guayerd

Fundamentos IA

Análisis con Python

Clase 8

En colaboración con

IBM SkillsBuild

¡Bienvenidos!

¿Nos presentamos?

- ¿Qué recuerdan de la clase anterior?
- ¿Qué esperan aprender?
- ¿Tienen alguna pregunta?

guayerd

En colaboración con

IBM SkillsBuild

Contenidos

Por temas

05
- Copilot Chat y prompts
- Demo asincrónica

06
- Limpieza y transformación

07
- Estadística aplicada

08
- Visualización

guayerd

En colaboración con

IBM **SkillsBuild**

Objetivos de la clase

- Visualización
- Matplotlib
- Seaborn

guayerd

En colaboración con

IBM SkillsBuild

Análisis con Python

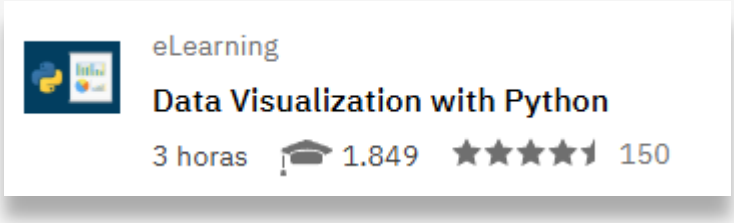
Visualización

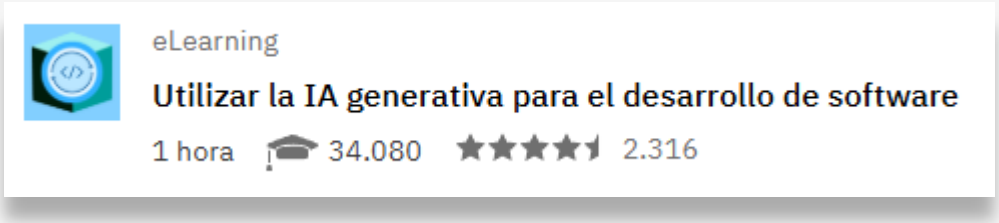
guayerd

En colaboración con

IBM SkillsBuild

Plataforma Skill Build: Python





guayerd

En colaboración con

IBM **SkillsBuild**

¿Cuál es la importancia de la visualización de datos?

La visualización de datos es crucial debido a su capacidad para proporcionar una comprensión profunda de tendencias, valores y patrones dentro de los datos. A través de representaciones gráficas, la información se vuelve más accesible y procesable.

Permite mayor contexto

Imagina que posees datos y tienes la habilidad de realizar análisis estadísticos, como cálculos de desviación estándar o correlación. Sin embargo, esto puede no ofrecer una visión completa de los datos. Por ejemplo, al visualizarlos gráficamente, utilizando un conjunto de datos como Datasaurus, se pueden identificar representaciones diversas a pesar de compartir la misma información.

Permite hallazgos en nuestros datos

La visualización de datos facilita la detección de hallazgos de manera directa y sencilla. Algunos ejemplos incluyen la capacidad para:

- Identificar tendencias.
- Descubrir comportamientos.
- Detectar valores atípicos (outliers).

Permite mayor claridad en el código

Además, la visualización de datos se utiliza en campos como la ciencia de datos orientada a machine learning e inteligencia artificial, como en las curvas de aprendizaje. Esto es fundamental en el entrenamiento de algoritmos, ya que las visualizaciones simplifican el entendimiento de aspectos complejos.

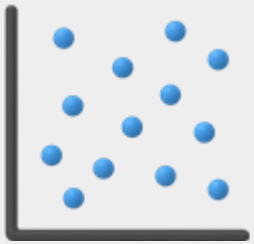
Librerías para visualización de datos

En el ámbito de la visualización de datos, dos librerías sobresalen como fundamentales:

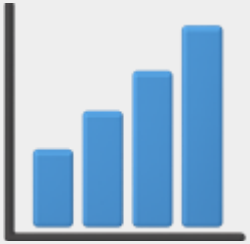
- **Matplotlib**
  - Emula comandos de Matlab.
  - Hace uso de Numpy.
  - Escrita en Python.
  - Facilidad y simplicidad de uso.
  - Personalización y rapidez en su implementación.
- **Seaborn**
  - Construida sobre Matplotlib.
  - Ofrece mayor abstracción y comodidad en la creación de gráficos.

Tipos de visualizaciones

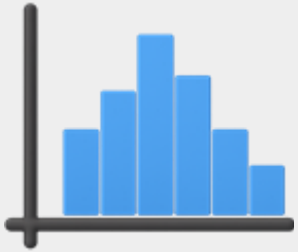
Evolución temporal



Relación entre variables



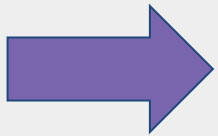
Comparación categórica



Distribución de datos

Identifica el gráfico

- Evolución de ventas durante 12 meses
- Comprar satisfacción entre 5 sucursales
- Analizar relación precio-demanda
- Distribución de edades de clientes
- Comparar ventas promedio entre temporadas
- Mostrar cambio de precios a lo largo de 2 años



- Líneas
- P
- Dispers
- Hist ma
- Barr
- Líneas

¿Qué es?

→ Matplotlib es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación Python y su extensión matemática NumPy.

→ Proporciona una API, pylab, diseñada para recordar a la de MATLAB.

5 › ⋮

Matplotlib

Librería fundamental para **crear gráficos en Python.**

- Compatible con arrays NumPy y DataFrames pandas
- Exportación en formatos profesionales (PNG, PDF, SVG)

Comandos

- **Instalación:** pip install matplotlib
- **Uso:** import matplotlib.pyplot as plt

Elementos fundamentales

Elemento	Código	Propósito
Crear figura	plt.figure(figsize=(8, 6))	Define tamaño del gráfico
Agregar datos	plt.plot(x, y)	Dibuja líneas y puntos
Títulos	plt.title('Texto')	Contexto del gráfico
Ejes	plt.xlabel('Variable'), plt.ylabel('Variable')	Claridad en interpretación
Mostrar	plt.show()	Renderizar gráfico final

Gráficos de líneas

Componente	Sintaxis	Descripción
Línea básica	plt.plot(x, y)	Conexión simple entre puntos
Con marcadores	plt.plot(x, y, marker='o')	Puntos visibles en cada dato
Múltiples series	plt.plot(x, y1), plt.plot(x, y2)	Comparar tendencias

Series temporales, métricas en evolución y comparación de tendencias

Gráficos de barras

Tipo	Sintaxis	Aplicación
Verticales	plt.bar(categorias, valores)	Comparaciones estándar
Horizontales	plt.barh(categorias, valores)	Etiquetas largas
Agrupadas	plt.bar(x, y1), plt.bar(x+width, y2)	Múltiples variables

Rankings, comparaciones categóricas y distribuciones por grupos

Gráficos de dispersión

Elemento	Sintaxis	Función
Puntos básicos	plt.scatter(x, y)	Relación simple
Tamaño variable	plt.scatter(x, y, s=tamaños)	Tercera dimensión
Colores por grupo	plt.scatter(x, y, c=categorias)	Segmentación

Correlaciones, segmentaciones y análisis multivariado

Histogramas

Parámetro	Sintaxis	Efecto
Básico	plt.hist(datos)	Frecuencias automáticas
Intervalos	plt.hist(datos, bins=20)	Control de granularidad
Transparencia	plt.hist(datos, alpha=0.7)	Comparar distribuciones

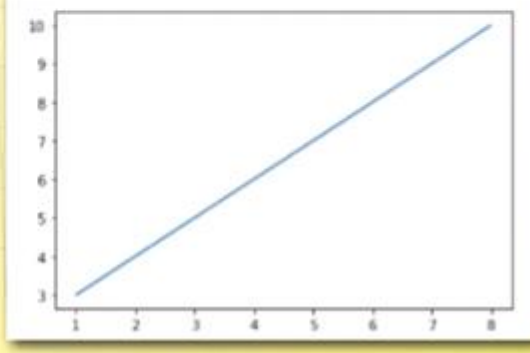
Análisis de normalidad, detección de valores atípicos y comparación de grupos

PLOT

¿Qué es?

- La función se utiliza para dibujar puntos (marcadores) en un diagrama.
- De forma predeterminada, la función dibuja una linea de un punto a otro.
- El parámetro 1 es una matriz que contiene los puntos del eje x.
- El parámetro 2 es una matriz que contiene los puntos del eje y.

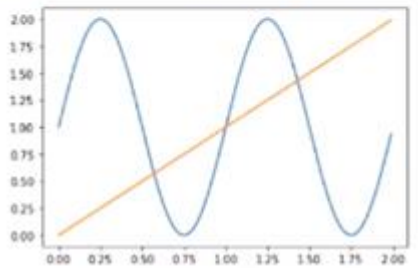
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.show()
```



Distintas figuras:

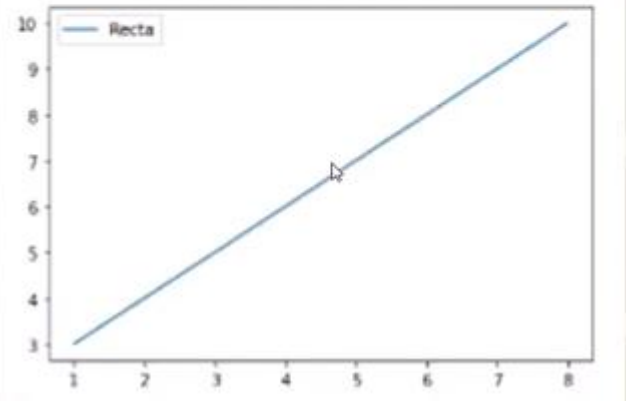
```
import matplotlib.pyplot as plt
import numpy as np

# Generamos las dos curvas
x = np.arange(0.0, 2.0, 0.01)
y = 1 + np.sin(2 * np.pi * x)

x2 = np.arange(0.0, 2.0, 0.01)
y2 = x2

# Generamos la figura y los ejes
fig = plt.figure()
ax = plt.axes()

# Ploteamos las dos líneas
ax.plot(x, y)
ax.plot(x2,y2)
```

```
[<matplotlib.lines.Line2D at 0x1f91633d7c0>]
```

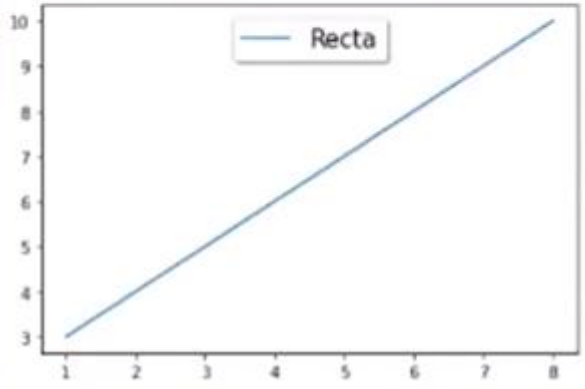


Leyendas:

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

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

ax = plt.axes()
plt.plot(xpoints, ypoints, label = 'Recta')
ax.legend()
plt.show()
```




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

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

ax = plt.axes()
plt.plot(xpoints, ypoints, label = 'Recta')
ax.legend(loc='upper center', shadow=True, fontsize=15)
plt.show()
```



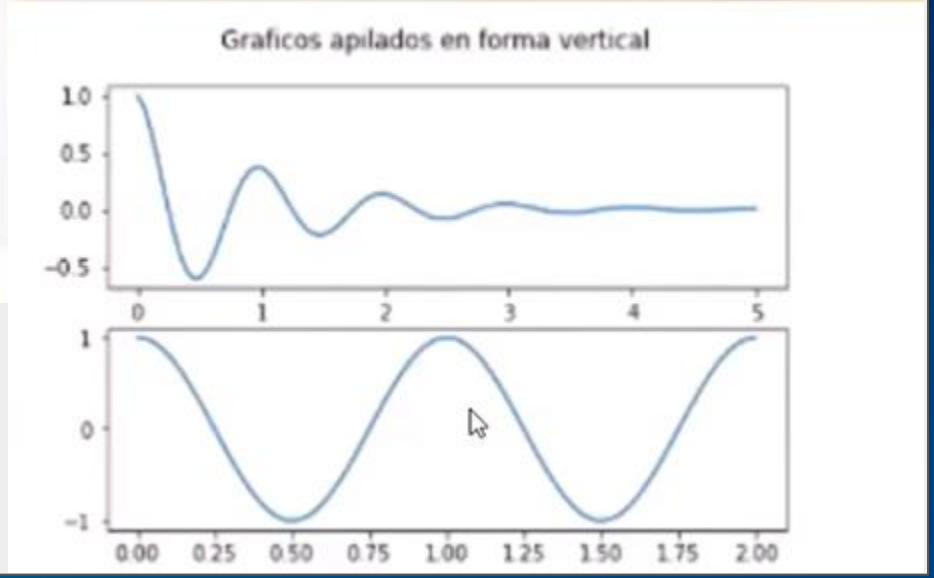
SubPlots

Para visualizar dos gráficos en una misma figura al mismo tiempo y compararlos debemos definir el objeto subplots de matplotlib.

subplots crea un objeto fig que corresponde a la figura (todo el rectángulo donde vamos a graficar) y varios ejes en el objeto axes, los cuales corresponden a los distintos subplots que vamos a hacer dentro de la figura.

```
# Primero definimos dos curvas (x1,y1) y (x2,y2)
x1 = np.linspace(0.0, 5.0, 100)
x2 = np.linspace(0.0, 2.0, 100)

y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)


# Creamos la figura 'fig' y dos ejes en 'axes'.
# Como ponemos (2,1) quiere decir que queremos 2 filas
fig, axes = plt.subplots(2,1)
fig.suptitle('Graficos apilados en forma vertical')
axes[0].plot(x1, y1)
axes[1].plot(x2, y2)
```

```
[<matplotlib.lines.Line2D at 0x1f9165e6af0>]
```

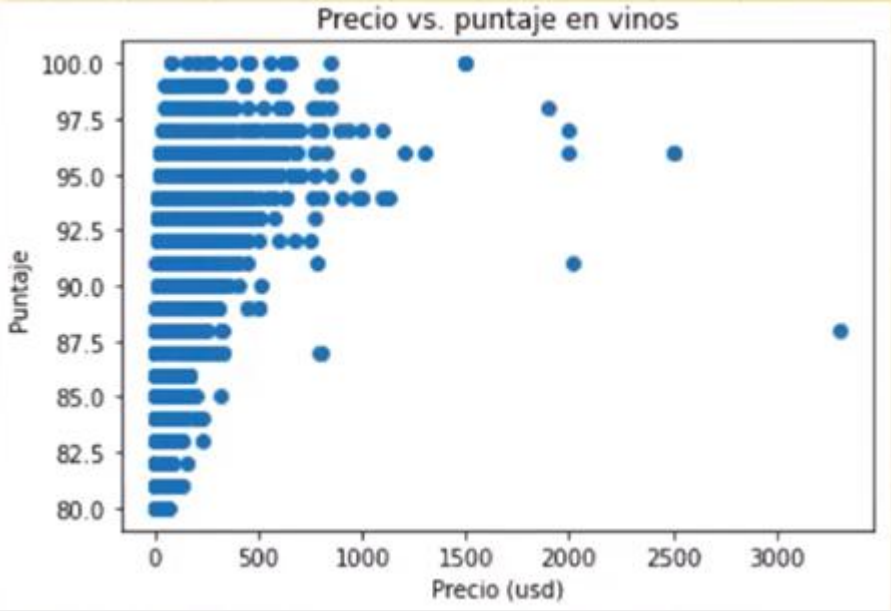


Scatter Plots:

- Otro tipo de gráfico que se utiliza mucho es el Scatter Plot.
- El mismo plotea una serie de puntos en un gráfico.
- En general estos puntos tomarán su coordenada horizontal de una lista de valores x y su coordenada vertical de una lista de valores y.

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

ax.scatter(x, y)
ax.set(xlabel='Precio (usd)', ylabel='Puntaje',
       title='Precio vs. puntaje en vinos')
# plt.plot(x, y)
```

```
[Text(0.5, 0, 'Precio (usd)'),
 Text(0, 0.5, 'Puntaje'),
 Text(0.5, 1.0, 'Precio vs. puntaje en vinos')]
```

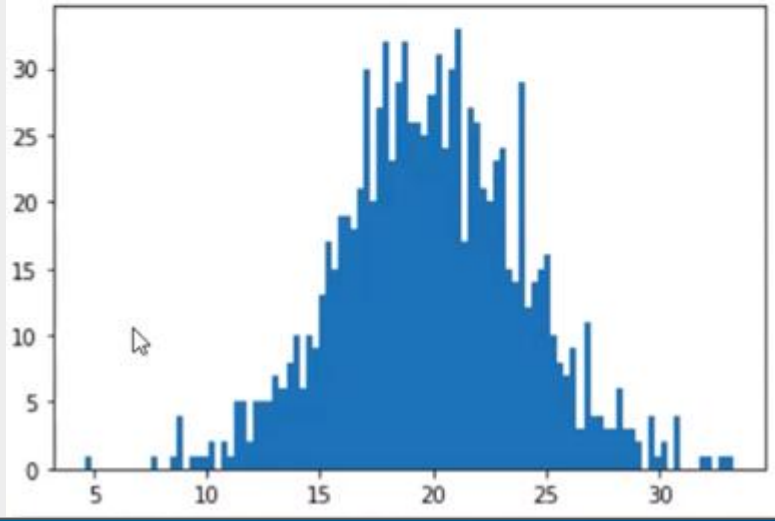


HISTOGRAMAS

Matplotlib también permite graficar fácilmente histogramas mediante la función hist.

Por ejemplo, creamos una serie de 1000 valores a partir de una distribución gaussiana en el vector valores...

```
# Aca decidimos la cantidad de bines que queremos tomar
num_bins = 100

# Creamos la figura y los ejes
fig, ax = plt.subplots()

# Ploteamos el histograma
n, bins, _ = ax.hist(valores, bins = num_bins)
```



Mejores prácticas

Títulos
Claros y con contexto

Colores
Paletas consistentes y accesibles

Texto
Legible y con buen contraste

Ejes
Escalas completas y proporcionales

Claridad
Gráficos simples y directos

Seaborn

Librería basada en Matplotlib para **gráficos estadísticos con diseño optimizado.**

- Sintaxis simplificada para gráficos complejos
- Integración directa con DataFrames de Pandas

Comandos

- **Instalación:** pip install seaborn
- **Uso:** import seaborn as sns

¿Qué es?

- Seaborn es una biblioteca utilizada principalmente para el trazado estadístico en Python.
- Está construido sobre Matplotlib y proporciona estilos predeterminados y paletas de colores para hacer que las gráficas estadísticas sean más atractivas.
- Esta librería nos permite generar gráficos a partir de DataFrames de Pandas de manera muy cómoda y accesible.

Gráficos estadísticos principales

Boxplot

Outliers y cuartiles por grupo

```
sns.boxplot(data=df,
x='cat', y='num')
```

Heatmap

Matriz de correlaciones

```
sns.heatmap(df.corr(),
annot=True)
```

Violinplot

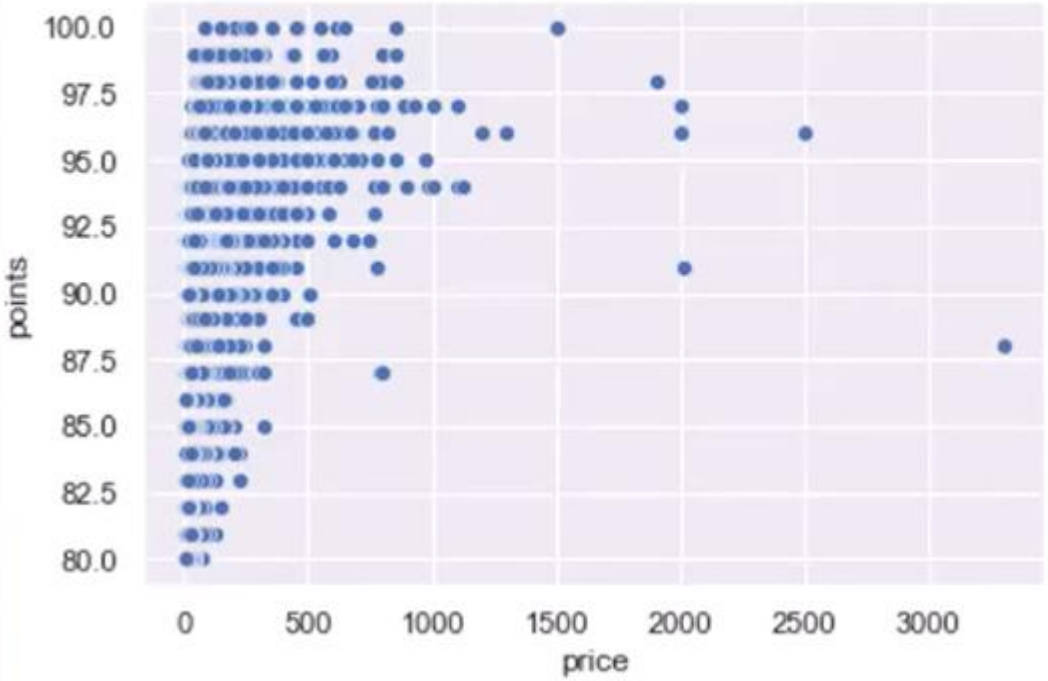
Densidad de distribuciones

```
sns.violinplot(data=d
f, x='cat', y='num')
```

ScatterPlot

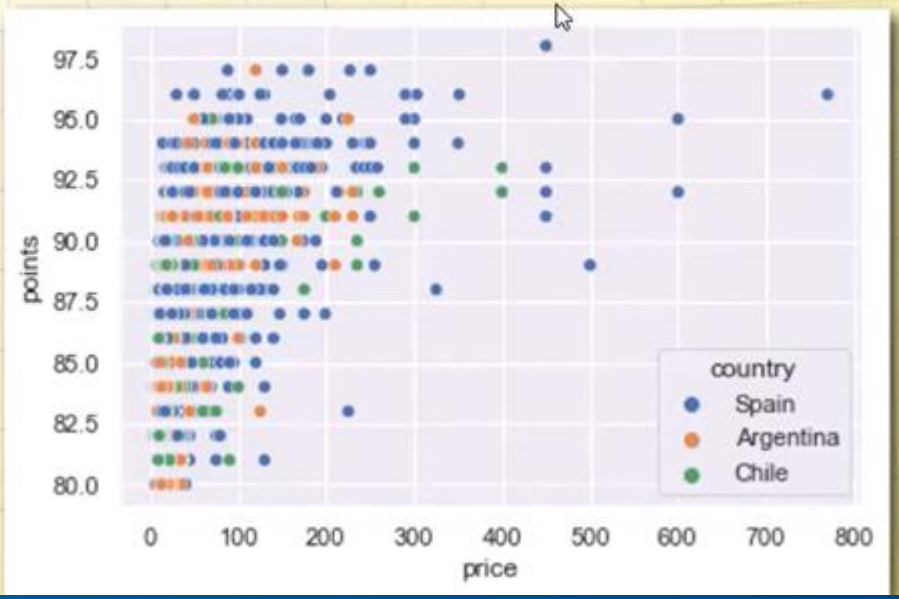
- Scatterplot se puede utilizar con varias agrupaciones semánticas que pueden ayudar a comprender bien en un gráfico.
- Pueden trazar gráficos bidimensionales que se pueden mejorar asignando hasta tres variables adicionales mientras se utiliza la semántica de los parámetros de tono, tamaño y estilo.
- Todos los parámetros controlan la semántica visual que se utilizan para identificar los diferentes subconjuntos.

```
ax = sns.scatterplot(x="price", y="points", data=wine_reviews)
```



Agregar labels:

```
ax = sns.scatterplot(x="price", y="points", hue="country", data=wine_reviews_filtradas)
```

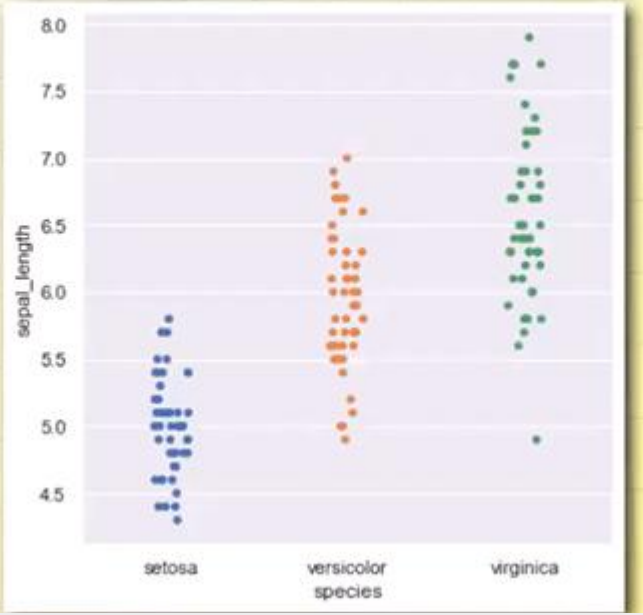


Categorical Plots

Como su nombre lo indica, los categorical plots son gráficos donde una de las variables a graficar es de tipo categórica.

Este tipo de gráficos son muy usados en Data Science y Seaborn tiene una función especial dedicada a ellos, catplot.

```
sns.catplot(data = iris_data, x = "species", y = "sepal_length",ci = "sd",estimator=np.median)
```

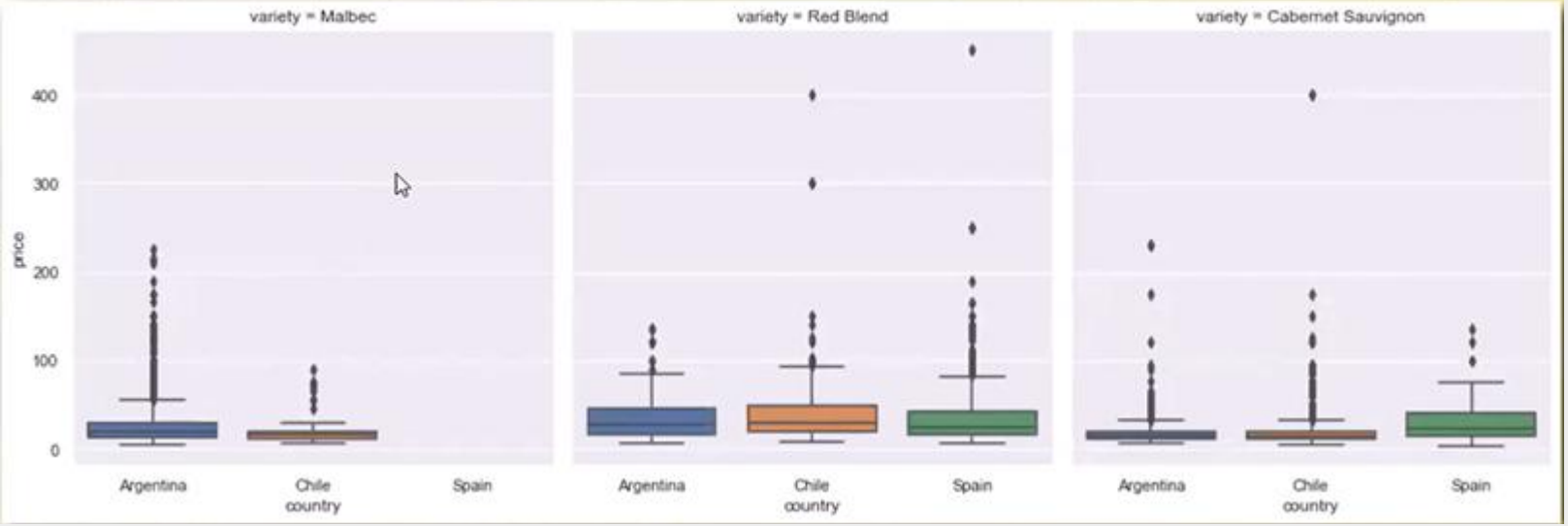


Box Plots

Un diagrama de caja ayuda a mantener la distribución de los datos cuantitativos de tal manera que facilita las comparaciones entre variables o entre niveles de una variable categórica.

- → Cuerpo principal: muestra los cuartiles y los intervalos de confianza de la mediana, si está habilitado.
- → Medianas: tienen líneas horizontales en la mediana de cada caja.
- → Bigotes: tienen las líneas verticales que se extienden hasta los puntos de datos más extremos.

```
sns.catplot(x="country", y="price", col="variety", kind='box' , data=vinos)
```



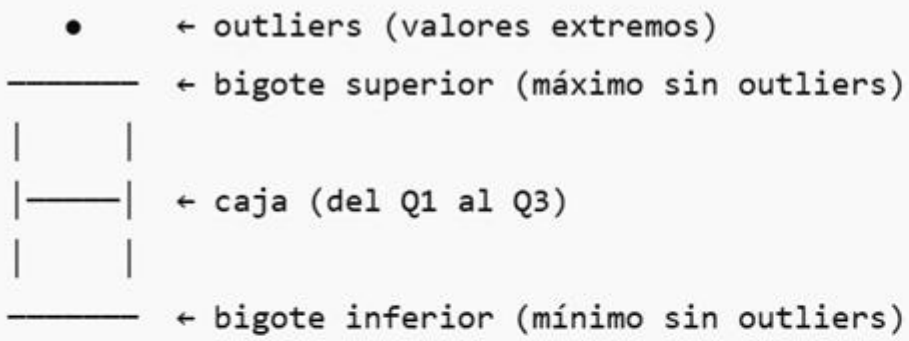
🧩 ¿Qué muestra un boxplot?

Un boxplot resume los datos mediante **cinco valores clave**:

1. **Mínimo** → el valor más pequeño (sin contar outliers).
2. **Primer cuartil (Q1)** → el valor que deja por debajo el 25 % de los datos.
3. **Mediana (Q2)** → el valor central (50 %).
4. **Tercer cuartil (Q3)** → el valor que deja por debajo el 75 % de los datos.
5. **Máximo** → el valor más grande (sin contar outliers).

Los **outliers** (valores atípicos) aparecen como **puntos separados** fuera de los "bigotes".

Partes de un boxplot



- **La caja** muestra el rango intercuartílico (**IQR = Q3 - Q1**).
- **La línea dentro de la caja** es la **mediana**.
- **Los bigotes** suelen llegar hasta `1.5 × IQR` por encima o por debajo de los cuartiles.
- **Los puntos fuera de ese rango** se consideran **outliers**.

Qué son los bigotes

Los **bigotes** muestran **hasta dónde llegan los valores que no se consideran atípicos (outliers)**.

- **Bigote inferior:** se extiende desde el **límite inferior de la caja (Q1)** hacia el valor más bajo que todavía está dentro del rango permitido.
- **Bigote superior:** se extiende desde el **límite superior de la caja (Q3)** hacia el valor más alto dentro del rango permitido.

Cálculo del rango permitido

El rango permitido se define así:

$$\text{Rango permitido} = [Q1 - 1.5 \times IQR,\ Q3 + 1.5 \times IQR]$$

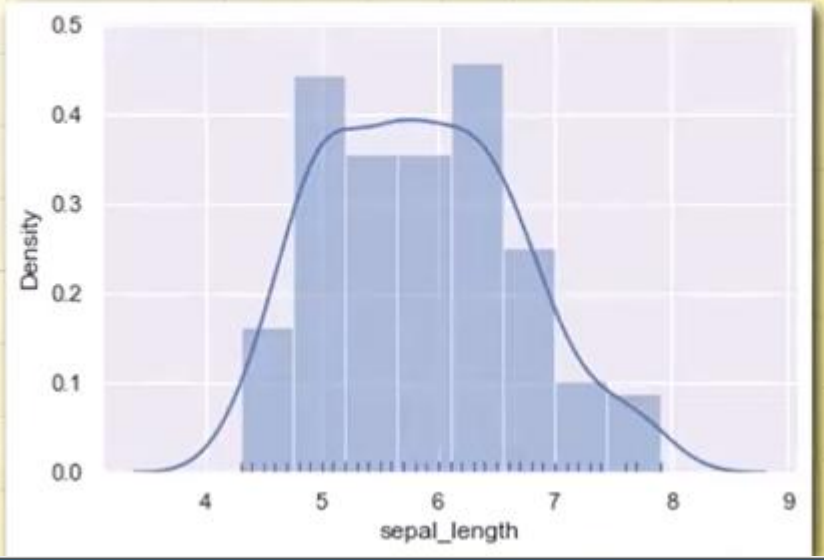
donde

$$IQR = Q3 - Q1$$

- Todo valor **fuera de ese rango** se considera **outlier** (y aparece como puntito o círculo).
- Los bigotes **terminan justo antes del primer valor que se sale del rango**.

Distplot:

```
sns.distplot(iris_data['sepal_length'], hist=True, kde=True, rug=True)
```



HeatMap:

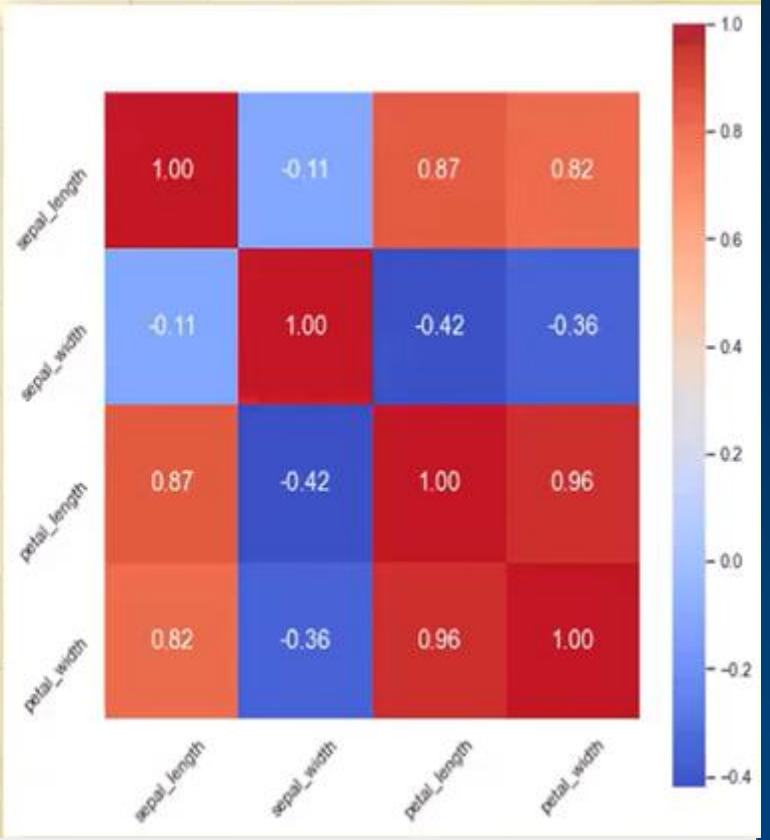
El mapa de calor se define como una representación gráfica de datos que utiliza colores para visualizar el valor de la matriz.

- → Para representar valores más comunes o actividades más altas se utilizan colores más brillantes, básicamente colores rojizos.
- → Para representar valores menos comunes o de poca actividad, se prefieren colores más oscuros.

El mapa de calor también se define por el nombre de la matriz de sombreado.

Los mapas de calor en Seaborn se pueden trazar utilizando la función seaborn.heatmap().

```
corr = iris_data.drop(columns = 'species').corr()  #is used for find corelation
plt.figure(figsize=(8,8))
sns.heatmap(corr, cbar = True,  square = True, annot=True, fmt= '.2f',annot_kws={'size': 15},
            xticklabels= iris_data.drop(columns = 'species').columns,
            yticklabels= iris_data.drop(columns = 'species').columns,
            cmap= 'coolwarm')
plt.xticks(rotation = 45)
plt.yticks(rotation = 45)
plt.show()
```



partes principales

. Ejes (Axes)

- **Eje X y Eje Y** representan las **variables categóricas o índices** de la matriz.
- En un *heatmap de correlación*, por ejemplo, ambos ejes muestran las **variables** del dataset.
- Cada **celda** representa la intersección de un valor en X con un valor en Y.

partes principales

2. Celdas (Cells)

- Cada **celda** es un **valor numérico** representado mediante **color**.
- El **color** indica la magnitud (por ejemplo, una correlación alta puede ser rojo intenso y una baja azul).
- Cuanto más oscuro o más claro el color, **mayor o menor es el valor**.

partes principales

3. Anotaciones (annot)

- Si activa `annot=True`, cada celda muestra el **valor numérico** correspondiente.
    - También se puede ajustar el formato de los números:

4. Barra de color (Colorbar)

- A la derecha del gráfico aparece una **barra de color** que indica la **escala de valores**.
    - Muestra el **rango mínimo y máximo** de los datos mapeados a los colores.
- Puedes personalizarla con `cbar=False` para ocultarla o `cbar_kws` para modificarla.

5. Etiquetas y Título

- Se añaden con las funciones normales de Matplotlib:

Visualizar rendimiento E-commerce

30 MIN

A partir del ejercicio trabajado en la clase anterior, se aplicarán distintas visualizaciones:

- **Líneas:** Evolución de ventas mensuales
- **Barras:** Comparación de eficiencia publicitaria (ventas/gasto)
- **Dispersión:** Relación visitantes vs ventas
- **Heatmap:** Correlaciones entre ventas, visitantes y gasto

Proyecto

Tienda Aurelion

- **Documentación:** notebook Markdown
- **Desarrollo técnico:** programa Python
- **Visualización de datos:** dashboard en Power BI
- **Presentación oral:** problema, solución y hallazgos

guayerd

En colaboración con

IBM **SkillsBuild**

Aplicando visualizaciones

Trabajo en equipo

1. Seleccionar **variables clave** de su dataset
2. Crear al menos **3 visualizaciones**
3. Identificar **patrones, tendencias o correlaciones**
4. **Interpretar resultados** en contexto de su problema
5. **Documentar con Copilot** hallazgos y decisiones

Preparación demo

próxima clase

guayerd

En colaboración con

IBM **SkillsBuild**

Características

2° demo: sincrónica

- **Tema:** python
- **Medio:** oral
- **Devolución:** en vivo
- **Entrega:** archivo .py o Notebook Jupyter
- **Presentación:** Power Point (opcional)
- **Exposición:** Cada equipo debe organizarse
- **Tiempo:** máximo 15 minutos por equipo

Contenido

2° demo: sincrónica

Documentación actualizada (.md)

- Estadísticas descriptivas básicas calculadas
- Identificación del tipo de distribución de variables
- Análisis de correlaciones entre variables principales
- Detección de outliers (valores extremos)
- Al menos 3 gráficos representativos
- Interpretación de resultados orientada al problema

Base de datos (.csv)

- Limpia y lista para análisis

Programa actualizado (.py)

- Información actualizada del proyecto

Retro

¿Cómo nos vamos?

- ¿Qué fue lo más útil de la clase?
- ¿Qué parte te costó más?
- ¿Qué te gustaría repasar o reforzar?

guayerd

En colaboración con

IBM SkillsBuild