



SESIÓN ANÁLISIS VISUAL DE DATOS

CONTENIDOS:

- Características de la librería Seaborn.
 - Importación de la librería.
- Tipos de gráfico.
 - Gráficos de distribución de observaciones: distplot.
 - Gráficos de dispersión y correlación de variables: joinplot., pairplot.
 - Gráficos de regresiones: regplot.
 - Gráficos de variables categóricas: barplot, countplot, boxplot y violinplot.
 - Gráficos de matrices: Heatmap.
- Grillas de gráficos: pairgrid y facetgrid.

CARACTERÍSTICAS DE LA LIBRERÍA SEABORN

La visualización de datos es una parte esencial del análisis exploratorio de datos (EDA). Permite comprender patrones, tendencias y relaciones en los datos de manera intuitiva. Seaborn es una librería de Python basada en Matplotlib que facilita la creación de gráficos estadísticos atractivos y fáciles de interpretar.

Características de la Librería Seaborn

- **Integración con Pandas**
Seaborn trabaja directamente con DataFrames de Pandas, lo que facilita la visualización de datos sin necesidad de transformaciones adicionales.
- **Estética Mejorada**
Proporciona temas de diseño y paletas de colores atractivos por defecto, lo que mejora la presentación de los gráficos sin necesidad de configuraciones avanzadas.
- **Gráficos Estadísticos**
Incluye funciones específicas para visualizar distribuciones, relaciones y datos categóricos, como histogramas, diagramas de dispersión y gráficos de caja (boxplots).

- **Facilidad de Uso**

Su sintaxis es sencilla y permite crear gráficos complejos con pocas líneas de código, optimizando el tiempo de desarrollo.

- **Anotaciones Automáticas**

Algunos gráficos de Seaborn incluyen líneas de regresión o intervalos de confianza sin necesidad de cálculos adicionales.

- **Compatibilidad con Matplotlib**

Seaborn está construido sobre Matplotlib, por lo que podemos combinar ambas librerías y personalizar los gráficos con mayor flexibilidad.

Importación De La Librería

Para utilizar Seaborn, primero debemos instalarla y luego importarla junto con otras librerías complementarias, como Matplotlib y Pandas.

Instalación

Si aún no tienes Seaborn instalado, puedes hacerlo con el siguiente comando:

```
pip install seaborn
```

Importación en Python

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

Una vez importadas, podemos comenzar a crear gráficos de manera sencilla utilizando los datos de Pandas.

TIPOS DE GRÁFICO

Seaborn ofrece una amplia variedad de gráficos para diferentes propósitos en análisis exploratorio de datos (EDA). A continuación, exploraremos los más comunes y su aplicación práctica.

1. Gráficos de Distribución de Observaciones: Distplot.

Los gráficos de distribución son una herramienta fundamental en el análisis exploratorio de datos (EDA) ya que permiten visualizar la forma en que los valores de una variable numérica están distribuidos.

distplot() en Seaborn

⚠️ Nota: La función `distplot()` ha sido depreciada en Seaborn y ha sido reemplazada por `histplot()` para histogramas y `kdeplot()` para la densidad de distribución.

¿Qué es distplot()?

El gráfico de distribución (`distplot()`) se utilizaba en Seaborn para mostrar la distribución de una variable numérica. Este gráfico combinaba un histograma con una curva de densidad (KDE, Kernel Density Estimation).

Propósito de distplot()

- Visualizar la distribución de una variable numérica.
- Identificar la forma de la distribución (simétrica, sesgada, bimodal, etc.).
- Evaluar la concentración de datos en ciertos rangos.
- Detectar valores atípicos (outliers).

Ejemplo de distplot() en Python (Obsoleto)

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

# Datos de ejemplo (Distribución Normal)
np.random.seed(42)
data = np.random.randn(1000)

# Gráfico de distribución
sns.distplot(data, kde=True, bins=30, color="blue")
plt.title("Distribución de Datos con KDE")
plt.show()
```

Salida:

- Un histograma con barras que representan la frecuencia de los datos.
- Una curva KDE que muestra la densidad de los valores.

Gráficos de Distribución de Observaciones en Seaborn: histplot() y kdeplot()

Los gráficos de distribución son una herramienta fundamental en el análisis exploratorio de datos (EDA) ya que permiten visualizar la forma en que los valores de una variable numérica están distribuidos.

¿Para qué se usan los gráficos de distribución?

Propósito:

- Examinar la distribución de los datos.
- Identificar asimetrías, sesgos o valores atípicos (outliers).
- Comparar distribuciones entre diferentes subconjuntos de datos.
- Evaluar la forma y dispersión de la variable.

histplot() – Histograma con opción de KDE

histplot() es la función más reciente y recomendada en Seaborn para visualizar histogramas, reemplazando al antiguo distplot().

Ejemplo en Python:

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

# Datos simulados (Distribución Normal)
data = np.random.randn(1000)

# Gráfico de histograma con KDE activado
sns.histplot(data, kde=True, color="blue", bins=30)
plt.title("Distribución de Datos con KDE")
plt.xlabel("Valores")
plt.ylabel("Frecuencia")
plt.show()
```

Salida:

- Un histograma con 30 bins (barras) que representa la frecuencia de los datos.
- Una curva KDE (Kernel Density Estimation) opcional que suaviza la distribución.

Parámetros clave de histplot()

Parámetro	Descripción	Ejemplo
data	Datos a graficar (array, lista o Pandas DataFrame).	data=np.random.randn(1000)
bins	Número de barras del histograma.	bins=20
kde	Si es True, añade una curva de densidad.	kde=True
color	Color del histograma.	color="green"
element	Tipo de representación (bars, step, poly).	element="step"
multiple	Permite superponer distribuciones (stack, dodge).	multiple="stack"

kdeplot() – Estimación de Densidad de Kernel (KDE)

kdeplot() se usa cuando solo queremos la curva de densidad sin el histograma.

Ejemplo en Python:

```
sns.kdeplot(data, shade=True, color="red")
plt.title("Distribución de Datos con KDE")
plt.xlabel("Valores")
plt.ylabel("Densidad")
plt.show()
```

Salida:

- Una curva KDE que estima la densidad de probabilidad de la variable.
- Sombra bajo la curva para visualizar mejor la distribución (shade=True).

Comparación de múltiples distribuciones con histplot() y kdeplot()

Podemos comparar la distribución de múltiples conjuntos de datos dentro de un mismo gráfico.

Ejemplo en Python:

```
# Datos simulados (Dos distribuciones normales)
data1 = np.random.randn(1000) * 1.5 + 2
data2 = np.random.randn(1000) * 1.0 + 5

# Histograma con comparación de distribuciones
sns.histplot(data1, kde=True, color="blue", label="Grupo 1", bins=30)
sns.histplot(data2, kde=True, color="red", label="Grupo 2", bins=30)
plt.title("Comparación de Distribuciones")
plt.legend()
plt.show()
```

Salida:

- Dos histogramas superpuestos, cada uno con su respectiva curva KDE.
- Permite analizar diferencias en la distribución entre los dos grupos.

¿Cuándo usar histplot() y kdeplot()?

Método	Cuándo usarlo
histplot()	Para analizar frecuencia y distribución de datos discretos o continuos.
kdeplot()	Cuando queremos una visualización más suave de la distribución sin histograma.
Ambos combinados	Para obtener una visión más detallada y comparar distribuciones.

2. Gráficos de Dispersion y Correlación de Variables: Joinplot, Pairplot.

Los gráficos de dispersión son herramientas fundamentales en el análisis exploratorio de datos (EDA) para visualizar la relación entre dos o más variables. Seaborn ofrece jointplot() y pairplot() para analizar correlaciones y patrones en los datos.

jointplot(): Relación entre dos variables con gráficos marginales

El jointplot() combina un gráfico de dispersión con histogramas marginales, permitiendo observar la relación entre dos variables y la distribución individual de cada una.

Propósito:

- Analizar la correlación entre dos variables numéricas.
- Detectar patrones de dispersión.
- Identificar valores atípicos (outliers).
- Visualizar la distribución marginal de cada variable.

Ejemplo en Python:

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Datos de ejemplo
np.random.seed(42)
data = pd.DataFrame({
    "Ingresos": np.random.normal(50000, 10000, 200),
    "Gastos": np.random.normal(30000, 8000, 200)
})

# Crear el jointplot
sns.jointplot(x="Ingresos", y="Gastos", data=data, kind="scatter", color="blue")
plt.show()
```

Salida:

- Un gráfico de dispersión en el centro.
- Histogramas en los ejes X e Y para mostrar la distribución de cada variable.

Variantes de jointplot()

- kind="reg": Agrega una línea de regresión.
- kind="hex": Usa hexágonos en lugar de puntos (útil para grandes volúmenes de datos).
- kind="kde": Usa estimación de densidad en lugar de histogramas.

pairplot(): Matriz de gráficos de dispersión

El pairplot() genera una matriz de gráficos de dispersión para múltiples variables, permitiendo analizar visualmente todas las correlaciones en un conjunto de datos.

Propósito:

- Comparar relaciones entre múltiples variables en un solo gráfico.
- Identificar correlaciones o patrones entre varias características.
- Explorar distribuciones de variables individuales en los histogramas diagonales.

Ejemplo en Python:

```
# Crear un pairplot
sns.pairplot(data)
plt.show()
```

Salida:

- Una matriz de gráficos de dispersión para cada par de variables.
- Histogramas en la diagonal, mostrando la distribución de cada variable.

Opciones de pairplot()

- hue="Categoría": Colorea los puntos según una variable categórica.
- diag_kind="kde": Usa curvas KDE en lugar de histogramas en la diagonal.

3. Gráficos de Regresiones: Regplot.

Los gráficos de regresión son esenciales para analizar visualmente las relaciones lineales entre dos variables. Seaborn ofrece el gráfico regplot(), que no solo muestra los puntos de los datos (dispersión), sino que también ajusta y visualiza una línea de regresión para observar tendencias.

regplot(): Relación lineal entre dos variables

El regplot() es ideal para visualizar la relación lineal entre dos variables numéricas y observar cómo una variable predice o influye en otra mediante una línea de regresión.

Propósito:

- Analizar la relación lineal entre una variable independiente y una variable dependiente.
- Visualizar una línea de tendencia ajustada sobre los puntos de datos.
- Detectar posibles patrones o anomalías en los datos.
- Probar la existencia de una relación lineal entre las variables.

Ejemplo en Python

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Datos de ejemplo
np.random.seed(42)
data = pd.DataFrame({
    "Edad": np.random.randint(18, 65, 100),
    "Ingresos": np.random.normal(50000, 12000, 100)
})

# Crear el gráfico de regresión
sns.regplot(x="Edad", y="Ingresos", data=data, color="green")
plt.title("Relación Lineal entre Edad e Ingresos")
plt.show()
```



Salida:

- Un gráfico de dispersión que muestra los puntos de los datos.
- Una línea de regresión que refleja la relación lineal entre las variables. La pendiente de esta línea indica el impacto de la variable independiente sobre la dependiente.

Variantes de `regplot()`

- `ci=None`: Elimina el intervalo de confianza de la línea de regresión, mostrando solo la línea ajustada.
- `order=n`: Ajusta una regresión polinomial de orden n en lugar de una recta.
- `x_jitter=0.1`: Agrega un pequeño desplazamiento en los datos para evitar la superposición de puntos.
- `logx=True`: Aplica una escala logarítmica a los valores del eje X.

4. Gráficos de Variables Categóricas: Barplot, Countplot, Boxplot Y Violinplot.

Los gráficos de variables categóricas permiten analizar cómo una variable numérica varía en función de una variable categórica. Seaborn ofrece diferentes opciones para visualizar estos datos de manera clara y efectiva.

barplot(): Promedio de una variable numérica por categorías

El `barplot()` es útil para comparar la media de una variable numérica en diferentes categorías, proporcionando una visión general del comportamiento de los datos.

Propósito:

- Mostrar la media de una variable numérica agrupada por una variable categórica.
- Incluir barras de error para representar la dispersión de los datos.
- Comparar diferencias entre grupos de manera visual.

Ejemplo en Python:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Datos de ejemplo
import pandas as pd
data = pd.DataFrame({
    "Categoría": ["A", "B", "C", "A", "B", "C", "A", "B", "C"],
    "Valor": [10, 15, 7, 12, 18, 9, 14, 20, 11]
})

# Crear el barplot
sns.barplot(x="Categoría", y="Valor", data=data, palette="Blues")
plt.title("Promedio de Valor por Categoría")
plt.show()
```

Salida:

- Un gráfico de barras donde cada barra representa la media de los valores en cada categoría.
- Se incluyen barras de error para mostrar la variabilidad en cada grupo.

countplot(): Conteo de observaciones por categoría

El countplot() es ideal para visualizar la frecuencia de cada categoría dentro de una variable categórica.

Propósito:

- Mostrar la cantidad de observaciones en cada categoría.
- Comparar la distribución de frecuencias en distintos grupos.

Ejemplo en Python:

```
sns.countplot(x="Categoría", data=data, palette="pastel")
plt.title("Distribución de Observaciones por Categoría")
plt.show()
```

Salida:

- Un gráfico de barras donde la altura de cada barra representa la cantidad de observaciones en esa categoría.

boxplot(): Distribución de una variable numérica por categorías

El boxplot() (gráfico de caja y bigotes) es excelente para analizar la distribución, asimetría y outliers en los datos.

Propósito:

- Visualizar la distribución de una variable numérica en diferentes categorías.
- Identificar la mediana, rango intercuartil (IQR) y valores atípicos.

Ejemplo en Python:

```
sns.boxplot(x="Categoría", y="Valor", data=data, palette="Set2")
plt.title("Distribución de Valores por Categoría")
plt.show()
```

Salida:

Un diagrama de caja, donde:

- La línea central representa la mediana.
- Los bordes de la caja muestran el rango intercuartil (IQR).
- Los bigotes indican el rango de valores sin outliers.
- Los puntos fuera de los bigotes son outliers.

violinplot(): Combinación de boxplot() y estimación de densidad

El violinplot() es una variante del boxplot que incorpora una estimación de densidad para visualizar mejor la distribución de los datos.

Propósito:

- Combinar la información del boxplot con una curva de densidad de probabilidad.
- Identificar la forma de la distribución en cada categoría.
- Comparar la dispersión de los valores en diferentes grupos.

Ejemplo en Python:

```
sns.violinplot(x="Categoría", y="Valor", data=data, palette="muted")
plt.title("Distribución y Densidad de Valores por Categoría")
plt.show()
```

Salida:

Un gráfico de violín, donde:

- La caja en el centro representa la distribución central de los datos (similar a un boxplot).
- La forma del violín muestra la densidad de los datos en diferentes valores.
- Los extremos del violín indican la dispersión de los datos.

5. Gráficos de Matrices: Heatmap

Los gráficos de matrices permiten visualizar datos tabulares de manera estructurada utilizando colores para representar valores. Son especialmente útiles en análisis exploratorio de datos (EDA) para detectar patrones, tendencias y correlaciones en grandes volúmenes de información.

heatmap(): Visualización de matrices de datos con colores

El heatmap() de Seaborn se usa para representar matrices de valores numéricos mediante una escala de colores, facilitando la interpretación visual de los datos.

Propósito:

- Representar matrices de correlación entre variables numéricas.
- Identificar patrones y relaciones en grandes volúmenes de datos.
- Resaltar valores altos y bajos mediante una escala de colores.
- Facilitar la detección de outliers o agrupaciones de datos.

Ejemplo en Python:

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# Crear un DataFrame con datos aleatorios
np.random.seed(42)
data = pd.DataFrame(np.random.rand(6, 6),
                    columns=["A", "B", "C", "D", "E", "F"])

# Crear el heatmap
sns.heatmap(data, cmap="coolwarm", annot=True, fmt=".2f", linewidths=0.5)
plt.title("Mapa de Calor de Datos Aleatorios")
plt.show()
```

Salida:

Un mapa de calor, donde:

- Los colores representan los valores en la matriz.
- La barra lateral muestra la escala de colores utilizada.
- Los números dentro de cada celda indican el valor específico (cuando annot=True).
- Se pueden agregar bordes entre celdas con linewidths.

Uso en Matrices de Correlación

Una de las aplicaciones más comunes del heatmap() es visualizar matrices de correlación, que muestran la relación entre variables numéricas en un conjunto de datos.

Ejemplo en Python:

```
# Crear un DataFrame con datos simulados
df = pd.DataFrame({
    "Ventas": [200, 220, 250, 270, 300, 320],
    "Publicidad": [20, 25, 28, 30, 35, 40],
    "Satisfacción": [3.5, 4.0, 4.2, 4.5, 4.8, 5.0]
})

# Calcular la matriz de correlación
correlation_matrix = df.corr()

# Crear el heatmap de la matriz de correlación
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Mapa de Calor de Correlaciones")
plt.show()
```

Salida:

Un mapa de calor de correlaciones, donde:

- Los valores cercanos a 1 indican una correlación positiva fuerte.
- Los valores cercanos a -1 indican una correlación negativa fuerte.
- Los valores cercanos a 0 indican ausencia de correlación.
- Los colores ayudan a visualizar las relaciones entre las variables.

Personalización del heatmap()

El heatmap() de Seaborn permite múltiples opciones para mejorar la visualización:

- `cmap="viridis"`: Cambia la paleta de colores.
- `annot=True`: Muestra los valores dentro de cada celda.
- `fmt=".2f"`: Controla el formato de los números en las celdas.
- `linewidths=0.5`: Agrega bordes entre las celdas.
- `vmin=-1, vmax=1`: Define el rango de valores en la escala de colores.

GRILLAS DE GRÁFICOS: PAIRGRID Y FACETGRID

Las grillas de gráficos permiten visualizar múltiples relaciones dentro de un conjunto de datos de manera organizada. Estas herramientas de Seaborn son útiles en el análisis exploratorio de datos (EDA), ya que facilitan la comparación de patrones y distribuciones a lo largo de diferentes variables y categorías.

PairGrid: Creación de una Grilla Personalizada de Gráficos

PairGrid permite construir una grilla de gráficos donde cada celda representa una combinación entre dos variables. A diferencia de pairplot(), PairGrid ofrece más flexibilidad para personalizar los tipos de gráficos en cada sección de la grilla.

Propósito:

- Visualizar relaciones entre múltiples variables numéricas.
- Personalizar los gráficos en la diagonal y fuera de la diagonal.
- Facilitar la exploración de correlaciones y distribuciones en un conjunto de datos.

Ejemplo en Python:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Cargar dataset de ejemplo
df = sns.load_dataset("iris")

# Crear la grilla con PairGrid
g = sns.PairGrid(df, hue="species")

# Especificar los tipos de gráficos
g.map_diag(sns.histplot) # Histogramas en la diagonal
g.map_offdiag(sns.scatterplot) # Diagramas de dispersión fuera de la diagonal
g.add_legend()

plt.show()
```

Salida:

- Una grilla de gráficos con histogramas en la diagonal y gráficos de dispersión fuera de la diagonal.
- Cada color representa una categoría de la variable "species".
- Se puede personalizar para usar otras combinaciones, como sns.kdeplot() o sns.regplot() en lugar de scatterplot().

FacetGrid: Creación de Grillas Basadas en Categorías

FacetGrid permite dividir un conjunto de datos en subconjuntos basados en una variable categórica y graficar cada subconjunto de manera separada.

Propósito:

- Comparar distribuciones o tendencias a través de diferentes categorías.
- Facilitar la visualización de datos segmentados.
- Mejorar la interpretación de patrones al dividir los datos en subgrupos.

Ejemplo en Python:

```
# Crear la grilla con FacetGrid
g = sns.FacetGrid(df, col="species")

# Aplicar un histograma a cada categoría
g.map(sns.histplot, "sepal_length", bins=10, color="blue")

plt.show()
```

Salida:

- Una grilla de histogramas, donde cada panel representa una categoría de la variable "species".
- Permite visualizar cómo varía la distribución de "sepal_length" en cada grupo.

Comparación entre PairGrid y FacetGrid

Característica	PairGrid	FacetGrid
<i>Tipo de datos</i>	Numéricos	Numéricos o categóricos
<i>Relación entre variables</i>	Muestra combinaciones entre múltiples variables	Separa los gráficos por categorías
<i>Personalización</i>	Muy flexible (distintos tipos de gráficos en diagonal y fuera de la diagonal)	Se centra en un solo tipo de gráfico por categoría
<i>Uso recomendado</i>	Análisis de correlaciones y tendencias entre varias variables numéricas	Comparaciones dentro de subgrupos categóricos

ACTIVIDAD PRÁCTICA GUIADA: GENERAR UN HEATMAP PARA VISUALIZAR LA CORRELACIÓN ENTRE VARIABLES

En este ejercicio práctico, aprenderás a generar un heatmap utilizando la librería Seaborn en Python. Un heatmap es una representación gráfica de datos donde los valores individuales contenidos en una matriz se representan con colores. En este caso, lo usaremos para visualizar la correlación entre variables en un conjunto de datos.

Paso 1: Importar Librerías

Primero, importamos las librerías necesarias:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

Paso 2: Crear un Conjunto de Datos

Vamos a crear un conjunto de datos simple con varias variables numéricas. Este conjunto de datos simulará información sobre personas, incluyendo su edad, ingresos, años de educación y horas de sueño.

```
# Crear un DataFrame con datos de ejemplo
df = pd.DataFrame({
    'Edad': [25, 30, 35, 40, 45, 50, 55, 60, 65, 70],
    'Ingresos': [50000, 60000, 70000, 80000, 90000, 100000, 110000, 120000, 130000, 140000],
    'Años_Educación': [12, 14, 16, 18, 20, 22, 24, 26, 28, 30],
    'Horas_Sueño': [7, 6.5, 7.5, 8, 7, 6, 7, 8, 7.5, 6.5]
})
```

Paso 3: Calcular la Matriz de Correlación

La matriz de correlación es una tabla que muestra los coeficientes de correlación entre las variables.

El coeficiente de correlación varía entre -1 y 1, donde:

- 1: Correlación positiva perfecta.
- -1: Correlación negativa perfecta.
- 0: No hay correlación.

```
# Calcular la matriz de correlación
correlacion = df.corr()
print(correlacion)
```

Salida:

	Edad	Ingresos	Años_Educación	Horas_Sueño
Edad	1.000000	0.997775	0.997775	-0.104828
Ingresos	0.997775	1.000000	1.000000	-0.104828
Años_Educación	0.997775	1.000000	1.000000	-0.104828
Horas_Sueño	-0.104828	-0.104828	-0.104828	1.000000

Paso 4: Generar el Heatmap

Ahora, usaremos Seaborn para generar un heatmap que visualice la matriz de correlación.

```
# Crear el heatmap
sns.heatmap(correlacion, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Matriz de Correlación')
plt.show()
```

Explicación de Parámetros:

- `annot=True`: Muestra los valores de correlación en cada celda.
- `cmap='coolwarm'`: Define la paleta de colores (rojo para correlaciones negativas, azul para positivas).
- `fmt='.2f'`: Formatea los números a dos decimales.

Paso 5: Interpretación del Heatmap

El heatmap generado mostrará una matriz de colores con los valores de correlación entre las variables. Aquí está cómo interpretarlo:

Ejes:

El eje horizontal y vertical representan las variables del conjunto de datos (Edad, Ingresos, Años_Educación, Horas_Sueño).

Colores:

- Azul oscuro: Correlación positiva fuerte (cercana a 1).
- Rojo oscuro: Correlación negativa fuerte (cercana a -1).
- Blanco o colores claros: Correlación cercana a 0.

Valores:

Cada celda muestra el valor de correlación entre dos variables.

Interpretación de los Resultados:

- Edad vs Ingresos: 0.99 (correlación positiva fuerte).
- Edad vs Años_Educación: 0.99 (correlación positiva fuerte).
- Ingresos vs Años_Educación: 1.00 (correlación positiva perfecta).
- Horas_Sueño vs Otras Variables: -0.10 (correlación negativa débil).

Paso 6: Personalización del Heatmap

Puedes personalizar el heatmap para mejorar su claridad y estética. Por ejemplo:

```
# Personalizar el heatmap
plt.figure(figsize=(8, 6)) # Tamaño del gráfico
sns.heatmap(corrrelacion, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5, linecolor='black')
plt.title('Matriz de Correlación', fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```

Mejoras:

- figsize=(8, 6): Ajusta el tamaño del gráfico.
- linewidths=0.5: Añade líneas entre las celdas.
- linecolor='black': Define el color de las líneas.
- fontsize: Ajusta el tamaño de las etiquetas.

Conclusión del Ejercicio

- Aprendiste a generar un heatmap para visualizar la correlación entre variables en un conjunto de datos.
- El heatmap es una herramienta poderosa para identificar relaciones fuertes o débiles entre variables.
- Personalizar el heatmap mejora su claridad y facilita la interpretación de los resultados.