

# 1.5 Resumen

- NumPy es una librería especializada para el calculo numérico.
- Utiliza los `array` como estructura de datos.
- Solo permite datos de un mismo tipo. Ya sea: Bool, Int, Float, str ...
- Puede tener múltiples dimensiones:
  - 1D
  - 2D
  - 3D
  - N-D
- Podemos crear *arrays* utilizando listas.
- Para operar entre *arrays* necesitamos que ambos contengan los mismos tipos de datos.
  - En sumas y restas han de tener las mismas dimensiones
  - En multiplicación y division se debe cumplir la regla de multiplicación de matrices.
- Podemos realizar operaciones entre *arrays* y escalares (siendo un escalar un número)
- Creación de *arrays*:
  - Podemos tener *arrays* de diferentes dimensiones:
    - Una dimensión : `np.array(lista)`
    - Dos dimensiones : `np.array(lista1, lista)`
    - Tres dimensiones: `np.array(lista1(lista), lista2(lista))`
- Métodos de *arrays*:
  - `.shape` : Nos devuelve la forma del *array*.
    - En un *array* bidimensional nos devuelve: numero filas X numero de columnas
    - En un *array* multidimensionales: numero de matrices X numero de filas X numero de columnas
  - `.size` : Nos devuelve el tamaño del *array*, que es el número de elementos.
  - `.ndim` : Nos devuelve el número de dimensiones del *array*.

- `.dtype`: Nos devuelve el tipo de dato que contiene nuestro *array*.
- Métodos de creación de *arrays*:
  - `np.random.randint()`: nos genera un array de X dimensiones con números aleatorios ENTEROS
 

```
np.random.randint(1, 20, (2,3,4))
```
  - `np.random.rand()`: nos genera un array de X dimensiones con números aleatorios entre 0-1. PARA ESPECIFICAR LA FORMA Y DIMENSIONES LO PONEMOS **SIN PARENTESIS**

```
array2 = np.random.rand(2,4)
```
  - `np.random.random_sample()`: nos genera un array de X dimensiones con números aleatorios entre 0-1. PARA ESPECIFICAR LA FORMA Y DIMENSIONES LO PONEMOS **CON PARENTESIS**

```
array = np.random.random_sample((3,4,5))
```
  - `np.ones()`: nos genera un array solo de 1.
 

```
unos = np.ones((2,3), dtype = int)
# el parametro dtype nos permite "castear" el tipo de dato a int
```
  - `np.zeros()`: nos genera un array solo de 0.
 

```
ceros = np.zeros((2,3,4), dtype = int)
```
  - `np.empty()`: nos genera un array vacío con la forma del array que le pasemos.
 

```
vacio = np.empty((2,3))
```
  - `np.array()`: nos genera arrays a partir de una lista, o de una lista de listas.
  - `np.arange()`: SOLO CREA ARRAYS UNIDIMENSIONAL.
- Podemos hacer operaciones en NumPy:
  - Podemos sumar con: `+` o `np.add()`
  - Restar con: `-`, `np.subtract()`
  - Multiplicar con: `*`, `np.multiply()`
  - Dividir con: `/`, `np.divide()`
  - 🌟 Podemos hacer todas las operaciones que queramos!!!!
  - Podemos hacer operaciones con escalares, números enteros o decimales.
-

- La Indexación en arrays es como en listas, siempre empieza en cero y siempre debe de ir entre corchetes.
- Indexación en función de las dimensiones
  - Unidimensional: `array[i]` , donde [i]= columna
  - Bidimensional: `array [i,j] = array[i][j]` , donde [i] = fila, [j]= columna
  - Multidimensional: `array[i,j,k] = array[i][j][k]` , donde [i] = array, [j] = fila, [k] = columna
- Indexación basado en rangos:
  - Unidimensional: `array[inicio:fin:salto]`
  - Bidimensional :  
`array[inicio:fin:salto,inicio:fin:salto ] = array[inicio:fin:salto][inicio:fin:salto ]`
  - Multidimensional:  
`array[inicio:fin:salto,inicio:fin:salto,inicio:fin:salto ] = array[inicio:fin:salto][inicio:fin:salto ][inicio:fin:salto ]`
- Podemos filtrar aplicando operadores `<` , `>` , `>=` , `<=` , `==` .
  - `array[array (operador) condicion]`  
 — : Eg `array[array > condicion]`
  - Si queremos meter más de una condición: `&` o `|`  
 — `array[array (operador) condicion1 anidamiento array (operador) condicion2]`  
 ■ : Eg `array[(array < 2) | (array > 5)]`
- Métodos:
  - `np.transpose()` : invierte los ejes de nuestra matriz o si se le especifica el orden, nos permuta los ejes de la matriz.
  - `np.swapaxes()` : intercambia dos ejes de la matriz, indicando sobre que ejes queremos realizar la operación
  - `np.reshape()` : cambia la forma de la matriz a la forma especificada.
  - `np.copy()` : realiza una copia completa de un array NumPy, incluyendo sus datos y su estructura.
  - `np.flatten()` : se utiliza para convertir un array multidimensional en un array unidimensional, es decir, aplanar el array.

# 2.5 Resumen

## 1. Introducción a `pandas`:

- **Importación de la biblioteca `pandas`:**

```
import pandas as pd
```

Importa `pandas`, proporcionando acceso a sus funciones y estructuras de datos bajo el alias `pd`.

- **Creación de objetos `Series` en `pandas`:**

- **Serie vacía:**

```
serie_vacia = pd.Series()
```

Crea una serie sin elementos, útil como punto de partida para operaciones que acumulan datos dinámicamente.

- **Serie a partir de una lista:**

```
lista = [23, 45, 17, 83, 67]  
serie_lista1 = pd.Series(lista)
```

Convierte una lista de valores en una serie con índices numéricos automáticos desde cero.

- **Serie con índices personalizados:**

```
serie_lista3 = pd.Series(lista, index=["Lunes", "Martes", "Miércoles"])
```

Asigna cada número de la lista a un día específico, facilitando la referencia por etiqueta.

## 2. Selección de Datos en DataFrames:

- **Uso de los métodos `loc` y `iloc`:**

- `loc`:

```
valor = df.loc["Martes", "Humedad"]
```

Selecciona datos por nombre de fila y columna, ideal para dataframes con índices o columnas etiquetados.

- `iloc`:

```
valor = df.iloc[1, 2]
```

Selecciona datos por posición numérica, similar a la indexación en matrices o arrays.

- **Diferencias entre `loc` e `iloc`:**

- `loc` incluye el extremo final en rangos, mientras que `iloc` no:

```
df.loc[0:2] # Incluye las filas 0, 1, y 2
df.iloc[0:2] # Incluye las filas 0 y 1, pero no la 2
```

### 3. Manipulación de DataFrames:

- **Creación de nuevas columnas:**

```
df['nueva_columna'] = df['columna_existente'] * 2
```

Genera una nueva columna calculando valores a partir de los datos existentes, duplicando, por ejemplo, el valor de una columna existente.

- **Uso de condiciones para filtrar datos:**

```
df_filtrado = df[(df['precio'] > 10) & (df['ventas'] > 5000)]
```

Extrae un subconjunto del DataFrame original, seleccionando filas que cumplen con condiciones específicas de precio y volumen de ventas.

## 3.4 Resumen

Tomando a "data" como el nombre de nuestro DataFrame tendremos los siguientes metodos.

| Método                       | Descripción  | Ejemplo  |
|------------------------------|--|--|
| <code>pd.read_csv()</code>   | Cargar datos desde un archivo CSV en un DataFrame.     | <pre>import pandas as pd data = pd.read_csv('archivo.csv')</pre>     |
| <code>.head()</code>         | Muestra las primeras filas de un DataFrame.            | <code>data.head()</code>   |
| <code>.tail()</code>         | Muestra las últimas filas de un DataFrame.             | <code>data.tail()</code>   |
| <code>.sample()</code>       | Devuelve filas aleatorias de un DataFrame.             | <code>data.sample(3)</code>  |
| <code>.shape</code>          | Devuelve el número de filas y columnas.                | <code>data.shape</code>  |
| <code>.columns</code>        | Muestra los nombres de las columnas.                   | <code>data.columns</code>  |
| <code>unique()</code>        | Devuelve valores únicos en una columna.                | <code>data['columna'].unique()</code>                                |
| <code>value_counts()</code>  | Cuenta la frecuencia de valores únicos en una columna. | <code>data['columna'].value_counts()</code>                          |
| <code>select_dtypes()</code> | Selecciona columnas por tipo de datos.                 | <code>data.select_dtypes(include=['int'])</code>                     |
| <code>drop()</code>          | Elimina filas o columnas de un DataFrame.              | <code>data.drop(columns=['columna_a_eliminar'], inplace=True)</code> |
| <code>.info()</code>         | Muestra información sobre el DataFrame.                | <code>data.info()</code>   |

| Método                           | Descripción  | Ejemplo  |
|----------------------------------|--|--|
| <code>.isnull().sum()</code>     | Calcula la cantidad de valores nulos en cada columna.    | <code>data.isnull().sum()</code>   |
| <code>.notnull().sum()</code>    | Calcula la cantidad de valores no nulos en cada columna. | <code>data.notnull().sum()</code>  |
| <code>.duplicated().sum()</code> | Calcula la cantidad de filas duplicadas en el DataFrame. | <code>data.duplicated().sum()</code>   |
| <code>describe()</code>          | Proporciona estadísticas descriptivas del DataFrame.     | <code>data.describe()</code> o<br><code>df.describe(include = "object")</code> |

## Ejercicios

1- Abrid el fichero de rating-and-performance con el método que sea más correcto (este archivo deberías haberlo descargado en el tema anterior).

2- Mostrad las 10 últimas filas.

3- Mostrad 5 filas aleatorias del DataFrame.

4- ¿Cuántas filas y columnas tenemos en el DataFrame?

5- ¿Cuáles son los nombres de las columnas del DataFrame?

6- Seleccionad solo las columnas de tipo categórico del DataFrame. Después, mostrad los valores únicos y sus frecuencias para cada una de las columnas seleccionadas.

7- En el DataFrame tenemos dos columnas que parece que nos dan la misma información "title" y "title\_orig". Eliminad "title".

8- ¿En que columnas tenemos valores nulos? Utilizad el método que más os guste.

9- ¿Tenemos valores duplicados en el DataFrame?



## 4.4 Resumen


Resumen de los métodos aprendidos.

| Método                    | Descripción   | Ejemplo en Python  |
|---------------------------|---|--|
| <code>concat</code>       | Combina DataFrames a lo largo de un eje específico. | <pre>result =<br/>pd.concat([df1, df2])</pre>                              |
| <code>merge</code>        | Combina DataFrames utilizando columnas comunes.     | <pre>result = pd.merge(df1,<br/>df2, on='clave')</pre>                     |
| <code>join</code>         | Combina DataFrames utilizando índices o columnas.   | <pre>result =<br/>df1.join(df2, on='clave')</pre>                          |
| <code>rename</code>       | Cambia los nombres de columnas o índices.           | <pre>df.rename(columns=<br/>{'anterior':<br/>'nuevo'}, inplace=True)</pre> |
| <code>set_index</code>    | Establece una columna como índice del DataFrame.    | <pre>df.set_index('columna')</pre>   |
| <code>drop</code>         | Elimina filas o columnas del DataFrame.             | <pre>df.drop(columns=<br/>['columna'], inplace=True)</pre>                 |
| <code>str.lower</code>    | Convierte el contenido de una columna a minúsculas. | <pre>df['columna'].str.lower()</pre>                                       |
| <code>str.split</code>    | Divide una cadena en columnas separadas.            | <pre>df['columna'].str.split(',')</pre>                                    |
| <code>str.replace</code>  | Reemplaza una cadena o patrón en una columna.       | <pre>df['columna'].str.replace('viejo', 'nuevo')</pre>                     |
| <code>isin</code>         | Comprueba si los elementos están en una lista.      | <pre>df['columna'].isin(['valor1',<br/>'valor2'])</pre>                    |
| <code>between</code>      | Filtra filas basadas en valores en un rango.        | <pre>df[df['columna'].between(5,<br/>10)]</pre>                            |
| <code>str.contains</code> | Comprueba si una columna contiene un patrón.        | <pre>df['columna'].str.contains('patrón')</pre>                            |



# Unión de Datos

- Para unir todo en un único `csv` podemos usar
  - `pd.concat()`: es el más fácil, los nombres de las columnas se tienen que llamar igual. Los ejes:
    - `axis = 0` filas DEBAJO
    - `axis = 1` filas a la DERECHA
  - `pd.merge()`: es el más el común para añadir nuevas columnas. Podemos incluir:
    - `how`: especificamos como unimos los dataframes, puede ser inner, right, left...
    - `on` y `right_on/left_on`:
      - `on`: si los nombres de las columnas por las que queremos unir son iguales
      - `right_on/left_on`: si los nombres de las columnas por las que queremos unir son distintas
  - `pd.join()`: es como un merge, pero UNA DE LAS COLUMNAS POR LAS QUE QUEREMOS UNIR TIENE QUE SER UN ÍNDICE.
  -  `rsuffix` y `lsuffix`: cuando tenemos columnas que se llaman igual, pero no son las columnas que usaremos para unir los dataframes, podemos usar estos parámetros para especificar el nombre "nuevo".
  - `.reset_index()`: cuando unimos tablas que tienen los mismos índices es aconsejable hacer un `reset_index()` para que nuestros vayan del 0-n sin que se repita ninguno. 

 **NOTA** Para el concat y el `axis = 1`, cuidado que une con índices.

# Filtrado de datos

- Formas de filtrar datos
  - Filtrar por condición
    - `df["Column"] == "Nevada"` : una serie de True y False

- `df[df["Column"] == "Nevada"]` : filtro mi DataFrame por las filas que sean True

- Combinar condiciones con & (and), | (or), ~ (not)

- `(df["Provincia"] == "Madrid") | (df["Provincia"] == "Barcelona")` :  
Una Serie de True y False

- `df[ (df["Provincia"] == "Madrid") | (df["Provincia"] == "Barcelona") ]`

: un DataFrame filtrado

- `df[ ~(df["Provincia"] == "Madrid") | (df["Provincia"] == "Barcelona")) ]`

: filtrado al revés

- Podemos guardar en variables los filtros:

- `filtro =`  
`(df["Provincia"] == "Madrid") | (df["Provincia"] == "Barcelona")`

- `filtro_inverso = df[~filtro]`

- `df["Column"].isin(values)`

- `df["Provincia"].isin(["Madrid", "Barcelona"])`

- `str.contains()` : filtrar con un patron de regex

- Guardar el dataset filtrado

Tenemos que hacer la asignación de variable a un nuevo DataFrame

- `df_nevada = df2[ df2["Column"] == "Nevada" ]`

# 5.4 Resumen

## Resumen de lo aprendido en el `groupby`

| Método<br><code>groupby</code> | Descripción                                    | Ejemplo en Python   |
|--------------------------------|--|---|
| <code>.groupby()</code>        | Agrupar un DataFrame por columnas específicas. | <code>grupos = df.groupby('columna')</code>                               |
| <code>.agg()</code>            | Realiza agregaciones en grupos.                | <code>resultados = grupos['columna_agregada'].agg(['sum', 'mean'])</code> |
| <code>.count()</code>          | Cuenta los elementos en cada grupo.            | <code>conteo = grupos['columna'].count()</code>                           |
| <code>.sum()</code>            | Calcula la suma de valores en grupos.          | <code>suma = grupos['columna'].sum()</code>                               |
| <code>.mean()</code>           | Calcula el promedio en grupos.                 | <code>promedio = grupos['columna'].mean()</code>                          |
| <code>.max()</code>            | Encuentra el valor máximo en grupos.           | <code>maximo = grupos['columna'].max()</code>                             |
| <code>.min()</code>            | Encuentra el valor mínimo en grupos.           | <code>minimo = grupos['columna'].min()</code>                             |
| <code>.std()</code>            | Calcula la desviación estándar en grupos.      | <code>desviacion = grupos['columna'].std()</code>                         |
| <code>.median()</code>         | Calcula la mediana en grupos.                  | <code>mediana = grupos['columna'].median()</code>                         |
| <code>.ngroups</code>          | Devuelve el número de grupos creados.          | <code>num_grupos = grupos.ngroups</code>                                  |

- Usos:
  - Analiza datos de algunas categorías
  -

- Divide, aplica (un estadístico) y agrupa (en función de la columna o columnas que especifiquemos)
- Si no le pasamos un estadístico nos devuelve un objeto de tipo groupby
- Filtra los datos
- Para convertir el resultado del groupby a un DataFrame:
  - `EL_RESULTADO_DEL_GROUPBY.reset_index()`
  - `pd.DataFrame(EL_RESULTADO_DEL_GROUPBY)`
- Estadísticos:
  - Se puede sacar un estadístico para todas las columnas o para una sola columna:

```
# PARA TODAS LAS COLUMNAS
df.groupby("gender").count()
```

```
# PARA UNA COLUMNA
df.groupby("gender")["num_polices"].count()
```

- También podemos incluir múltiples condiciones

```
# ESTAMOS HACIENDO UNA AGRUPACIÓN POR GÉNERO Y TAMAÑO DE VEHÍCULO
df.groupby(["gender", "vehicule_size"])["num_polices"].count()
```

- Si quiero calcular más de un estadístico ==

```
.agg(TIENE QUE IR EN FORMATO LISTA)
```

```
df.groupby("gender")["col1"].agg(["mean", "std"])
```

- Nulos:
  - Tenemos el parámetro `dropna`.
  - El groupby por defecto **LOS IGNORA**, es decir, `dropna = True`.
  - En caso de que queramos que **nos incluya** los nulos tendremos que establecer `dropna = False`.

## Apply

- Utilizamos el metodo `.apply()` para aplicar una misma función a las filas o columnas de un objeto DataFrame de Pandas.
-

- Sintaxis general: `df['nombre_columna'].apply(funcion_a_aplicar)`
- Utilizamos el metodo apply con lambdas, cuando queremos aplicar la misma funcion a varias columnas o utilizar funciones con más de un argumento.

- Sintaxis general:

```
df.apply(lambda x:funcion_a_aplicar(argumento1,argumento2))
```

- Si no tenemos una funcion definida previamente podemos aplicarlo directamente. Eg: `df.apply(lambda x:df[columna_1]/df[columna_2])`

- Funcion que recibe un parametro: SOLO APPLY

```
df[col].apply(mifuncion)
```

- OJO!!! QUE SI MI FUNCIÓN RECIBE UN PARAMETRO PERO EN EL RETURN TENEMOS MAS DE UN VALOR ENTONCES APPLY + LAMBDA

```
df.apply(lambda lola: mifuncion(lola[col1]), axis = 1, result_type =
```

- Funcion que recibe mas de un parametro: APPLY + LAMBDA

```
df.apply(lambda lololo: mifuncion(lololo[col1], lololo[col2]), axis = 1)
```

## Repaso de lo aprendido en el apply y métodos de limpieza

| Método               | Descripción   | Ejemplo en Python   |
|----------------------|---|---|
| <code>max</code>     | Encuentra el valor máximo en una serie o DataFrame.     | <code>df['columna'].max()</code>                                  |
| <code>min</code>     | Encuentra el valor mínimo en una serie o DataFrame.     | <code>df['columna'].min()</code>                                  |
| <code>dtypes</code>  | Devuelve los tipos de datos de las columnas.            | <code>df.dtypes</code>  |
| <code>apply</code>   | Aplica una función a lo largo de una serie o DataFrame. | <code>df['columna'].apply(funcion)</code>                         |
| <code>map</code>     | Reemplaza los valores en una serie utilizando un mapeo. | <code>df['columna'].map(mapeo)</code>                             |
| <code>replace</code> | Reemplaza valores específicos en una serie o DataFrame. | <code>df['columna'].replace({valor_original: valor_nuevo})</code> |

## 6.3 Resumen

### Instalación de Librerías

Instalar las siguientes librerías:

- scikit-learn
- seaborn
- matplotlib

### Importación de Librerías

Se importan `pandas`, `numpy`, y las herramientas de `sklearn` para imputación de nulos, junto con `seaborn` y `matplotlib` para visualización.

### Estrategias para la Imputación de Nulos

La imputación se diferencia de inventar datos ya que utiliza criterios basados en los datos vecinos.

### Métodos Comunes de Imputación

1. **Imputación Simple** usando la media, mediana o moda.
2. **Imputación Iterativa** utilizando modelos predictivos.
3. **KNN Imputation** utilizando los k vecinos más cercanos.

### Visualización de Datos

Utilización de `seaborn` y `matplotlib` para validar la imputación.

## Métodos de Gestión de Valores Nulos

| Métodos                    | Pros   | Contras   |
|----------------------------|--|---|
| <code>drop()</code>        | <ul style="list-style-type: none"><li>- Fácil de usar y entender.</li><li>- Elimina filas o columnas completas con valores nulos.</li></ul>  | <ul style="list-style-type: none"><li>- Puede resultar en la pérdida de información si se eliminan muchas filas o columnas.</li><li>- Puede afectar la representatividad de la muestra si se eliminan muchas filas.</li></ul> |
| <code>dropna()</code>      | <ul style="list-style-type: none"><li>- Permite eliminar filas o columnas con valores nulos de manera flexible.</li><li>- Permite especificar condiciones adicionales para la eliminación.</li></ul>                       | <ul style="list-style-type: none"><li>- Puede resultar en la pérdida de información si se eliminan muchas filas o columnas.</li><li>- Puede afectar la representatividad de la muestra si se eliminan muchas filas.</li></ul> |
| <code>fillna()</code>      | <ul style="list-style-type: none"><li>- Permite reemplazar los valores nulos con valores específicos, como la media, mediana o moda.</li><li>- Permite personalizar el valor de reemplazo según las necesidades.</li></ul> | <ul style="list-style-type: none"><li>- La elección del valor de reemplazo puede afectar los resultados del análisis.</li><li>- No tiene en cuenta las relaciones entre variables.</li></ul>                                  |
| <code>SimpleImputer</code> | <ul style="list-style-type: none"><li>- Permite imputar un mismo valor a todos los registros nulos de una columna.</li><li>- Puede utilizar estadísticos como la media, mediana o moda como valor de imputación.</li></ul> | <ul style="list-style-type: none"><li>- No tiene en cuenta las relaciones entre variables.</li><li>- No es adecuado para datos categóricos o variables con distribuciones no normales.</li></ul>                              |
| <code>KNNImputer</code>    | <ul style="list-style-type: none"><li>- Utiliza el algoritmo de los k-vecinos más cercanos para imputar valores nulos basándose en los valores de los vecinos más cercanos.</li></ul>                                      | <ul style="list-style-type: none"><li>- Requiere un tamaño de muestra suficientemente grande para obtener estimaciones precisas.</li><li>- Puede ser</li></ul>  |

| Métodos          | Pros   | Contras   |
|------------------|--|---|
|                  | <ul style="list-style-type: none"> <li>- Puede capturar relaciones no lineales entre variables.</li> </ul>   | <ul style="list-style-type: none"> <li>computacionalmente costoso para conjuntos de datos grandes.</li> </ul>   |
| IterativeImputer | <ul style="list-style-type: none"> <li>- Utiliza un enfoque iterativo para imputar valores nulos basado en modelos de aprendizaje automático.</li> <li>- Puede capturar relaciones complejas y no lineales entre variables.</li> </ul> | <ul style="list-style-type: none"> <li>- Requiere un tamaño de muestra suficientemente grande para obtener estimaciones precisas.</li> <li>- Puede ser computacionalmente costoso para conjuntos de datos grandes.</li> </ul> |

## Conclusiones

- La imputación es crucial para el análisis de datos.
- Diferentes métodos deben ser utilizados según el tipo de variable.
- La visualización es esencial para validar la imputación.



# 7.3 Resumen

## Resumen

En esta lección, aprendiste a utilizar diversas herramientas y técnicas para la visualización de datos en Python, específicamente con la librería Seaborn.

## Conceptos Clave

1. **Violin Plot:** Combina características de boxplot y KDE plot para mostrar la distribución de datos.

- Sintaxis básica:

```
sns.violinplot(x, y, data, color, palette, linewidth)
```

- Parámetros importantes:
  - `x` : Variable categórica.
  - `y` : Variable numérica.
  - `data` : DataFrame con los datos.
  - `color` : Color para una sola variable.
  - `palette` : Paleta de colores para variables categóricas.
  - `linewidth` : Grosor de las líneas.

2. **Interpretación del Violin Plot:**

- La forma del violín muestra la distribución.
- Los picos indican densidad alta de datos.
- La línea central puede representar la mediana o la media.

3. **Boxplot:** Visualiza la distribución de datos a través de sus cuartiles y valores atípicos.

- Sintaxis básica:

```
sns.boxplot(x, y, data, hue, palette)
```

- Parámetros:
  - `hue` : Agrupación adicional.

#### 4. Subplots y Facets:

- Utilización de `plt.subplots` y `sns.FacetGrid` para crear múltiples gráficos en una sola figura.

#### 5. Heatmaps: Representan datos matriciales mediante colores.

- Sintaxis básica:

```
sns.heatmap(data, annot, cmap)
```

- Parámetros:
  - `annot` : Anotaciones en cada celda.
  - `cmap` : Mapa de colores.

## Notas

- Todas las gráficas deben tener título.
- Etiquetas de los ejes deben estar en español.
- Se anima a personalizar las gráficas para practicar.

## Métodos para modificar las gráficas:

| Modificación               | Método <b>sin</b> subplot                                 | Método <b>con</b> subplot                               |
|----------------------------|---|---|
| Añadir título              | <code>plt.title()</code>                                  | <code>axes[n].set_title()</code>                        |
| Cambiar nombre del eje x   | <code>plt.xlabel()</code>                                 | <code>axes[n].set_xlabel()</code>                       |
| Cambiar nombre del eje y   | <code>plt.ylabel()</code>                                 | <code>axes[n].set_ylabel()</code>                       |
| Quitar línea de la derecha | <code>plt.gca().spines['right'].set_visible(False)</code> | <code>axes[n].spines['right'].set_visible(False)</code> |
| Limitar el eje x           | <code>plt.xlim()</code>                                   | <code>axes[n].set_xlim()</code>                         |
| Limitar el eje y           | <code>plt.ylim()</code>                                   | <code>axes[n].set_ylim()</code>                         |

|  |                           |                                   |
|--|---------------------------|-----------------------------------|
| Cambiar las propiedades de las etiquetas del eje x | <code>plt.xticks()</code> | <code>axes[n].set_xticks()</code> |
| Cambiar las propiedades de las etiquetas del eje y | <code>plt.yticks()</code> | <code>axes[n].set_yticks()</code> |

## 8.6 Resumen

### ¿Qué es la estadística descriptiva?

La estadística descriptiva es una rama de la estadística que se encarga de recolectar, analizar y caracterizar un conjunto de datos con el objetivo de describir las características de dicho conjunto.

### Tipos de medidas en estadística descriptiva

- **Medidas de centralización:** Indicadores que muestran el centro de una distribución de datos.
- **Medidas de dispersión:** Indicadores que muestran la variabilidad de los datos.
- **Medidas de correlación:** Indicadores que muestran la relación entre dos variables.

### Medidas de centralización

- **Media:** La media aritmética es el promedio de un conjunto de datos, calculado sumando todos los valores y dividiendo por el número total de datos.
- **Mediana:** La mediana es el valor que separa la mitad superior de la mitad inferior de un conjunto de datos ordenados.
- **Moda:** La moda es el valor que aparece con mayor frecuencia en un conjunto de datos.

### Medidas de dispersión

- **Rango:** El rango es la diferencia entre el valor máximo y el valor mínimo de un conjunto de datos.
- **Varianza:** La varianza mide la dispersión de los datos con respecto a la media. Se calcula como el promedio de las diferencias al cuadrado entre cada valor y la media.
- **Desviación estándar:** La desviación estándar es la raíz cuadrada de la varianza y proporciona una medida de la dispersión de los datos en las mismas unidades

que los datos originales.

## Medidas de correlación

- Covarianza: La covarianza indica la dirección de la relación lineal entre dos variables.
- Coeficiente de correlación: El coeficiente de correlación mide la fuerza y la dirección de la relación lineal entre dos variables, con valores que van desde -1 hasta 1.

## Conclusión

La estadística descriptiva es una herramienta fundamental para el análisis y la comprensión de datos. A través de sus diferentes medidas, es posible obtener una visión clara y concisa de las características principales de un conjunto de datos.

## 9.7 Resumen

La estadística inferencial permite hacer generalizaciones sobre una población basándose en los datos obtenidos de una muestra. A diferencia de la estadística descriptiva, que se enfoca en describir los datos, la inferencial se centra en hacer predicciones y pruebas de hipótesis.

### ¿Qué diferencia hay entre estadística descriptiva y estadística inferencial?

- **Estadística descriptiva:** Se utiliza para describir y resumir los datos de una muestra mediante gráficos, tablas y medidas de resumen como la media, mediana y desviación estándar.
- **Estadística inferencial:** Se utiliza para hacer inferencias sobre una población a partir de una muestra, utilizando métodos como intervalos de confianza y pruebas de hipótesis.

## Conceptos de Población y Muestra

### Población

Es el conjunto completo de todos los elementos que estamos interesados en estudiar.

### Muestra

Es un subconjunto de la población que se selecciona para el estudio. Debe ser representativa para que las conclusiones sean válidas.

### Diferencia entre Población y Muestra

La población incluye todos los elementos posibles de estudio, mientras que la muestra es solo una parte seleccionada de la población.

## Distribución de la muestra

La distribución de la muestra describe cómo se distribuyen los valores de la muestra y se utiliza para hacer inferencias sobre la población.

## Hipótesis nula vs Hipótesis alternativa

- **Hipótesis nula ( $H_0$ ):** Es una afirmación general o defecto que no hay efecto o diferencia.
- **Hipótesis alternativa ( $H_1$ ):** Es lo opuesto a la hipótesis nula, indicando que hay un efecto o una diferencia significativa.

## Pasos en la prueba de hipótesis

1. Plantear las hipótesis nula y alternativa.
2. Seleccionar un nivel de significancia.
3. Recopilar y analizar los datos de la muestra.
4. Tomar una decisión basada en los resultados.

## Intervalos de confianza

Un intervalo de confianza proporciona un rango de valores, derivado de los datos de la muestra, que probablemente contiene el valor verdadero del parámetro de la población.

## Error Tipo I y Tipo II

- **Error Tipo I:** Rechazar la hipótesis nula cuando es verdadera.
- **Error Tipo II:** No rechazar la hipótesis nula cuando es falsa.

## Distribución muestral de la media

La distribución de la media muestral se utiliza para entender cómo varía la media de la muestra y se aproxima a la distribución normal según el Teorema Central del Límite.

## Pruebas de hipótesis

Las pruebas de hipótesis son procedimientos que permiten decidir si los datos de la muestra proporcionan suficiente evidencia para rechazar la hipótesis nula.

### Tipos de pruebas

- Prueba t de Student
- Prueba Z
- Pruebas no paramétricas

## Conclusión

La estadística inferencial es una herramienta poderosa que permite tomar decisiones informadas y hacer predicciones basadas en datos muestrales. Es fundamental comprender los conceptos de hipótesis, intervalos de confianza y errores para interpretar correctamente los resultados.



## 10.2 Resumen

El A/B testing es una técnica utilizada en marketing y la investigación de usuarios para comparar dos versiones de un elemento con el fin de determinar cuál es más efectiva para lograr un objetivo específico, usualmente relacionado con la conversión.

### Importancia para los Analistas de Datos

1. **Optimización de Decisiones**
2. **Mejora de Experiencia de Usuario**
3. **Evidencia Empírica**

### Flujo de Trabajo en A/B Testing

1. **Definir Objetivos**
2. **Formular Hipótesis**
3. **Seleccionar Métricas**
4. **Dividir la Audiencia**
5. **Implementar Variaciones**
6. **Recoger Datos**
7. **Analizar Datos**
8. **Pruebas Estadísticas**
9. **Interpretar Resultados**

### Conclusiones

- El A/B Testing es crucial para la optimización continua.
- Permite decisiones informadas basadas en datos.
- Requiere planificación y análisis rigurosos.