basura*** son aquellos que incorporan elementos visuales innecesarios –colores distractores o leyendas superfluas–, que pueden dificultar la interpretación de los datos. De forma contraria, los ***minigráficos*** son representaciones compactas de grandes volúmenes de datos, que permiten resaltar tendencias clave **sin sobrecargar al espectador** (Tufte, 2001). Es fundamental, además, evitar los ***gráficos engañosos***, que manipulan la escala, omiten datos o alteran visualmente la realidad y pueden llevar a conclusiones erróneas (Cleveland, 1993).

Para crear visualizaciones efectivas, los diseñadores de gráficos deben considerar herramientas como la ***arquitectura Matplotlib***, que proporciona una base robusta para la creación de gráficos en Python (Hunter, 2007). Entre los tipos más comunes de visualizaciones se incluyen los ***scatterplots*****,** ***lineplots*** **y** ***barcharts;*** cada uno adecuado para diferentes tipos de relaciones de datos, como **correlaciones o comparaciones** (Wilke, 2019). Finalmente, uno de los principios más importantes al **crear gráficos** es la eliminación del *ruido visual;* es decir, la reducción de elementos que no aportan valor, lo que mejora la claridad y permite que el mensaje principal sea más fácilmente comprendido (Knaflic, 2015).

RDA 3: Implementa modelos de aprendizaje automático de manera efectiva en sistemas de software.

Clase 8. Principios de visualización de información

Gráficos basura

Los gráficos basura son representaciones visuales de datos que, debido a un diseño inadecuado o a la inclusión de elementos innecesarios, dificultan la correcta interpretación de la **información**. En el análisis de datos, los gráficos deben ser claros, concisos y relevantes, pero los gráficos basura suelen incorporar elementos que confunden al espectador, como colores excesivos, **imágenes decorativas** o **distorsiones** en las escalas de los ejes; esto afecta a la calidad y precisión de la visualización (Tufte, 2001). Estas visualizaciones no solo desvirtúan los patrones que se desean mostrar, sino que pueden inducir a errores de interpretación, ya que ocultan o exageran las tendencias subyacentes de los datos (Knaflic, 2015).

En este contexto, es importante reconocer que la simplicidad y la claridad son principios claves en la visualización de datos. Un gráfico debe servir como herramienta para descubrir *insights*, y no para desorientar al usuario con elementos visuales superfluos. La eliminación de estos '**ruidos visuales'** y la correcta **representación** de los **datos** pueden marcar la diferencia entre una visualización efectiva y un gráfico basura (Cleveland, 1993). En consecuencia, al crear gráficos en el análisis de datos, se deben seguir principios como la correcta escala, la eliminación de redundancias y la selección de los tipos adecuados de gráficos –como *scatterplots* o *lineplots*– que permiten revelar las relaciones entre los datos sin distracciones (Wilke, 2019).

Hay varios tipos de *gráficos basura,* uno de ellos es ***unintended optical art,*** que se refiere a la aparición de efectos visuales no intencionales, que surgen cuando elementos gráficos o patrones **interactúan** con la percepción del **espectador**, generando **ilusiones ópticas no deseadas**. Este fenómeno ocurre cuando la **disposición** de **colores**, formas o estructuras causan **distorsiones** o efectos visuales inesperados, sin que el diseñador haya planeado específicamente crear una ilusión óptica. Aunque este tipo de arte no es deliberado, puede interferir con la claridad de la **visualización**, generando **confusión** y dificultando la correcta **interpretación** de la **información** (Knaflic, 2015). En el análisis de datos, los gráficos con *unintended optical art* pueden actuar como ruido visual, desvirtuando la transmisión efectiva del mensaje que se intenta comunicar (Tufte, 2001). En la Figura 1 se muestra un ejemplo de este tipo de **gráfico basura.**

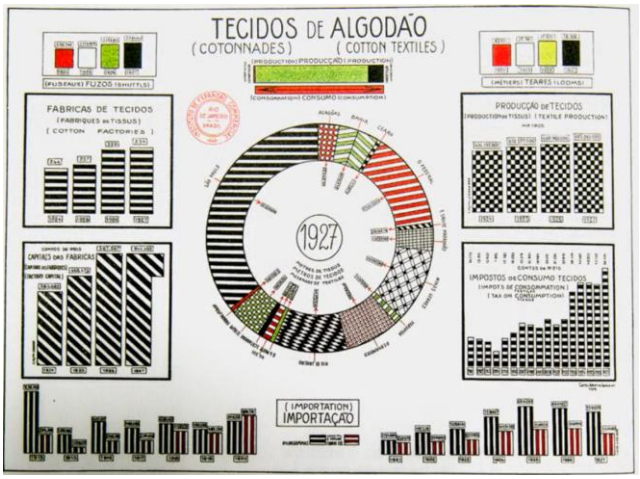


Figura 1. *Unintended optical art example.*

Otra forma de crear un ***gráfico basura*** es añadiendo líneas innecesarias o un exceso de *labels* o aclaraciones de texto en los gráficos. Estas cosas, lejos de aportar al entendimiento de la visualización, pueden generar mayor dificultad para que sea entendido e **interpretado** de manera correcta.

Una tercera forma o tipo de **gráfico basura** es **The Duck,** una **metáfora** utilizada para describir la apariencia **superficial** y **engañosa** de un **gráfico** que, aunque parece ser informativo a simple vista, **no ofrece una representación precisa o clara de los datos**. Esta expresión se refiere a la **discrepancia** entre la **primera impresión** que genera el gráfico (lo que 'parece un pato') y la realidad subyacente de la información presentada, lo que puede conducir a **interpretaciones erróneas** (Tufte, 2001). En otras palabras, un gráfico que **inicialmente parece correcto puede**, al ser analizado más a fondo, revelar **distorsiones**, **omisiones** o **manipulaciones** que afectan su precisión y utilidad, lo que resalta la importancia de una visualización que sea tanto estéticamente atractiva como fiel a los datos (Knaflic, 2015). En Figura 2 se muestra un ejemplo.

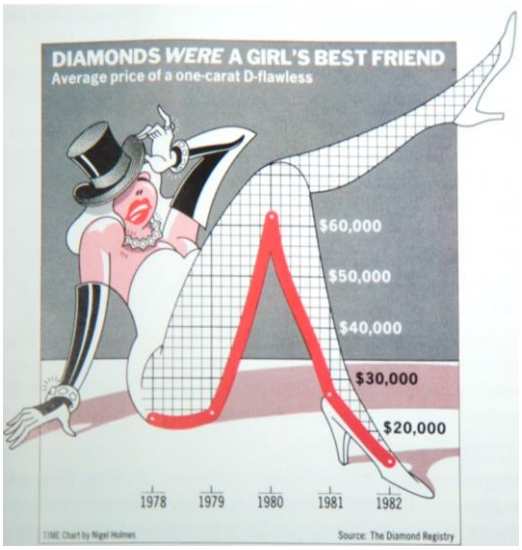


Figura 2. *The Duck Chart Example.*

Useful Junk The effects of visual embellishment on.pdf

Minigráficos

Los ***minigráficos*** son representaciones **visuales compactas,** que permiten mostrar **grandes volúmenes** de datos de manera eficiente y concisa. Estas pequeñas **visualizaciones** ofrecen una forma de resumir patrones y tendencias clave –como la evolución temporal o las variaciones en un conjunto de datos– sin sobrecargar al espectador con información **innecesaria** (Tufte, 2001). Los **minigráficos** suelen **utilizarse** en tablas o **informes,** donde se requiere una **visualización rápida** pero informativa, **proporcionando** una visión general sin desentonar con el diseño general del documento (Knaflic, 2015). Al igual que las **representaciones** gráficas **más grandes**, los minigráficos deben seguir los **principios** de **claridad** y **simplicidad,** para evitar confusión; su efectividad depende de la elección adecuada de los tipos de gráficos que los componen, como líneas o barras, en función del tipo de datos que se desean mostrar (Wilke, 2019).

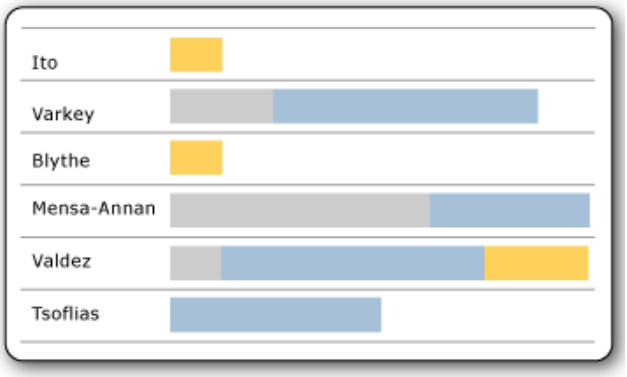


Figura 3. Minigráficos.

Minigráficos en Power BI: https://learn.microsoft.com/es-es/power-bi/paginated-reports/report-design/sparklines-data-bars-report-builder

Además, la clave de los **minigráficos** radica en su capacidad para ofrecer un **contexto visual** inmediato, sin que el espectador tenga que desplazarse por grandes cantidades de información. Esta síntesis **visual facilita** la comprensión rápida de tendencias y anomalías en los datos (Cleveland, 1993). Sin embargo, como con cualquier tipo de **visualización**, su diseño debe evitar la sobrecarga de detalles y **garantizar** que los **datos** que se muestran sean claros y relevantes, ayudando a los usuarios a tomar decisiones **informadas** de **manera eficiente**.

Arquitectura Matplotlib

Matplotlib es una biblioteca de **visualización** de datos en **Python,** que se utiliza para crear gráficos estáticos, animados e interactivos. Es muy popular en la comunidad de ciencia de datos, debido a su **versatilidad** y **facilidad** de uso para generar gráficos de **líneas**, **barras**, **dispersión**, **histogramas** y muchos más. Permite personalizar todos los aspectos del **gráfico**, como colores, etiquetas y leyendas; y es compatible con otras bibliotecas, como *NumPy* y *Pandas*.

Los orígenes de ***Matplotlib*** se remontan a 2003, cuando fue creado por John D. Hunter, un neurocientífico y programador. Hunter quería una herramienta para visualizar datos en Python que fuera más potente y flexible que las opciones existentes en ese momento. Al principio, la biblioteca se desarrolló como una alternativa a **MATLAB**, que era muy popular en su época, pero no tenía una versión gratuita o de código abierto. Con el tiempo, **Matplotlib** se convirtió en la biblioteca de **visualización** estándar en Python, consolidándose como una herramienta clave para el **análisis** y la **presentación** de datos en ciencia, ingeniería y matemáticas.

La **arquitectura de Matplotlib** se estructura en torno a varios componentes claves que facilitan la creación y personalización de gráficos. El **componente Figura (Figure)** es el contenedor principal, y puede incluir uno o más objetos de tipo **Ejes (Axes)**. La figura representa el lienzo global donde se dibujan los elementos visuales y organiza el espacio para los gráficos, subgráficos, leyendas y otras anotaciones (Hunter, 2007).

Los **Ejes (Axes)** son los objetos responsables de definir los sistemas de coordenadas y cómo se representan los datos dentro de la figura. Cada gráfico o subgráfico tiene uno o más ejes, que incluyen configuraciones como las escalas de los ejes X e Y, títulos, etiquetas y líneas de rejilla. Los ejes actúan como los 'espacios de trabajo' donde se grafican los datos (Hunter, 2007).

Dentro de los ejes, los elementos visuales, como **líneas, puntos, barras y textos,** son representados por objetos llamados **artistas (Artists)**. Cada artista tiene propiedades ajustables, como color, tamaño y estilo, esto permite personalizar la visualización según las necesidades del usuario (Droettboom et al., 2015).

Finalmente, **Canvas** es el área de dibujo donde los artistas son renderizados. Canvas está vinculado al **backend**, que es el sistema responsable de mostrar los gráficos en diferentes plataformas, como en pantallas, o guardarlos en archivos de imagen (Hunter, 2007).

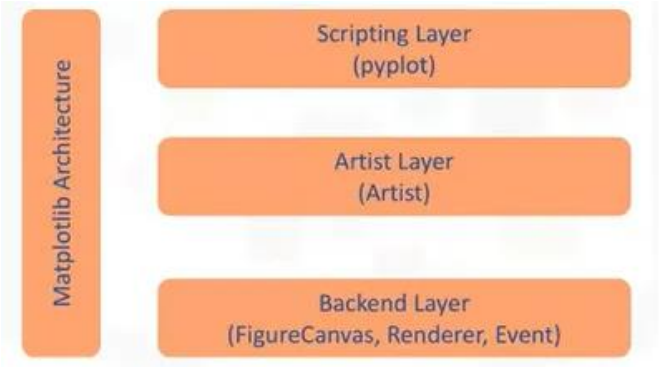


Figura 4. Arquitectura de *Matplotlib.*

Data Visualization [https://geekscoach.medium.com/data-visualization-matplotlib-e67c9cd64d1]

Scatterplots, lineplots, barcharts

A continuación, se brindan algunos ejemplos que permiten entender el funcionamiento y el uso de *Matplotlib.*

```
import matplotlib.pyplot as plt

plt.figure() # instanciamos el objeto figura
plt.plot(3,2,'o') # punto en el gráfico
plt.plot(1.5, 1.5, 'o')
plt.plot(2, 2, 'o')
plt.plot(2.5, 2.5, 'o')
ax = plt.gca() # objeto que define el área dentro de una figura donde se dibujan los gráficos
ax.axis([0,4,0,6]) # definimos el máximo y mínino del eje x e y
```

0.2s

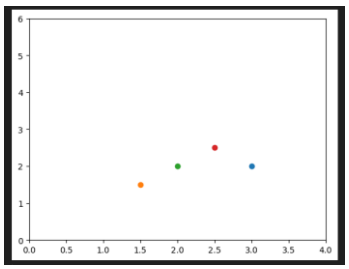


Figura 5. Uso de *Matplotlib*.

En la Figura 5 se muestra cómo **llamar** a la librería de ***Matplotlib*** para generar **visualizaciones**, cómo instanciar un **objeto** de tipo ***figure*** y añadir diferentes series de datos. Finalmente, definimos los valores **mínimos** y **máximos** de los ejes **x e y,** respectivamente.

Scatterplot

Un ***scatterplot*** o **gráfico de dispersión** es una representación gráfica utilizada para mostrar la **relación** entre **dos variables numéricas**. Cada punto en el gráfico corresponde a una observación en el conjunto de datos; donde la posición en el eje X representa el valor de una variable y la posición en el eje Y representa el valor de la otra. Los *scatterplots* son útiles para **visualizar patrones**, **correlaciones** o **tendencias** entre las variables, permitiendo identificar de manera rápida si existe una relación lineal, no lineal o incluso si los datos están dispersos sin una clara relación. También son comúnmente utilizados para detectar posibles **outliers** o valores **atípicos** en los **datos**.

```python
import matplotlib.pyplot as plt

x = [1,2,3,4,5]
y = [6,7,8,9,10]

plt.figure()
plt.scatter(x[:2], y[:2], s = 100, c= 'red', label = 'Estudiantes altos')
plt.scatter(x[2:], y[2:], s = 100, c= 'blue', label = 'Estudiantes bajos')

plt.xlabel('Número de veces que el estudiante patea una pelota')
plt.ylabel('La calificaicón del estudiante')
plt.title('Relación entre patear una pelota y las calificaciones')
plt.legend()
```

Figura 6. *Scatterplot* básico.

La Figura 6 muestra un ejemplo de creación de ***scatterplot*** con ***Matplotlib.*** Primero cargamos la librería, posteriormente definimos la lista x e y. Definimos el objeto figure y luego realizamos dos ***scatterplots*** en el mismo gráfico, uno para los puntos bajos y el otro para los puntos altos. Finalmente, añadimos algunas configuraciones, como el *labeling* de x e y, así como el **título** y las **leyendas** (los nombres de los dos grupos).

Lineplots

Un ***lineplot*** o **gráfico de líneas** es una **visualización** que representa la relación entre dos variables numéricas mediante una línea continua que conecta los **puntos** de **datos** en el orden en que aparecen. Se usa comúnmente para mostrar tendencias a lo largo del tiempo, como **series temporales**, y permite **identificar patrones** como aumentos, disminuciones o fluctuaciones en los datos. Los ***lineplots*** son especialmente útiles cuando se desea analizar la evolución de una variable dependiente en función de una variable independiente, como el cambio de precios a lo largo de los meses o la variación de **temperatura** en diferentes días.

```python
# line plots
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
linear_data = np.array([1,2,3,4,5,6,7,8])
quadratic_data = linear_data**2
plt.figure()
plt.plot(linear_data, '-o', quadratic_data, '-o')
plt.xlabel('data 1')
plt.ylabel('data 2')
plt.title('El titulo')
plt.legend(['Baseline', 'Competition'])
```

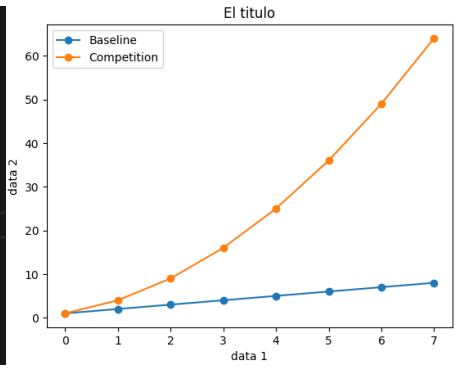


Figura 7. Primer ejemplo de *lineplots*.

En la Figura 7 tenemos un gráfico de líneas que se ejecuta con la función ***plot*** de ***Matplotlib***, primero cargamos las librerías, luego generamos los datos con ***numpy***. Posteriormente, creamos la instancia figura y hacemos el ***plot*** (en este caso solo le damos la coordenada y dado que ***plt.plot*** reconoce a el índice de las series como coordenada x).

```python
plt.figure()

observation_dates = np.arange('2025-01-01', '2025-01-09' ,dtype = 'datetime64[D]')
observation_dates = list(map(pd.to_datetime, observation_dates))

plt.plot(observation_dates, linear_data, '-o',
         observation_dates, quadratic_data)

x = plt.gca().xaxis

for item in x.get_ticklabels():
    item.set_rotation(45)

plt.xlabel('Fechas')
plt.ylabel('Unidades')
plt.title('Performance Linear vs. Cuadratico')
```

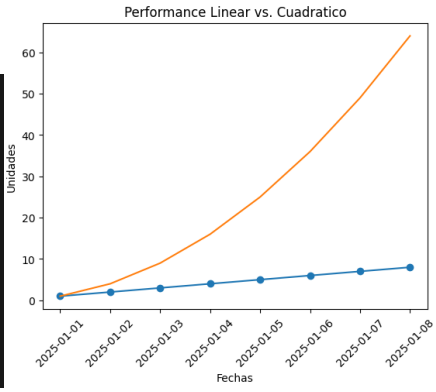


Figura 8. Segundo ejemplo de *lineplots*.

En la Figura 8 se puede apreciar que añadimos como coordenadas x valores de fechas usando la función de ***Pandas to_datetime;*** adicionalmente, aplicamos la función ***map*** y, posteriormente, lo convertimos en un objeto **lista,** para que se puede usar en el gráfico. Finalmente, rotamos los valores de fechas usando ***plt.gca().xaxis*** y obteniendo cada uno de los ***labels,***

Barcharts

Un ***barchart*** o **gráfico de barras** es una representación visual utilizada para comparar valores de diferentes categorías. Consiste en barras rectangulares, cuya altura (en gráficos verticales) o longitud (en gráficos horizontales) es proporcional al valor que representan. Se emplea comúnmente para visualizar distribuciones, comparar cantidades y mostrar datos categóricos, como ventas por producto o población por país. Los ***barcharts*** permiten **identificar** rápidamente **diferencias** entre grupos, tendencias y patrones en los datos, facilitando el **análisis** y la **toma** de **decisiones**.

```python
# barchart
import numpy as np
import matplotlib.pyplot as plt


plt.figure()

linear_data = np.array([1,2,3,4,5,6,7,8])
quadratic_data = linear_data**2

xvals = range(len(linear_data))
new_xvals = [x  +0.3 for x in xvals]

plt.bar(xvals, linear_data, width= 0.3)
plt.bar(new_xvals, quadratic_data, width= 0.3)
```

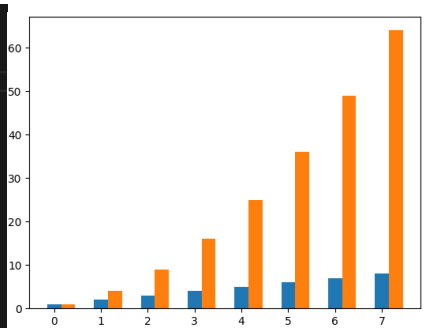


Figura 9. Ejemplo de *barchart*.

En la Figura 9 se muestra un ejemplo de ***barchart*** en el que, al igual que los casos anteriores, importamos las **librerías** e instanciamos el ***plt.figure,*** así como definimos los datos de entrada. Finalmente, graficamos las dos series de datos con dos ***plt.bar***, eso hace que las dos visualizaciones se puedan apreciar en el mismo **gráfico.**

```python
plt.barh(xvals, linear_data, height= 0.3, color = 'b')
plt.barh(xvals, quadratic_data, height= 0.3, left = linear_data, color = 'r')
```

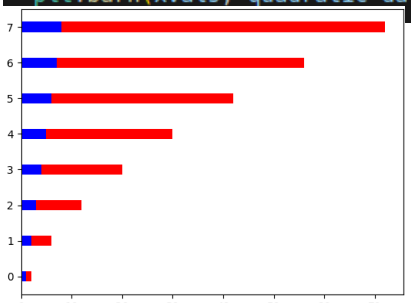


Figura 10. Ejemplo de *barchart* horizontal.

Finalmente, en la Figura 10 se muestra un ejemplo de ***barchart*** horizontal. El objeto que usamos es ***plt.barh;*** cambiando ciertos parámetros, tal y como se indica en la imagen, obtenemos las **barras horizontales.**

Eliminar ruido de un gráfico

En ***machine learning***, la eliminación de ruido en un gráfico es fundamental para mejorar la interpretación de los datos y optimizar modelos predictivos. A continuación, se presentan varias estrategias para reducir el ruido en gráficos utilizando diferentes técnicas de procesamiento y filtrado de datos.

Uno de los enfoques más comunes es el uso de **suavizado de datos,** mediante técnicas como el **método de media móvil**, donde cada punto en el gráfico se reemplaza con el promedio de un subconjunto de valores vecinos. Esto ayuda a reducir fluctuaciones aleatorias y a resaltar tendencias en los datos (Brownlee, 2018). Otro método es el **suavizado exponencial**, que asigna pesos decrecientes a los valores históricos, permitiendo reducir la influencia de valores atípicos y enfatizar la información más reciente (Hyndman & Athanasopoulos, 2018).

Desde la perspectiva de **preprocesamiento de datos en *machine learning***, los algoritmos de **filtrado de ruido**, como el **filtro de mediana**, son efectivos para eliminar valores atípicos y preservar bordes en señales o imágenes. Este método reemplaza cada punto con la mediana de los valores vecinos, reduciendo el impacto del ruido impulsivo en gráficos de datos discretos (Gonzalez & Woods, 2018). Además, técnicas como el **filtrado gaussiano** suavizan los datos aplicando una convolución con una función gaussiana, eliminando fluctuaciones aleatorias sin afectar demasiado a la estructura subyacente (Goodfellow et al., 2016).

Otro enfoque relevante en ***machine learning*** es la **reducción de dimensionalidad**, que permite eliminar ruido y preservar solo las características más importantes de los datos. Métodos como el **Análisis de Componentes Principales (PCA)** transforman los datos en un nuevo espacio de menor dimensión, eliminando redundancias y filtrando información irrelevante (Jolliffe & Cadima, 2016). De manera similar, los ***autoencoders*** basados en redes neuronales pueden aprender representaciones comprimidas de los datos, eliminando ruido al reconstruir la información más relevante (Goodfellow et al., 2016).

En el caso de gráficos de series temporales, se pueden aplicar **modelos de *machine learning* robustos,** como los **modelos ARIMA y Prophet,** que identifican tendencias y patrones eliminando variaciones aleatorias no significativas (Hyndman & Athanasopoulos, 2018). Además, técnicas de **detección y eliminación de *outliers***, como el uso de **métodos estadísticos (z-score, IQR)** o algoritmos como **Isolation Forest y DBSCAN**, pueden ayudar a limpiar los datos antes de visualizarlos, mejorando la calidad del gráfico y la interpretación del modelo (Liu et al., 2008).

En conclusión, la eliminación de ruido en gráficos dentro del contexto de *machine learning* es esencial para mejorar la interpretación de los datos y la precisión de los modelos. Técnicas como el suavizado de datos, filtrado de ruido, reducción de dimensionalidad y detección de *outliers* permiten obtener representaciones más claras y confiables, optimizando tanto la visualización como el análisis predictivo.

Mejores prácticas en visualización

***Scatterplots*:** se utilizan para analizar la relación entre dos variables numéricas.

Usar colores y transparencia (alpha)

- Si hay muchos puntos superpuestos, aplicar alpha=0,5 o menor, para mejorar la visibilidad.
- Usar una escala de colores (cmap) si hay una variable numérica adicional.

Añadir etiquetas y títulos claros

- Usar plt.xlabel(), plt.ylabel() y plt.title() para contextualizar la visualización.

Ajustar el tamaño de los puntos según la importancia

- Si se quiere resaltar ciertos puntos, cambiar su tamaño con s= , basado en otra variable.

Agregar líneas de tendencia si es relevante

- Utilizar numpy.polyfit() o sns.regplot() de Seaborn para mostrar tendencias.

Evitar sobrecargar con demasiados datos

- Si hay millones de puntos, considerar técnicas de muestreo para mejorar la legibilidad.

***Barcharts*:** se usan para comparar valores categóricos.

Ordenar las barras de mayor a menor

- Ayuda a interpretar mejor la información.

Evitar el uso excesivo de colores

- Un solo color con variaciones de tonalidad es más limpio visualmente.

Etiquetar las barras cuando sea necesario

- Añadir valores en las barras con plt.text() puede mejorar la interpretación.

Usar barh() para gráficos horizontales si los nombres son largos

- Mejora la legibilidad de etiquetas largas.

Agregar márgenes adecuados

- Evitar que las barras queden pegadas a los límites del gráfico (plt.ylim() o plt.xlim()).

Linecharts: son ideales para representar tendencias a lo largo del tiempo.

Usar líneas suaves y colores adecuados

- Evitar líneas gruesas o colores que dificulten la lectura.
- Se recomienda linewidth=2 y colores de alta visibilidad.

Agregar puntos en los datos clave (marker)

- Mejora la interpretación de tendencias (marker='o').

Manejar correctamente las escalas de tiempo en ejes X

- Si la variable en X representa fechas, usar plt.xticks(rotation=45).

Evitar conectar puntos con valores faltantes

- Se pueden interpolar los valores o usar np.nan para evitar uniones incorrectas.

Agregar líneas de referencia horizontales o verticales si es necesario

- Usar plt.axhline() o plt.axvline() para resaltar valores críticos.

Referencias citadas en la Clase 8

- Cleveland, W. S. (1993). *Visualizing data*. Hobart Press.
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90-95.
- Knaflic, C. (2015). *Storytelling with data: A data visualization guide for business professionals*. Wiley.
- Tufte, E. R. (2001). *The visual display of quantitative information*. Graphics Press.
- Wilke, C. O. (2019). *Fundamentals of data visualization*. O'Reilly Media.
- Droettboom, M., Caswell, T., & Hunter, J. D. (2015). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering, 12*(5), 10-11.
- Brownlee, J. (2018). *Machine Learning Mastery With Python: Understand Your Data, Create Accurate Models, and Work Projects End-To-End.* Machine Learning Mastery.
- Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing.* Pearson.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* MIT Press.
- Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: Principles and Practice.* OTexts.
- Jolliffe, I. T., & Cadima, J. (2016). Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 374*(2065), 20150202.
- Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation forest. *2008 Eighth IEEE International Conference on Data Mining*, 413-422.

Definición de los términos citados en la Clase 8

Minigráficos.
Los **minigráficos** o ***sparklines*** son pequeñas representaciones gráficas de datos que se integran dentro de textos, tablas o *dashboards*, para visualizar tendencias de manera rápida y compacta. Su principal ventaja es que permiten mostrar información clave sin ocupar mucho espacio, facilitando la **interpretación de patrones** sin necesidad de gráficos complejos. Son ideales para representar series temporales, comparaciones de valores o detectar anomalías en datos de manera eficiente. Además, al eliminar elementos visuales innecesarios, los **minigráficos** reducen el ruido visual y mejoran la accesibilidad a la información en reportes y paneles de control.

Gráficos basura.
Los **gráficos basura** (o *chartjunk*, término acuñado por Edward Tufte) son visualizaciones sobrecargadas de elementos innecesarios que dificultan la interpretación de los datos en lugar de facilitarla. Estos gráficos suelen incluir colores excesivos, efectos 3D innecesarios, sombras, patrones complejos o decoraciones que no aportan valor analítico. El problema principal de los gráficos basura es que distraen al espectador de la información esencial y pueden llevar a interpretaciones erróneas. Para evitarlo, se recomienda mantener un diseño limpio, eliminar adornos superfluos y enfocarse en una representación clara y efectiva de los datos.

PUCE Virtual
EDUCACIÓN FLEXIBLE

La excelencia no se improvisa

síguenos

MDASC - 12 - 241