Módulo 2: Selección, filtrado y ordenamiento de datos y visualización con Python (3 horas)

Selección, filtrado y ordenamiento de datos con Pandas

Pandas proporciona diversas funciones para seleccionar, filtrar y ordenar datos en DataFrames.

```python
import pandas as pd

# Creación de un DataFrame de ejemplo
data = {'A': [1, 2, 3, 4], 'B': [5, 6, 7, 8], 'C': [9, 10, 11, 12]}
df = pd.DataFrame(data)

# Selección de columnas
columna_A = df['A']

# Selección de filas por índice
fila_1 = df.loc[1]

# Selección de filas por condición
filas_pares = df[df['A'] % 2 == 0]

# Ordenamiento de datos
df_ordenado = df.sort_values(by='B', ascending=False)
```

Operaciones vectoriales y matriciales con NumPy

NumPy permite realizar operaciones vectoriales y matriciales de manera eficiente.

```python
import numpy as np

# Operaciones elementales entre vectores
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

suma = a + b
producto_elemental = a * b

# Operaciones matriciales
A = np.array([[1, 2], [3, 4]])
B = np.array([[5, 6], [7, 8]])

producto_matricial = np.dot(A, B)
```

Carga y guardado de datos desde y hacia diferentes formatos

Pandas permite cargar y guardar datos en diversos formatos, incluyendo CSV, Excel y SQL.

```
# Leer datos desde un archivo CSV
datos_csv = pd.read_csv('datos.csv')

# Guardar datos en un archivo CSV
datos_csv.to_csv('datos_exportados.csv', index=False)

# Leer datos desde un archivo Excel
datos_excel = pd.read_excel('datos.xlsx', sheet_name='Hoja1')

# Guardar datos en un archivo Excel
datos_excel.to_excel('datos_exportados.xlsx', sheet_name='Hoja1', index=False)

# Leer datos desde una base de datos SQL
import sqlite3

conn = sqlite3.connect('database.db')
datos_sql = pd.read_sql_query("SELECT * FROM tabla", conn)
conn.close()

# Guardar datos en una base de datos SQL
conn = sqlite3.connect('database.db')
datos_sql.to_sql('tabla_exportada', conn, if_exists='replace', index=False)
conn.close()
```

Introducción a Matplotlib y Seaborn para visualización de datos

Matplotlib y Seaborn son bibliotecas de Python para la creación de gráficos y visualizaciones de datos.

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

Creación de gráficos de barras, líneas, histogramas, dispersión y diagramas de caja

```
# Gráfico de barras
categorias = ['A', 'B', 'C']
valores = [10, 15, 7]
plt.bar(categorias, valores)
plt.show()

# Gráfico de líneas
```

```
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y)
plt.show()

# Histograma
data = np.random.randn(100)
plt.hist(data, bins=10)
plt.show()

# Gráfico de dispersión
x = np.random.rand(50)
y = np.random.rand(50)
plt.scatter(x, y)
plt.show()

# Diagrama de caja
data = [np.random.normal(0, std, 100) for std in range(1, 4)]
plt.boxplot(data, vert=True, patch_artist=True)
plt.show()
```

Estos son ejemplos básicos de cómo crear gráficos de barras, líneas, histogramas, dispersión y diagramas de caja utilizando Matplotlib. En el siguiente módulo, exploraremos cómo personalizar estos gráficos con etiquetas, títulos, leyendas, colores y estilos.

Módulo 2: Selección, filtrado y ordenamiento de datos y visualización con Python (Continuación)

Personalización de gráficos con etiquetas, títulos, leyendas, colores y estilos

Etiquetas y títulos

Para agregar etiquetas a los ejes y un título al gráfico, utilizamos las funciones `xlabel()`, `ylabel()` y `title()`.

```
x = np.arange(1, 6)
y = x ** 2

plt.plot(x, y)
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.title('Gráfico de línea personalizado')
plt.show()
```

Leyendas

Las leyendas ayudan a identificar las líneas o elementos en un gráfico. Utilizamos la función `legend()` para agregar leyendas.

```
x = np.arange(1, 6)
y1 = x ** 2
y2 = x ** 3

plt.plot(x, y1, label='Cuadrado')
plt.plot(x, y2, label='Cubo')
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.title('Gráfico de línea con leyenda')
plt.legend()
plt.show()
```

Colores y estilos

Para cambiar el color y el estilo de las líneas, utilizamos argumentos adicionales en la función `plot()`.

```
x = np.arange(1, 6)
y1 = x ** 2
y2 = x ** 3

plt.plot(x, y1, color='red', linestyle='-', linewidth=2, marker='o',
label='Cuadrado')
plt.plot(x, y2, color='blue', linestyle='--', linewidth=2, marker='x',
label='Cubo')
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.title('Gráfico de línea con colores y estilos personalizados')
plt.legend()
plt.show()
```

Estilos predefinidos de Matplotlib

Matplotlib ofrece varios estilos predefinidos que se pueden utilizar para cambiar la apariencia de los gráficos. Para ver los estilos disponibles, utilizamos la función `plt.style.available`. Para aplicar un estilo, utilizamos la función `plt.style.use()`.

```
print(plt.style.available)

# Ejemplo utilizando el estilo 'ggplot'
plt.style.use('ggplot')

x = np.arange(1, 6)
y = x ** 2

plt.plot(x, y)
plt.xlabel('Eje X')
plt.ylabel('Eje Y')
plt.title('Gráfico de línea con estilo ggplot')
plt.show()
```

Con estas herramientas de personalización, puedes crear gráficos más atractivos y fáciles de interpretar en tus proyectos de ciencia de datos. En el próximo módulo, aprenderemos sobre técnicas más avanzadas de análisis y visualización de datos.