Módulo 6.3: Manipulación de datos

Prof. Carlos Cedeño

Nuestro Dataset: Pasajeros de Aerolíneas

Para esta clase, usaremos el dataset `flights` de Seaborn. Contiene el número de pasajeros de una aerolínea (en miles) por mes y año, de 1949 a 1960. Es ideal para analizar tendencias.

```
import pandas as pd
import seaborn as sns # Importamos Seaborn para cargar el dataset
import matplotlib.pyplot as plt

# Cargamos el dataset 'flights' directamente desde Seaborn
df = sns.load_dataset('flights')

# No necesita limpieza, ¡está listo para usar!
print("Dataset de Vuelos listo para analizar:")
print(df.head())
```

1. Filtros y Operaciones Lógicas

Hacer las preguntas correctas es clave. Pandas nos permite filtrar nuestros datos usando condiciones lógicas de manera muy intuitiva.

Filtros Booleanos (`&`, `|`, `~`)

- `&` (Y): Ambas condiciones deben ser verdaderas.
- `|` (O): Al menos una condición debe ser verdadera.
- `~` (NO): Invierte la condición (niega).

```python
# Ejemplo 1: Encontrar los datos de 'Julio' (July) para los años posteriores a 1955
vuelos_julio_recientes = df[(df['month'] == 'Jul') & (df['year'] > 1955)]
print("Vuelos en Julio después de 1955:\n", vuelos_julio_recientes)

# Ejemplo 2: Encontrar los datos del primer año (1949) O del último año (1960)
vuelos_inicio_o_fin = df[(df['year'] == 1949) | (df['year'] == 1960)]
print("\nDatos de 1949 o 1960:\n", vuelos_inicio_o_fin.head())

# Ejemplo 3: Todos los datos que NO son del mes de Diciembre ('Dec')
vuelos_no_diciembre = df[~(df['month'] == 'Dec')]
print("\nDatos excluyendo Diciembre:\n", vuelos_no_diciembre.head())
```

Sustitución Condicional con `.where()` y `.mask()`

A veces no queremos filtrar filas, sino reemplazar valores basados en una condición.

- `.where(condicion)`: Mantiene los valores donde la condición es **Verdadera** y reemplaza el resto.
- `.mask(condicion)`: Mantiene los valores donde la condición es **Falsa** (enmascara la verdad) y reemplaza el resto.

```
# Crear una columna 'Decada'. Si el año es < 1955, será 'Inicio', si no, será 'Final'.
df['Decada'] = df['year'].where(df['year'] >= 1955, other='Inicio')
df['Decada'] = df['Decada'].mask(df['year'] >= 1955, other='Final')

print(df[['year', 'month', 'Decada']].tail())
```

2. Agregaciones y Estadísticas Descriptivas

Una vez que filtramos, el siguiente paso es resumir.

Resumen Rápido con `.describe()`

Este método es tu mejor amigo para obtener un resumen estadístico rápido de todas las columnas numéricas.

```
# Obtener estadísticas de las columnas numéricas
print(df.describe())
```

Estadísticas Individuales y Ordenamiento

Podemos calcular métricas específicas y ordenar el DataFrame para encontrar los valores más altos o más bajos.

```
# Estadísticas individuales
print(f"Promedio de pasajeros por mes: {df['passengers'].mean():.0f} mil")
print(f"Mes con el mayor número de pasajeros: {df['passengers'].max()} mil")
print(f"Total de pasajeros transportados (en miles): {df['passengers'].sum()}")

# Ordenar para encontrar los 5 meses con más pasajeros en la historia
top_5_meses = df.sort_values(by='passengers', ascending=False)
print("\nTop 5 meses con más pasajeros:")
print(top_5_meses.head(5))
```

3. Agregaciones por Categoría con `.groupby()`

Aquí es donde el análisis se vuelve "poderoso". `.groupby()` nos permite ejecutar las mismas agregaciones estadísticas, pero **para cada categoría por separado**.

El concepto es **Dividir-Aplicar-Combinar**:

1. **Dividir**: Separa el DataFrame en grupos basados en una categoría (ej. Año).
2. **Aplicar**: Calcula una estadística para cada grupo (ej. la suma de pasajeros).
3. **Combinar**: Une los resultados en un nuevo DataFrame.

```
# ¿Cuál fue el total de pasajeros para cada AÑO?
pasajeros_por_año = df.groupby('year')['passengers'].sum()
print("Total de pasajeros por año:\n", pasajeros_por_año)

# ¿Cuál es el promedio de pasajeros para cada MES a lo largo de los años? (Para ver la estacionalidad)
pasajeros_promedio_mes = df.groupby('month')['passengers'].mean().sort_values(ascending=False)
print("\nPasajeros promedio por mes:\n", pasajeros_promedio_mes)
```

4. Visualización Básica

Pandas se integra directamente con Matplotlib para crear visualizaciones de manera sencilla con el método `.plot()`.

Gráfico de Líneas: Viendo Tendencias en el Tiempo

Los gráficos de líneas son perfectos para ver la evolución de una variable en el tiempo. Este dataset es ideal para ello.

```
# Gráfico de líneas del total de pasajeros a lo largo de los años
pasajeros_por_año = df.groupby('year')['passengers'].sum()

pasajeros_por_año.plot(kind='line', figsize=(12, 7), marker='o', c='blue')

plt.title('Evolución del Tráfico Anual de Pasajeros (1949-1960)')
plt.ylabel('Total de Pasajeros (en miles)')
plt.xlabel('Año')
plt.grid(True)
plt.show()
```

Gráfico de Barras: Comparando Categorías

Los gráficos de barras son perfectos para comparar los resultados de un `groupby`, como la estacionalidad.

```
# Gráfico de barras del promedio de pasajeros por mes
pasajeros_promedio_mes = df.groupby('month')['passengers'].mean()

pasajeros_promedio_mes.plot(kind='bar', figsize=(12, 7), color='skyblue', edgecolor='black')

plt.title('Promedio Mensual de Pasajeros (Estacionalidad)')
plt.ylabel('Promedio de Pasajeros (en miles)')
plt.xlabel('Mes')
plt.xticks(rotation=0)
plt.show()
```

Boxplot: Entendiendo la Distribución por Categoría

Un boxplot (diagrama de caja) es excelente para ver cómo se distribuyen los datos y cómo esa distribución cambia entre diferentes categorías.

```
# Boxplot para ver la distribución de pasajeros cada año
# Esto nos muestra no solo cómo aumenta la media, sino también cómo aumenta la variabilidad.
df.boxplot(column='passengers', by='year', figsize=(12, 8))

plt.title('Distribución del Número de Pasajeros por Año')
plt.suptitle('') # Elimina el título automático que pone pandas
plt.ylabel('Número de Pasajeros (en miles)')
plt.xlabel('Año')
plt.show()
```