



Visualizaciones Avanzadas y Dashboards Interactivos - Día 3

Change Status

in-progress

40 min



Learning Objectives

- 1 Diseñar figuras compuestas que muestren múltiples perspectivas de datos
- 2 Crear visualizaciones estadísticas sofisticadas para análisis profundos
- 3 Implementar interactividad básica en dashboards analíticos

Theory

Practice

Quiz

Evidence



Actividades y Aprendizajes

Aprende todo sobre funciones y módulos en Python con ejemplos prácticos.

Task 1: Diseño de Figuras Compuestas con Subplots (10 minutos)

Las visualizaciones efectivas rara vez consisten en un solo gráfico. Los subplots permiten crear dashboards comprensivos que muestran múltiples perspectivas de los datos simultáneamente.

Arquitecturas de Subplots Comunes

Grid rectangular: Distribución regular de visualizaciones relacionadas.

```
# 2x2 grid para análisis multivariado
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle('Dashboard Ejecutivo de Ventas', fontsize=16, fontweight='bold')

# Asignar visualizaciones específicas
ax1.set_title('Tendencia Mensual') # Serie temporal
ax2.set_title('Distribución por Región') # Barras
```



us English



Sign Out





```
ax3.set_title('Top Productos') # Barras horizontales
ax4.set_title('Distribución de Clientes') # Pie o barras
```

```
plt.tight_layout()
```

Layout asimétrico: Distribuciones que enfatizan jerarquía visual.

```
# Layout con gráfico principal y secundarios
fig = plt.figure(figsize=(16, 10))

# Gráfico principal (70% del espacio)
ax_main = plt.subplot2grid((3, 4), (0, 0), colspan=3, rowspan=2)
ax_main.set_title('Evolución Principal de Ventas', fontsize=14, fontweight='bold')

# Panel derecho con KPIs (30% del espacio)
ax_kpi1 = plt.subplot2grid((3, 4), (0, 3), rowspan=1)
ax_kpi2 = plt.subplot2grid((3, 4), (1, 3), rowspan=1)

# Panel inferior para detalles
ax_detail = plt.subplot2grid((3, 4), (2, 0), colspan=4)
ax_detail.set_title('Análisis Detallado por Segmento', fontsize=12)

plt.tight_layout()
```

Coordinación Visual entre Subplots

Escalas compartidas: Mantener consistencia en ejes para comparación.

```
# Compartir eje Y para comparación directa
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6), sharey=True)

# Ambos gráficos tendrán la misma escala Y
ax1.bar(categorias_a, valores_a)
ax2.bar(categorias_b, valores_b)
```

Paletas coordinadas: Colores consistentes que signifiquen lo mismo en gráficos diferentes.

```
# Definir mapeo de colores consistente
color_map = {
    'Producto A': '#1f77b4',
```



```
'Producto B': '#ff7f0e',
'Producto C': '#2ca02c',
'Producto D': '#d62728'
}

# Aplicar en múltiples subplots
for ax in [ax1, ax2, ax3]:
    for categoria, color in color_map.items():
        # Usar mismo color para misma categoría en todos los gráficos
        datos_categoria = df[df['categoria'] == categoria]
        ax.plot(datos_categoria['x'], datos_categoria['y'], color=color, label=categoria)
```

Task 2: Visualizaciones Estadísticas Avanzadas (12 minutos)

Más allá de gráficos básicos, las visualizaciones estadísticas avanzadas permiten explorar relaciones complejas y distribuciones multivariadas.

Heatmaps: Visualización de Matrices de Correlación

Los heatmaps usan color para representar valores en matrices bidimensionales, ideales para correlaciones y patrones complejos.

```
# Matriz de correlación completa
correlation_matrix = df.select_dtypes(include=[np.number]).corr()

# Heatmap de correlaciones
fig, ax = plt.subplots(figsize=(10, 8))
im = ax.imshow(correlation_matrix, cmap='RdYlBu_r', vmin=-1, vmax=1)

# Configurar ejes
ax.set_xticks(np.arange(len(correlation_matrix.columns)))
ax.set_yticks(np.arange(len(correlation_matrix.columns)))
ax.set_xticklabels(correlation_matrix.columns, rotation=45, ha='right')
ax.set_yticklabels(correlation_matrix.columns)

# Añadir valores numéricos
for i in range(len(correlation_matrix.columns)):
    for j in range(len(correlation_matrix.columns)):
        text = ax.text(j, i, f'{correlation_matrix.iloc[i, j]:.2f}',
                       ha='center', va='center', color='black', fontsize=8)

# Barra de color
cbar = ax.figure.colorbar(im, ax=ax, shrink=0.8)
cbar.ax.set_ylabel('Correlación', rotation=-90, va='bottom')
```



```
ax.set_title('Matriz de Correlación de Variables Empresariales', fontweight='bold', pad=20)
plt.tight_layout()
plt.savefig('heatmap_correlaciones.png', dpi=300, bbox_inches='tight')
```

Violin Plots: Distribuciones con Densidad

Los violin plots combinan box plots con estimaciones de densidad kernel, mostrando forma completa de distribuciones.

```
# Comparación de distribuciones por categoría
fig, ax = plt.subplots(figsize=(12, 8))

# Datos por segmento de cliente
segmentos = df['segmento_cliente'].unique()
data_por_segmento = [df[df['segmento_cliente'] == seg]['valor_compra']
                      for seg in segmentos]

# Violin plot
parts = ax.violinplot(data_por_segmento, showmeans=True, showextrema=True)

# Personalización
for i, pc in enumerate(parts['bodies']):
    pc.set_facecolor(colores[i % len(colores)])
    pc.set_alpha(0.7)

# Configurar partes del violin plot
parts['cmeans'].set_color('black')
parts['cmins'].set_color('black')
parts['cmaxes'].set_color('black')
parts['cbars'].set_color('black')

# Etiquetas y título
ax.set_xticks(range(1, len(segmentos) + 1))
ax.set_xticklabels(segmentos)
ax.set_title('Distribución de Valor de Compra por Segmento de Cliente',
             fontweight='bold', pad=20)
ax.set_ylabel('Valor de Compra ($)', labelpad=10)
ax.grid(True, alpha=0.3, axis='y')

plt.savefig('violin_segmentos_cliente.png', dpi=300, bbox_inches='tight')
```

Scatter Plots Matrices: Análisis Multivariado

Las matrices de scatter plots permiten explorar relaciones entre múltiples variables simultáneamente.



```
# Variables de interés para análisis multivariado
```

```
variables_clave = ['ventas', 'margen', 'satisfaccion', 'retencion', 'valor_vida']
```

```
# Crear scatter plot matrix
```

```
n_vars = len(variables_clave)
```

```
fig, axes = plt.subplots(n_vars, n_vars, figsize=(15, 15))
```

```
for i in range(n_vars):
```

```
    for j in range(n_vars):
```

```
        ax = axes[i, j]
```

```
        if i == j:
```

```
            # Diagonal: histogramas
```

```
            ax.hist(df[variables_clave[i]], bins=20, alpha=0.7, color=colores[0])
```

```
            ax.set_title(f'{variables_clave[i].title()}', fontsize=10)
```

```
        else:
```

```
            # Off-diagonal: scatter plots
```

```
            ax.scatter(df[variables_clave[j]], df[variables_clave[i]],
```

```
                        alpha=0.6, s=30, color=colores[1])
```

```
            # Añadir línea de tendencia si correlación fuerte
```

```
            corr = df[variables_clave[j]].corr(df[variables_clave[i]])
```

```
            if abs(corr) > 0.5:
```

```
                z = np.polyfit(df[variables_clave[j]], df[variables_clave[i]], 1)
```

```
                p = np.poly1d(z)
```

```
                x_trend = np.linspace(df[variables_clave[j]].min(), df[variables_clave[j]].max(), 50)
```

```
                ax.plot(x_trend, p(x_trend), 'r--', alpha=0.8, linewidth=1)
```

```
# Configurar etiquetas
```

```
for i, var in enumerate(variables_clave):
```

```
    axes[-1, i].set_xlabel(var.title(), fontsize=10)
```

```
    axes[i, 0].set_ylabel(var.title(), fontsize=10)
```

```
plt.suptitle('Matriz de Relaciones Multivariadas', fontsize=14, fontweight='bold', y=0.95)
```

```
plt.tight_layout()
```

```
plt.savefig('scatter_matrix_multivariada.png', dpi=300, bbox_inches='tight')
```

Task 3: Introducción a Dashboards Interactivos (8 minutos)

La interactividad permite a los usuarios explorar datos dinámicamente, creando dashboards que se adaptan a diferentes preguntas.

Widgets Básicos con Matplotlib



Matplotlib ofrece interactividad básica a través de widgets, aunque limitada comparada con bibliotecas especializadas.

```
from matplotlib.widgets import Slider, Button

# Configurar figura con espacio para controles
fig, ax = plt.subplots(figsize=(12, 8))
plt.subplots_adjust(bottom=0.25)

# Gráfico inicial
x = np.linspace(0, 10, 100)
line, = ax.plot(x, np.sin(x), linewidth=2, color=colores[0])
ax.set_title('Función Interactiva: sen(x + fase)')
ax.grid(True, alpha=0.3)

# Crear slider para fase
ax_phase = plt.axes([0.2, 0.1, 0.5, 0.03])
slider_phase = Slider(ax_phase, 'Fase', -np.pi, np.pi, valinit=0)

# Función de actualización
def update(val):
    phase = slider_phase.val
    line.set_ydata(np.sin(x + phase))
    fig.canvas.draw_idle()

# Conectar slider a función de actualización
slider_phase.on_changed(update)

# Botón de reset
ax_reset = plt.axes([0.8, 0.05, 0.1, 0.04])
button_reset = Button(ax_reset, 'Reset')

def reset(event):
    slider_phase.set_val(0)
    update(0)

button_reset.on_clicked(reset)

plt.show()
```

Introducción a Plotly para Interactividad Web

Plotly ofrece interactividad nativa del navegador web, creando visualizaciones que responden a clics, hover y zoom.



Dashboard

Career Path

Forms

Profile



Sign Out

Toggle theme: Light Theme



```
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Crear subplot interactivo
fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=('Ventas por Región', 'Tendencia Temporal',
                    'Distribución por Producto', 'Top Clientes'),
    specs=[[{'type': 'bar'}, {'type': 'scatter'}],
           [{'type': 'pie'}, {'type': 'bar'}]]
)

# Gráfico 1: Barras de ventas por región
ventas_region = df.groupby('region')['total_venta'].sum()
fig.add_trace(
    go.Bar(x=ventas_region.index, y=ventas_region.values,
           name='Ventas por Región', marker_color=colores[0]),
    row=1, col=1
)

# Gráfico 2: Tendencia temporal
ventas_tiempo = df.groupby('fecha')['total_venta'].sum().reset_index()
fig.add_trace(
    go.Scatter(x=ventas_tiempo['fecha'], y=ventas_tiempo['total_venta'],
               mode='lines+markers', name='Evolución de Ventas',
               line=dict(color=colores[1], width=3)),
    row=1, col=2
)

# Gráfico 3: Pie chart de productos
ventas_producto = df.groupby('producto')['total_venta'].sum()
fig.add_trace(
    go.Pie(labels=ventas_producto.index, values=ventas_producto.values,
           name='Distribución por Producto'),
    row=2, col=1
)

# Gráfico 4: Top clientes
top_clientes = df.groupby('cliente')['total_venta'].sum().nlargest(10)
fig.add_trace(
    go.Bar(x=top_clientes.values, y=top_clientes.index, orientation='h',
           name='Top 10 Clientes', marker_color=colores[3]),
    row=2, col=2
)

# Configurar layout
fig.update_layout(
```



```
        title_text='Dashboard Interactivo de Ventas Empresariales',
        title_font_size=20,
        showlegend=False,
        height=800
    )

# Guardar como HTML interactivo
fig.write_html('dashboard_interactivo.html')
print("Dashboard interactivo guardado como 'dashboard_interactivo.html'")
```