

✓ Plotly con Python

¿Qué es Plotly?

- Plotly es una biblioteca de visualización de datos en Python que permite crear gráficos interactivos y altamente personalizables.
- A diferencia de Matplotlib y Seaborn, Plotly genera gráficos que pueden interactuar con el usuario, lo que lo hace ideal para análisis de datos, dashboards y aplicaciones web.
- Características principales.
 - Gráficos interactivos (zoom, selección de datos, tooltips).
 - Compatibilidad con Jupyter Notebook y Dash (para dashboards).+
 - Permite exportar gráficos en HTML, PNG, SVG, y PDF.
 - Soporta gráficos con 2D, 3d y de mapas geográficos.
 - Integración con Pandas, NumPy y otras bibliotecas de ciencia de datos.



Instalación y Configuración.

Para instalar Plotly, usamos el siguiente comando en tu terminal o directo en tu cuaderno de Jupyter:

- `pip install plotly` ---> Terminal
- `%pip install plotly` ---> Celda de Notebook

✓ Importación.

- Para gráficos rápidos y fáciles.

```
import plotly.express as px
```

- Para gráficos más avanzados.

```
import plotly. graph_objects as go
```

```
import plotly.express as px
import plotly. graph_objects as go
```

✓ Si queremos que los gráficos se muestren dentro del notebook, usamos:

```
import plotly.io as pio
pio.renderers.default = 'notebook'
```

```
import plotly.io as pio
pio.renderers.default = 'iframe'
```

✓ Si es el caso en que se usa Google Colab, hay que cambiar 'notebook' por 'colab':

```
import plotly.io as pio
pio.renderers.default = 'colab'
```

✓ Diferencias entre Plotly Express y graph Objects.

- Plotly tiene dos formas principales de crear gráficos:
 - Plotly Express (px) --> Forma rápida y sencilla
 - Graph Objects (go) --> Permite mayor personalización.



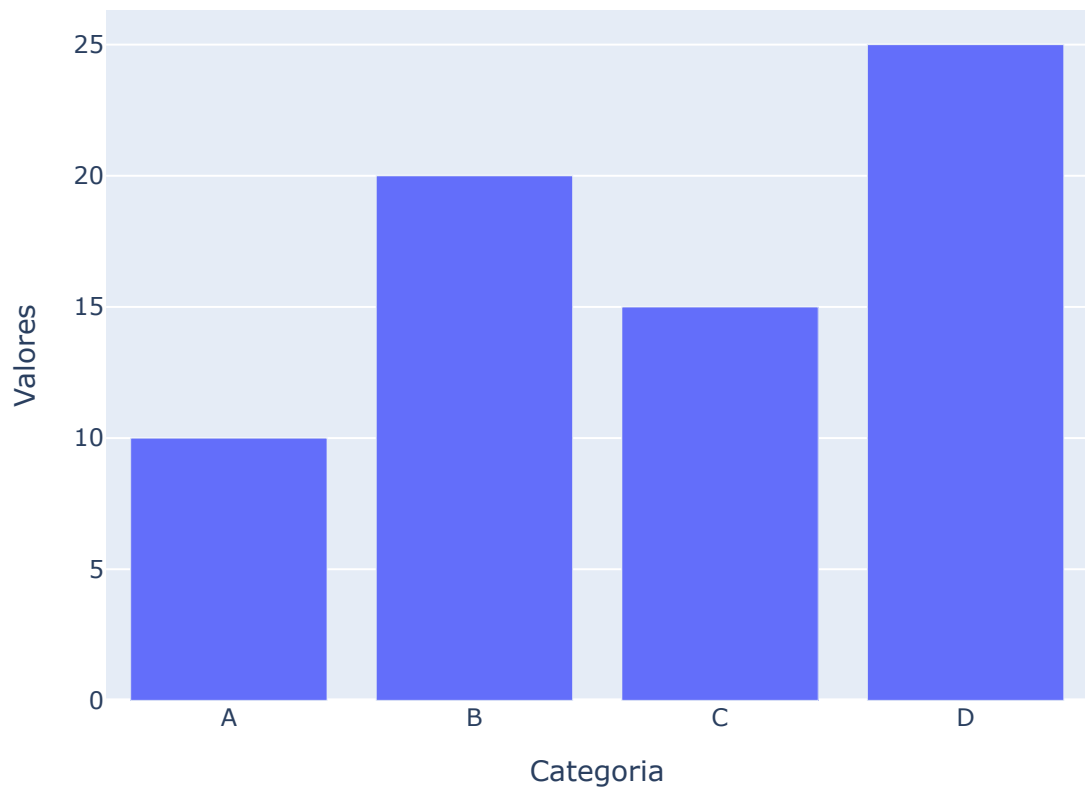
```
import plotly.express as px
import pandas as pd

data = pd.DataFrame({
    'Categoria': ['A', 'B', 'C', 'D'],
    'Valores': [10, 20, 15, 25]
})

fig = px.bar(data, x='Categoria', y='Valores', title='Gráfico de Barras con Plotly Express')
fig.show()
```



Gráfico de Barras con Plotly Express

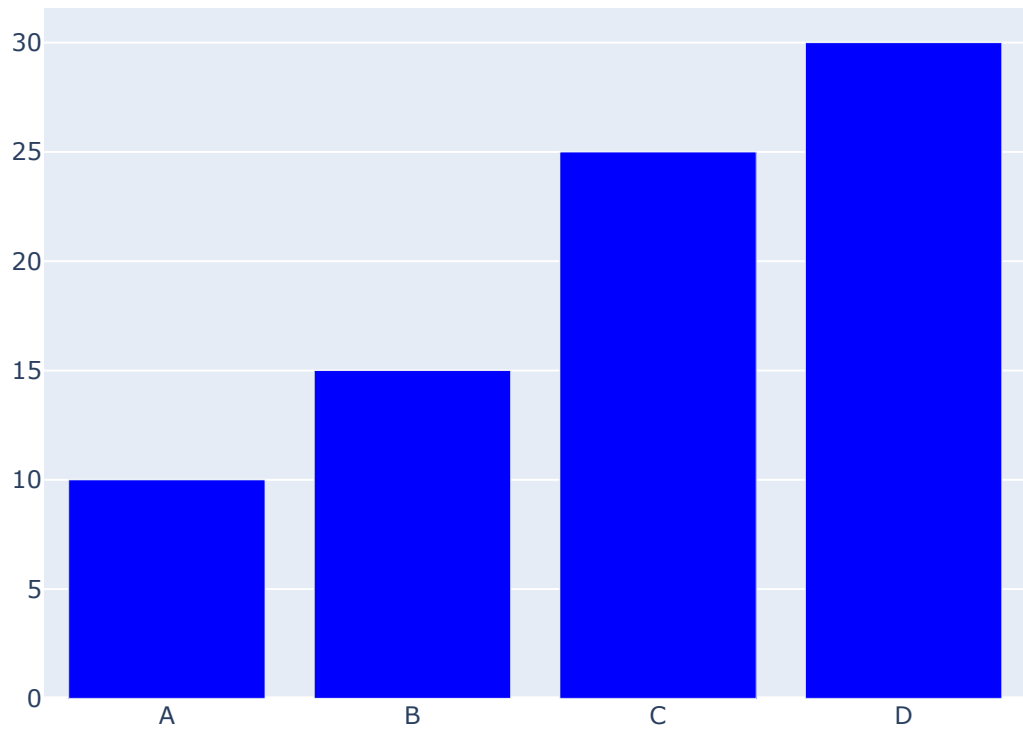


✓ Ejemplo con Graph Objects (más personalizable)

```
import plotly.graph_objects as go

fig = go.Figure()

fig.add_trace(go.Bar(
    x = ['A', 'B', 'C', 'D'],
    y = [10, 15, 25, 30],
    name = 'Valores',
    marker_color = 'blue'
))
```



¿Cuál usar?

- px → Cuando quieres hacer gráficos rápidos sin mucha personalización
- go → Cuando necesitas ajustar cada detalle del gráfico.

✓ Primeros gráficos interactivos con Plotly Express

- Aquí veremos algunos gráficos con Plotly Express

✓ Gráfico de dispersión (scatter plot)

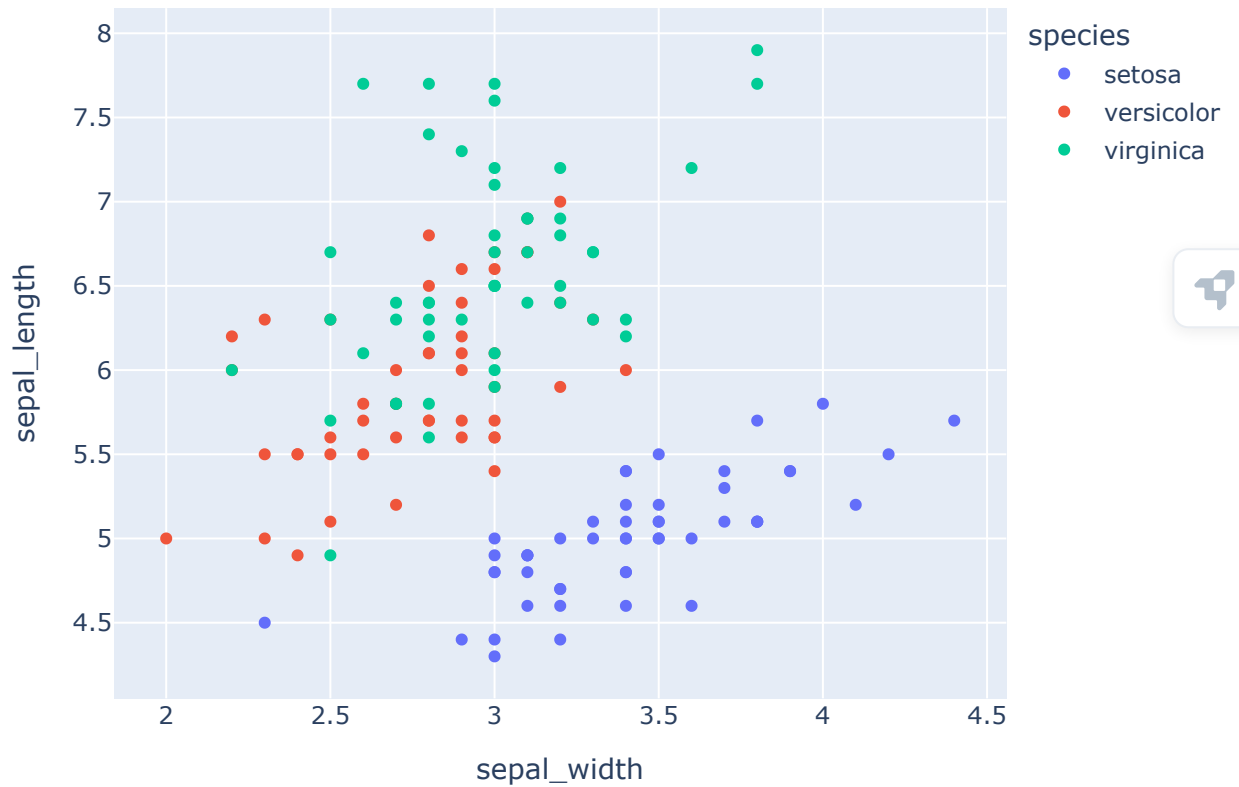
```
import plotly.express as px

df = px.data.iris()

fig = px.scatter(
    df, x='sepal_width', y='sepal_length', color='species',
    title='Gráfico de dispersión con plotly Express'
)
fig.show()
```



Gráfico de dispersión con plotly Express



✓ Gráfico de líneas

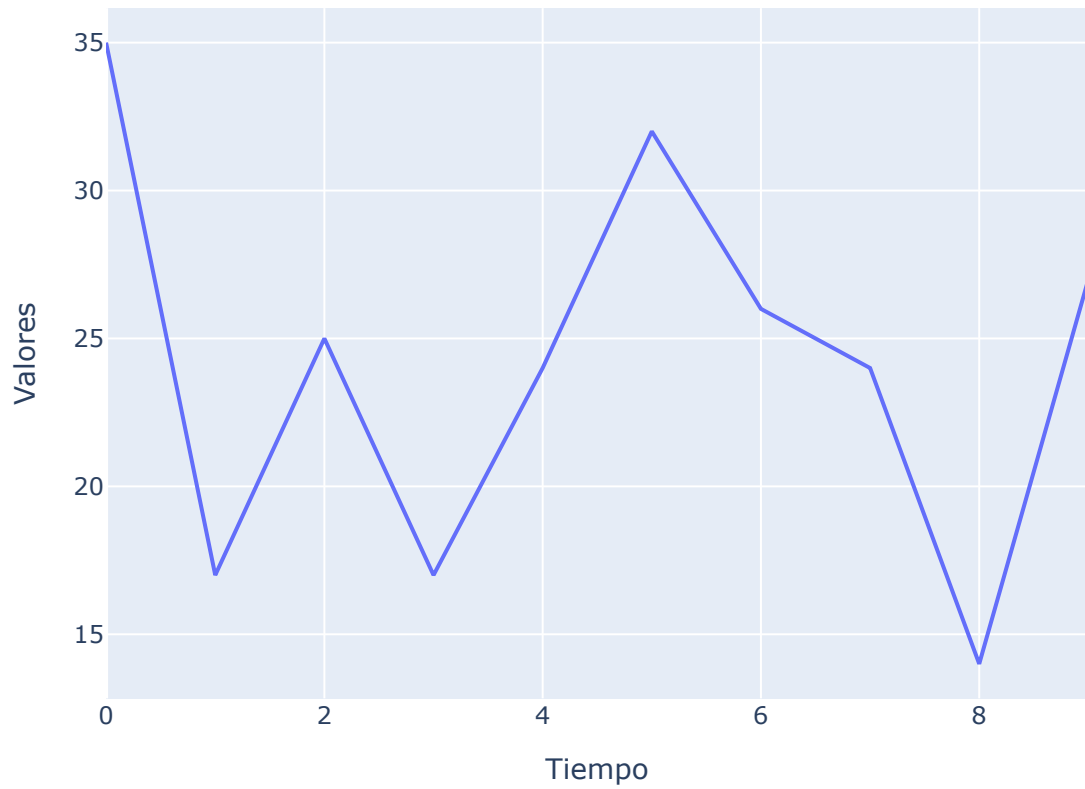
```
import numpy as np
import pandas as pd

df = pd.DataFrame({
    'Tiempo': np.arange(10),
    'Valores': np.random.randint(10, 50, 10)
})

fig = px.line(df, x='Tiempo', y='Valores', title='Gráfico de Líneas')
fig.show()
```



Gráfico de Líneas



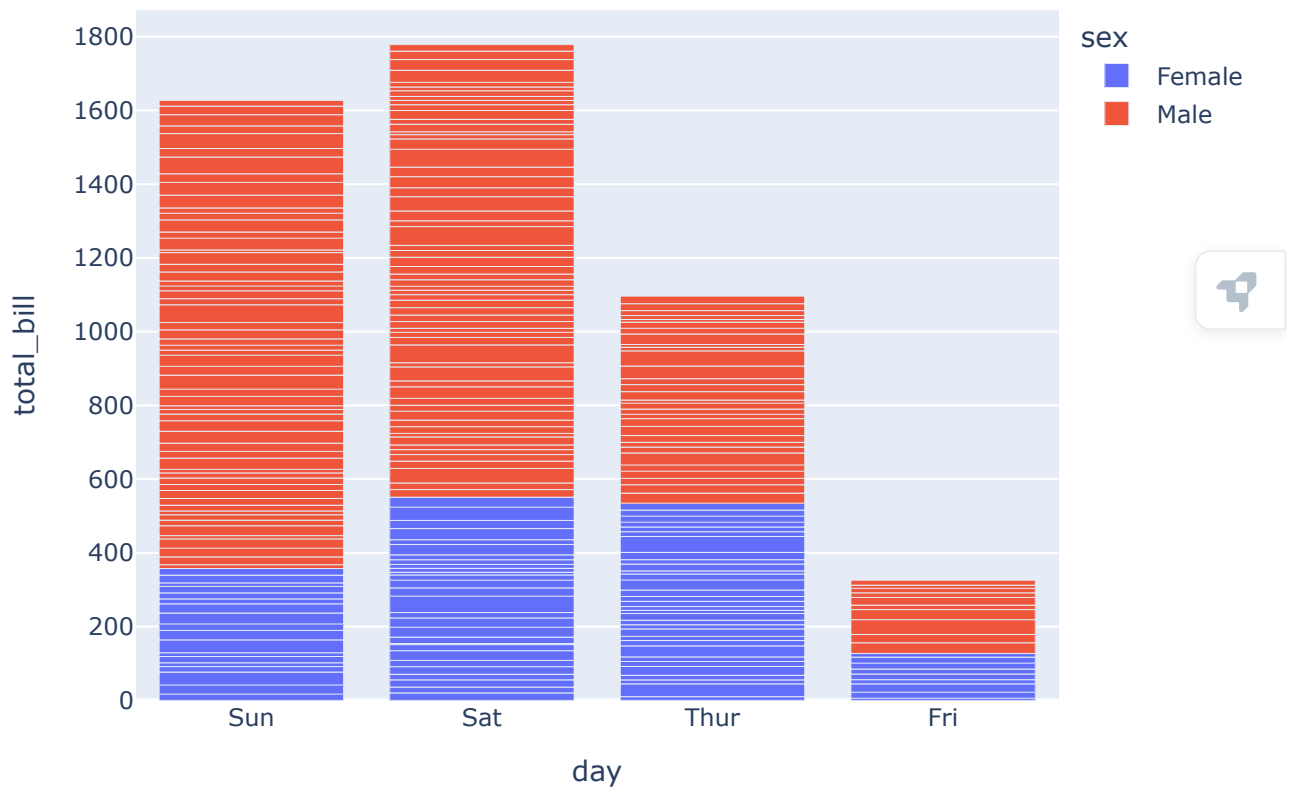
✓ Gráfico de barras con categorías

```
df = px.data.tips()

fig = px.bar(df, x='day', y='total_bill', color='sex',
             title='Total de cuentas por día y género' )
fig.show()
```



Total de cuentas por día y género

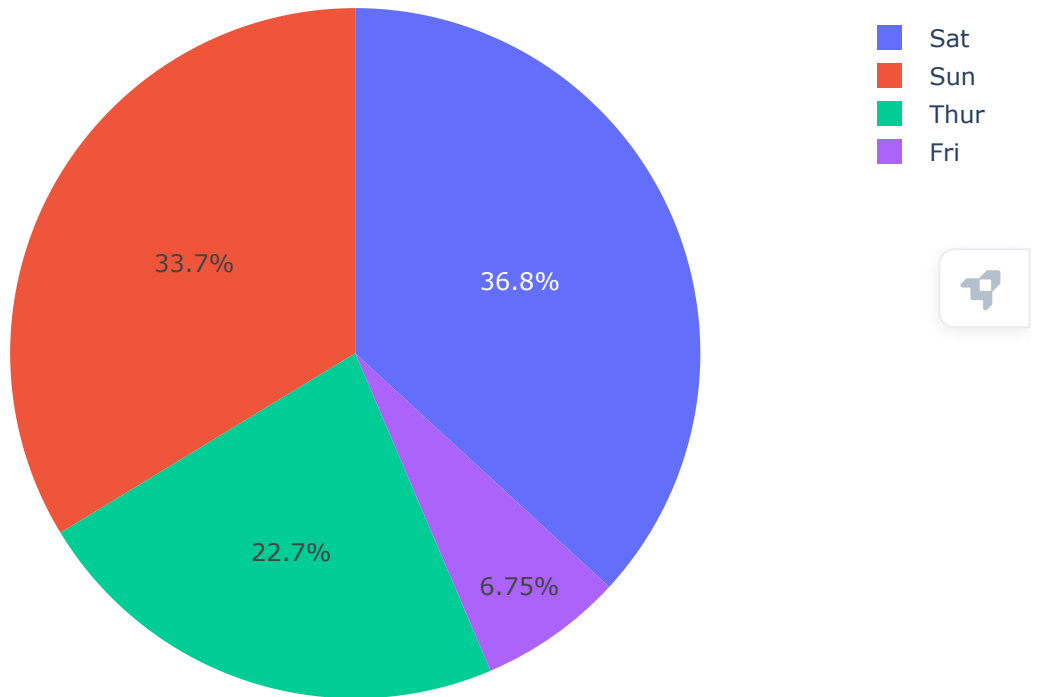


✓ Gráfico de barras con categorías

```
fig = px.pie(df, names='day', values='total_bill',  
             title='Distribución de cuentas por día')  
fig.show()
```



Distribución de cuentas por día



✓ Guardar gráficos en distintos formatos

- Plotly permite guardar gráficos en varios formatos.
- PNG O SVG

```
!pip install -U kaleido
```



Collecting kaleido

Downloading kaleido-0.2.1-py2.py3-none-manylinux1_x86_64.whl.metadata (15 kB)

Downloading kaleido-0.2.1-py2.py3-none-manylinux1_x86_64.whl (79.9 MB)

79.9/79.9 MB 9.4 MB/s eta 0:00:00

Installing collected packages: kaleido

Successfully installed kaleido-0.2.1

```
import kaleido
```

```
fig.write_image("grafico.png")
```

```
fig.write_image("grafico.svg")
```

✓ Guardar como HTML interactivo


```
fig.write_html("grafico_interactivo.html")
```

✓ guardar como JSON para reproducirlo después

```
import json

json_data = fig.to_json()
with open('grafico.json', 'w') as f:
    json.dump(json_data, f)
```

✓ Introducción a la API de Plotly para personalización avanzada

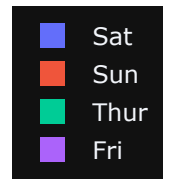
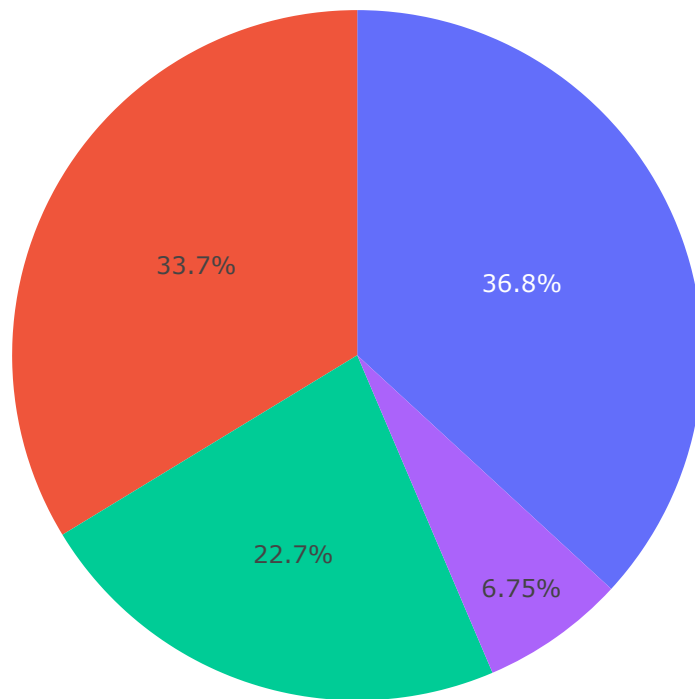


- Plotly tiene una API que permite hacer cambios dinámicos en los gráficos.
- Ejemplo de modificación de diseño con `update_layout()`:

```
fig.update_layout(
    title='Titulo Personalizado',
    xaxis_title='Eje X',
    yaxis_title='Eje Y',
    template='plotly_dark'
)
fig.show()
```



Titulo Personalizado

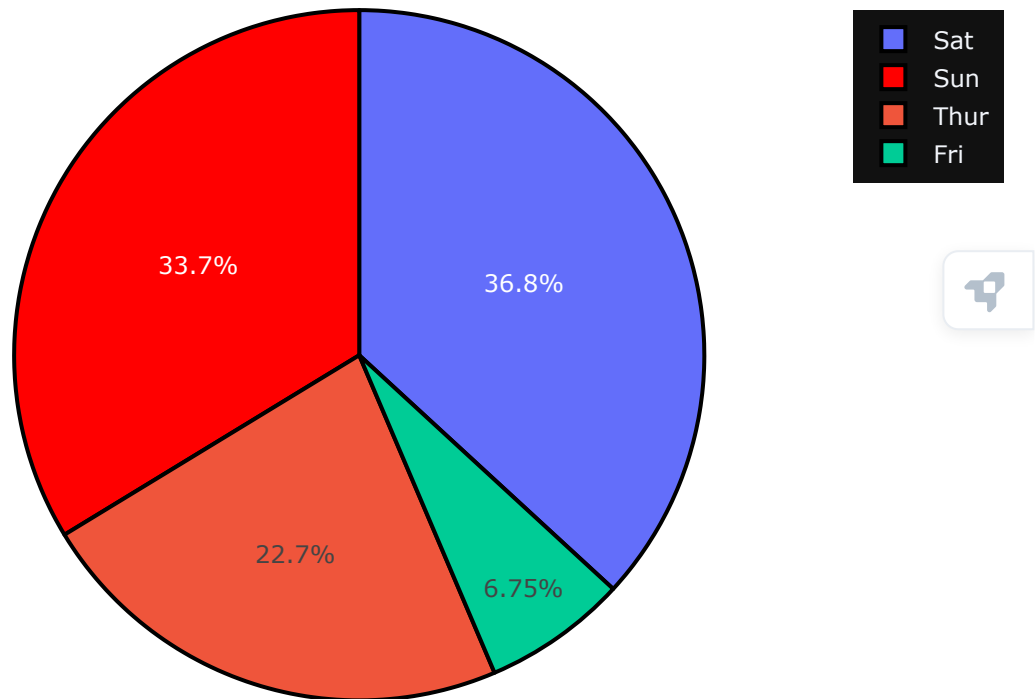


✓ Ejemplo de cambio de colores y diseño de ejes.

```
fig.update_traces(marker=dict(colors=["red"], line=dict(width=2, color="black")))
fig.update_xaxes(showgrid=False, title="Nuevo Eje X")
fig.update_yaxes(showgrid=False, title="Nuevo Eje Y")
fig.show()
```



Titulo Personalizado



✓ Gráficos básicos con Plotly

Estructura general de un gráfico en Plotly

- La estructura básica de un gráfico en Plotly se compone principalmente de tres elementos:
 1. Datos (data) ---> Información que se va a graficar.
 2. Diseño (layout) ---> Configuración visual del gráfico.
 3. Configuración (config) ---> Controla aspectos interactivos y de visualización.

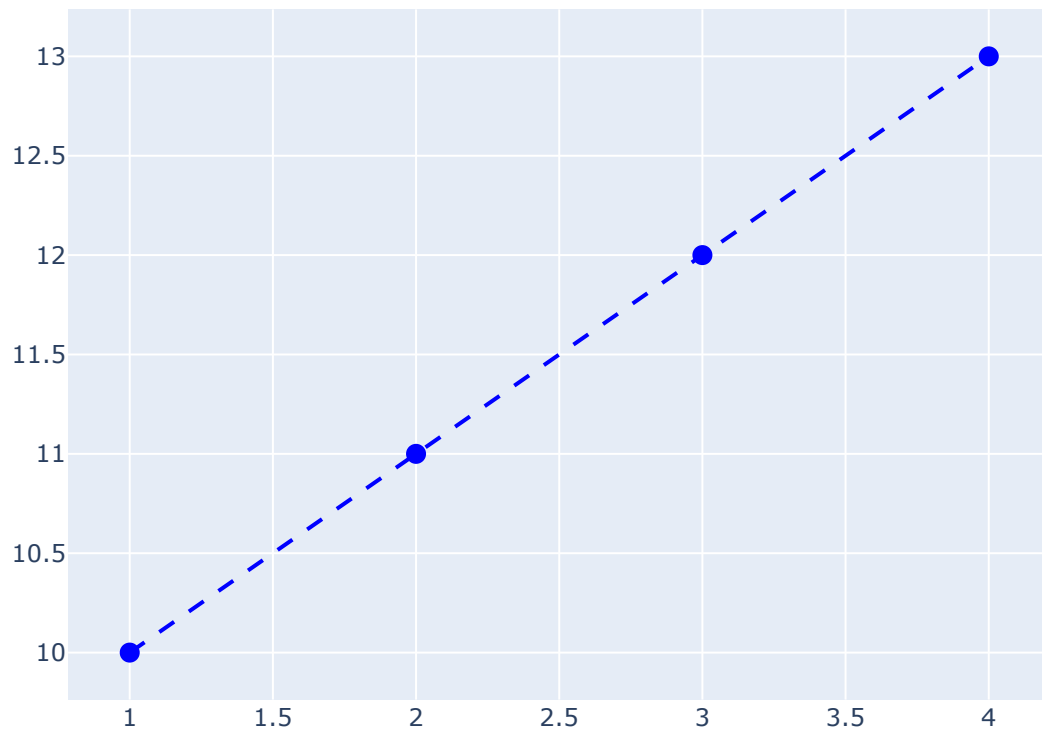
✓ Datos (data)

- Los datos se definen como go o px dependiendo de la biblioteca interna que uses.
- Ejemplo con plotly.graph_objects (más flexible y detallado)

```
import plotly.io as pio
pio.renderers.default = 'colab'
```

```
import plotly.graph_objects as go

fig = go.Figure(
    data = [
        go.Scatter(
            x = [1,2,3,4],
            y = [10,11,12,13],
            mode = 'lines+markers',
            name='Linea con puntos',
            marker = dict(color='blue', size=10),
            line=dict(color='blue',width=2, dash='dash')
        )
    ]
)
fig.show()
```



✓ Diseño (layout)

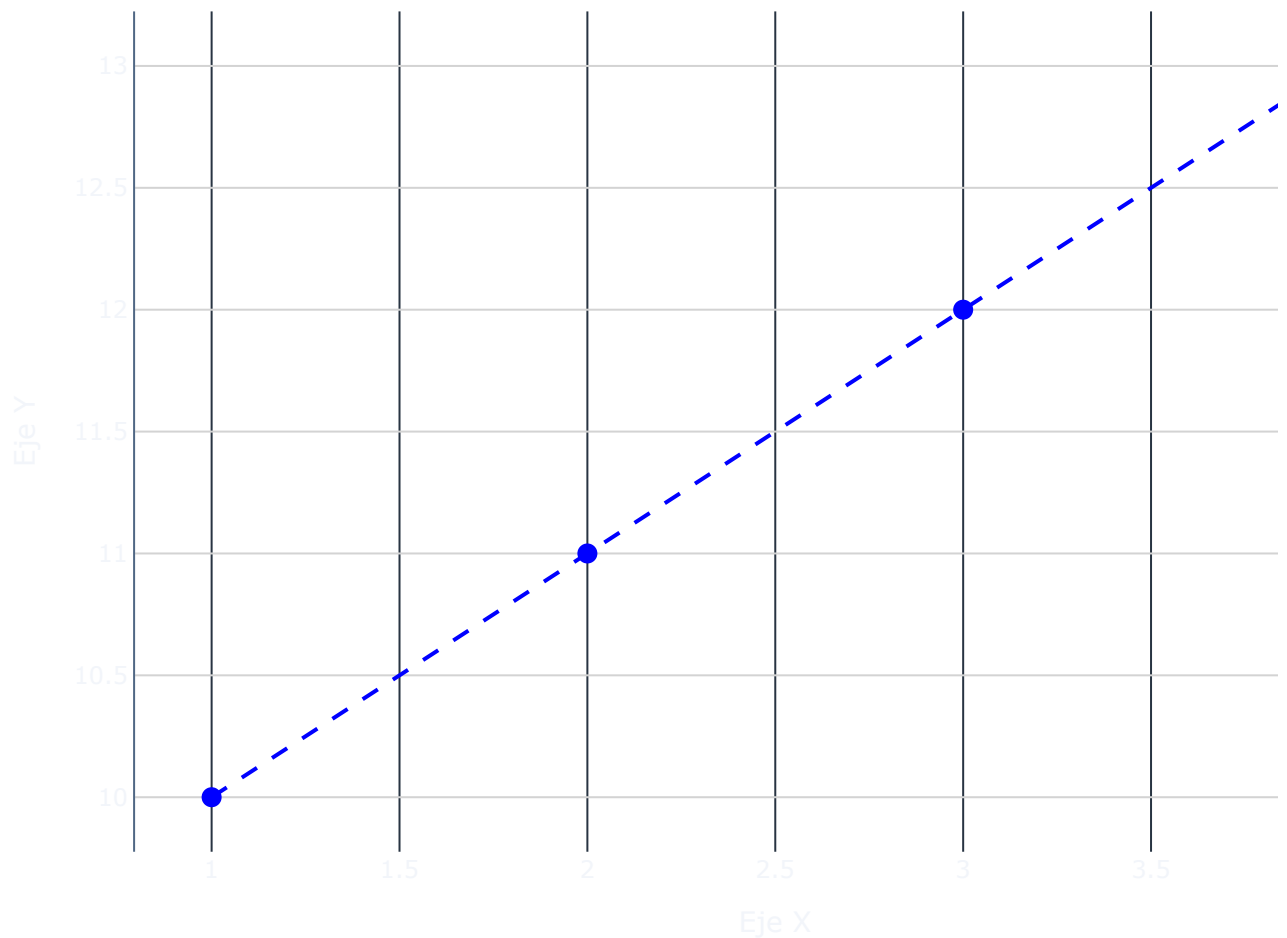
- El layout define el aspecto visula del gráfico.
- Propiedades clave en layout
 - title ---> Titulo del gráfico.
 - xaxis, yaxis ---> Configuración de los ejes.

- width, height ---> Tamaño del gráfico.
- template ---> Estilo visual del gráfico (ej. 'plotly_dark', 'ggplot2', 'seaborn').
- legend ---> Configuración de la leyenda.
- margin ---> controla los márgenes del gráfico.
- annotations ---> permite agregar etiquetas personalizadas.

```
fig.update_layout(
    title='Gráfico básico en Plotly',
    xaxis=dict(title='Eje X', showgrid=True, zeroline=False),
    yaxis=dict(title='Eje Y', showline=True, gridcolor='lightgray'),
    width=800,
    height=600,
    template='plotly_dark',
    legend=dict(
        x=0.02,
        y=0.98
    )
)
```



Gráfico básico en Plotly



✓ Configuración (config)

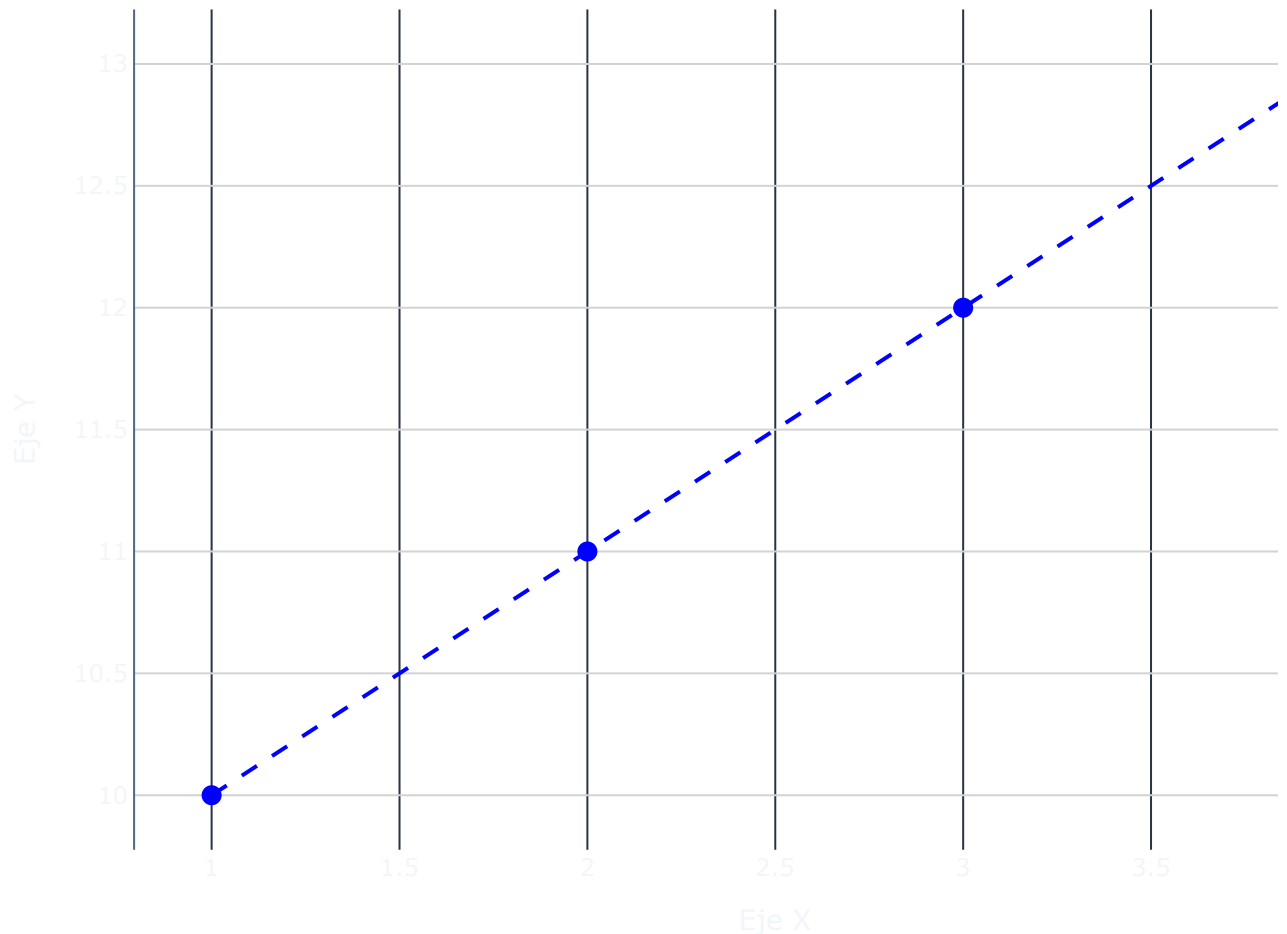
- El config permite controlar la interactividad y otras opciones del gráfico.
- Opciones clave en config
 - scrollZoom ---> Permite hacer zoom con la rueda del ratón.
 - displayModeBar ---> Muestra u oculta la barra de herramientas.
 - editable ---> Permite editar etiquetas directamente en el gráfico.
 - showTips ---> Activa o desactiva los tips interactivos.

```
fig.show(  
    config={  
        'scrollZoom': True,  
        'displayModeBar': True,  
        'editable': True,  
        'showTips': True  
    }  
)
```



Gráfico básico en Plotly

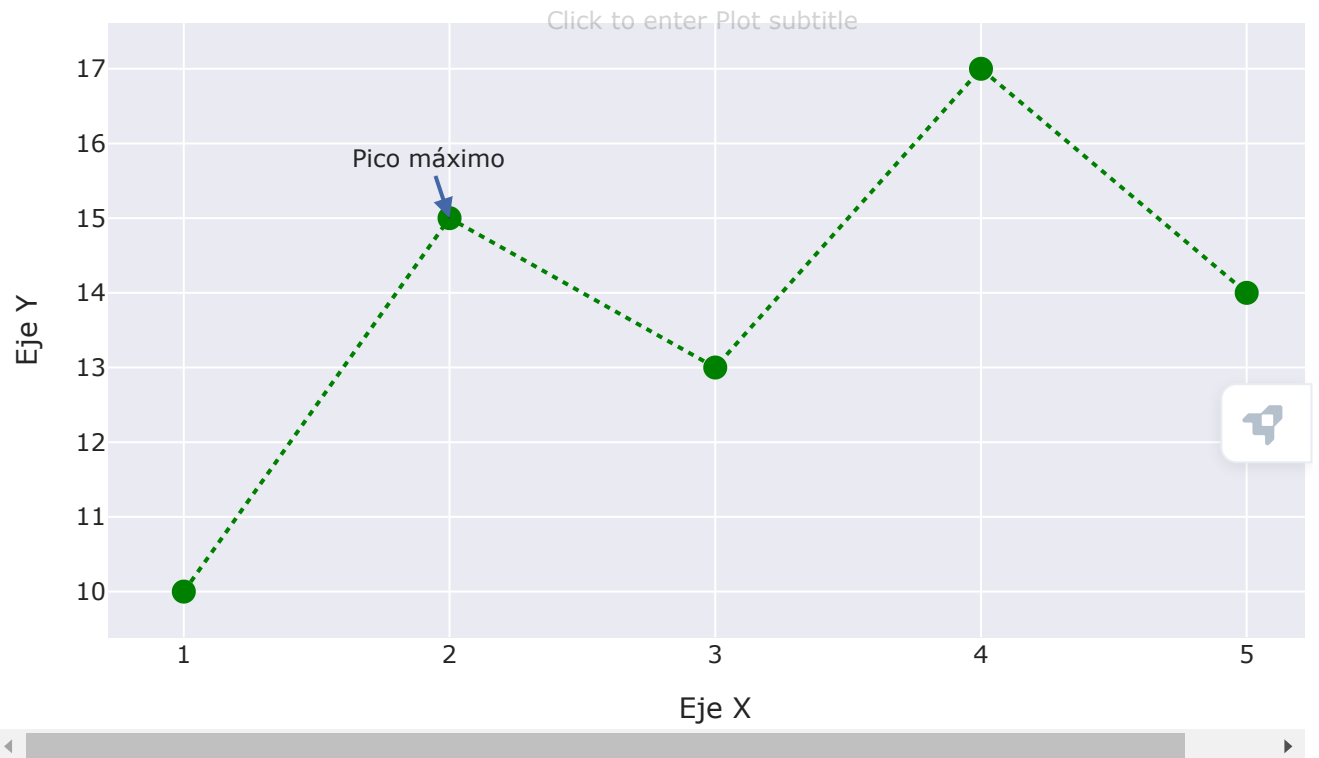
Click to enter Plot subtitle



✓ Ejemplo Completo con estructura Completa

```
fig = go.Figure(  
    data=[  
        go.Scatter(  
            x=[1, 2, 3, 4, 5],  
            y=[10, 15, 13, 17, 14],  
            mode='markers+lines',  
            name='Datos de ejemplo',  
            marker=dict(color='green', size=12),  
            line=dict(color='green', width=2, dash='dot')  
        )  
    ]  
)  
  
fig.update_layout(  
    title='Gráfico Personalizado en Plotly',  
    xaxis=dict(title='Eje X', showgrid=True),  
    yaxis=dict(title='Eje Y', showgrid=True),  
    width=700, # Ancho  
    height=400, # Alto  
    template='seaborn', # Estilo visual  
    legend=dict(x=0.02, y=0.98),  
    margin=dict(l=40, r=40, t=40, b=40), # Márgenes  
    annotations=[  
        dict(  
            x=2, y=15,  
            xref='x', yref='y',  
            text='Pico máximo',  
            showarrow=True,  
            arrowhead=2  
        )  
    ]  
)  
  
fig.show(config={  
    'scrollZoom': True,  
    'displayModeBar': True,  
    'editable': True,  
    'showTips': True  
})
```





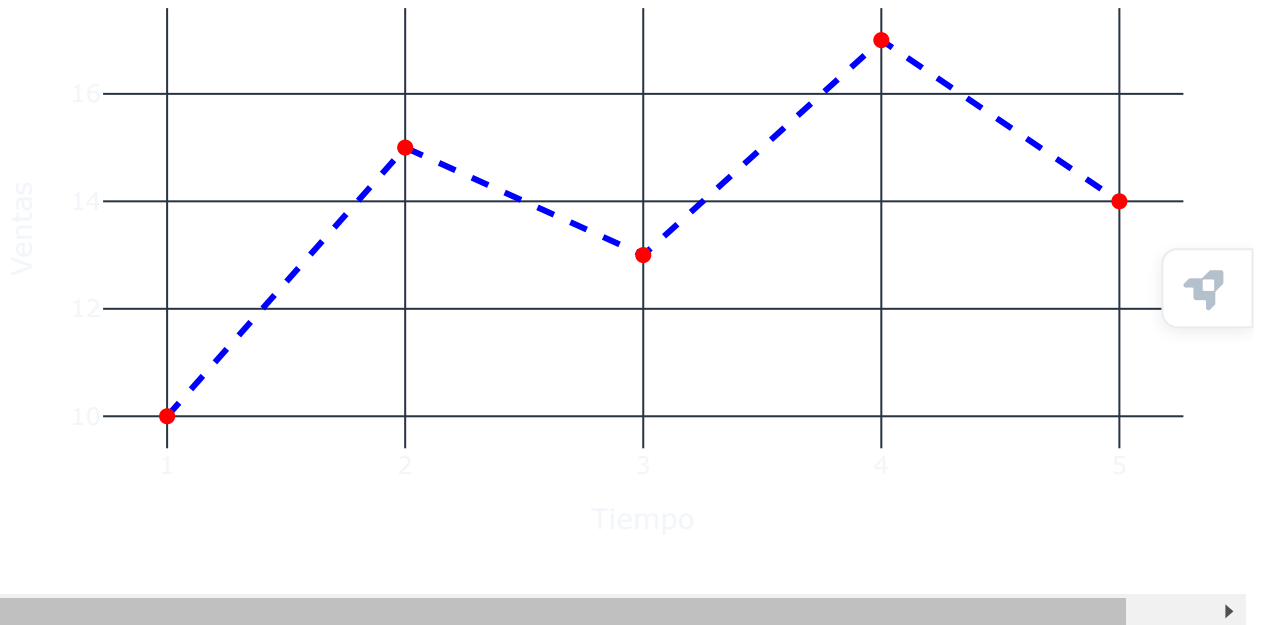
✓ Gráfico de Líneas (go.Scatter)

- El gráfico de líneas se utiliza para mostrar tendencias en el tiempo o en secuencias de datos.

```
fig = go.Figure(  
    data=[  
        go.Scatter(  
            x=[1,2,3,4,5],  
            y=[10,15,13,17,14],  
            mode='lines+markers',  
            name='Ventas',  
            marker=dict(color='red', size=8),  
            line=dict(color='blue', width=3, dash='dash')  
        )  
    ]  
)  
  
fig.update_layout(  
    title='Gráfico de Líneas',  
    xaxis=dict(title='Tiempo'),  
    yaxis=dict(title='Ventas'),  
    width=700,  
    height=400,  
    template='plotly_dark',  
    legend=dict(x=0.02, y=0.98)  
)  
  
fig.show()
```




Gráfico de Líneas



✓ Gráfico de Dispersión (go.Scatter)

- El gráfico de dispersión se utiliza para mostrar la relación entre dos variables.

```
import numpy as np

np.random.seed(42)

x = np.random.rand(100)
y = np.random.rand(100)

fig = go.Figure(
    data=[
        go.Scatter(
            x=x,
            y=y,
            mode='markers',
            marker=dict(
                size=10,
                color=np.random.rand(50),
                colorscale='Viridis',
                showscale=True
            )
        )
    ]
)

fig.update_layout(
    title='Gráfico de Dispersión',
```

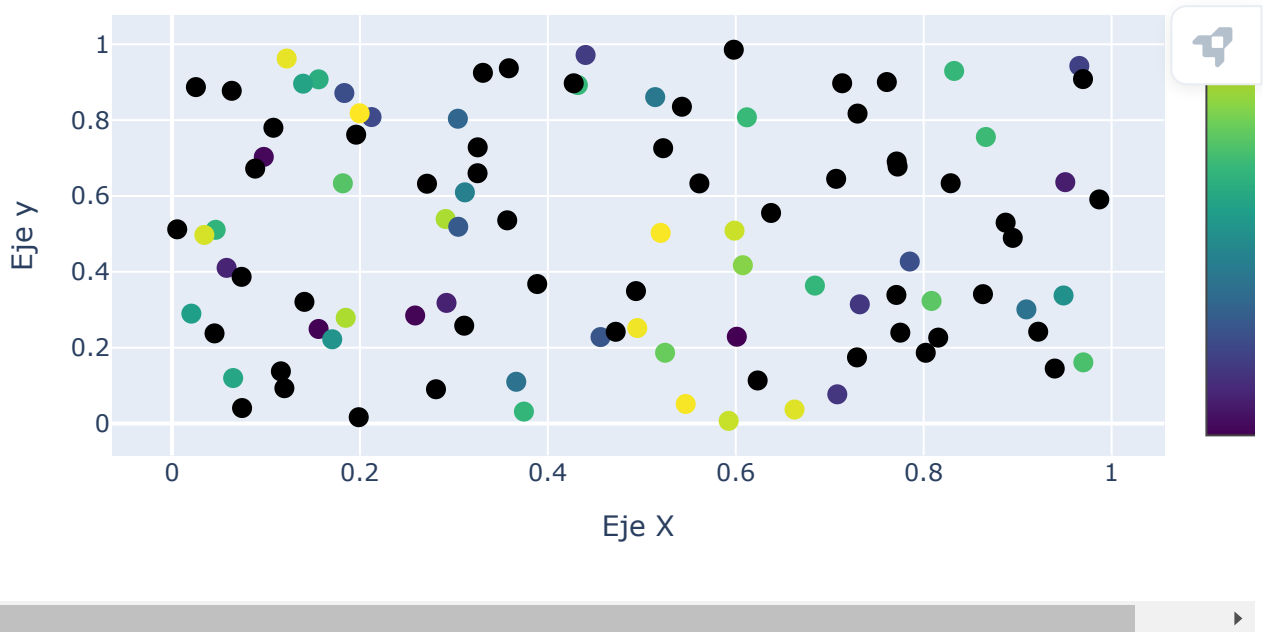
```

xaxis=dict(title='Eje X'),
yaxis=dict(title='Eje y'),
width=700,
height=400
)
fig.show()

```



Gráfico de Dispersión



✓ Gráfico de Barras (go.bar)

- El gráfico de barras se usa para comparar categorías o valores discretos

```

categorias = ['A', 'B', 'C', 'D']
valores = [20, 14, 23, 25]

fig = go.Figure(
    data=[
        go.Bar(
            x=categorias,
            y=valores,
            text=valores,
            textposition='auto',
            marker=dict(
                color='skyblue',
                line=dict(color='blue', width=2)
            )
        )
    ]
)

fig.update_layout(

```

```

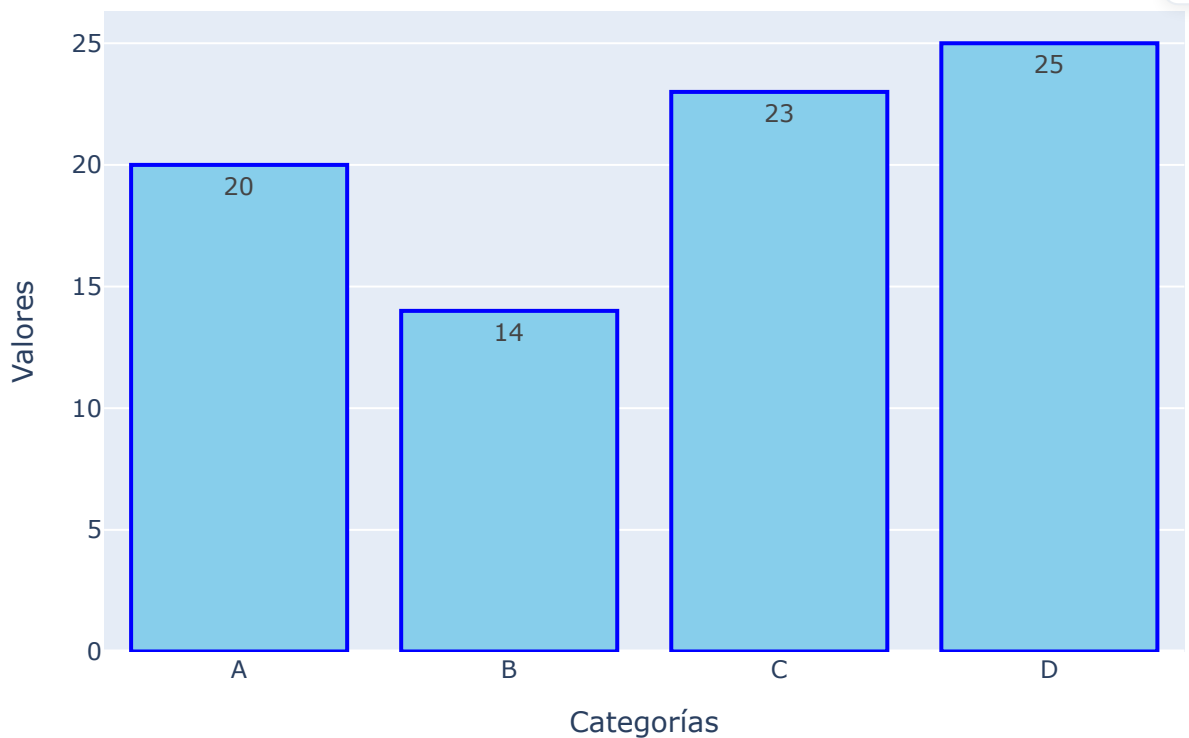
title='Gráfico de Barras',
xaxis=dict(title='Categorías'),
yaxis=dict(title='Valores'),
width=700,
height=500
)

fig.show()

```



Gráfico de Barras



✓ Gráfico de Pastel (go.Pie)

- El gráfico de pastel se usa para representar proporciones de un total.

```

etiquetas = ['Manzanas', 'Naranjas', 'Platanos', 'Uvas']
valores = [450, 300, 150, 100]

fig = go.Figure(
    data=[
        go.Pie(
            labels=etiquetas,
            values=valores,
            hole=0.3,
            pull=[0, 0.1, 0, 0]
        )
    ]
)

```

```

)

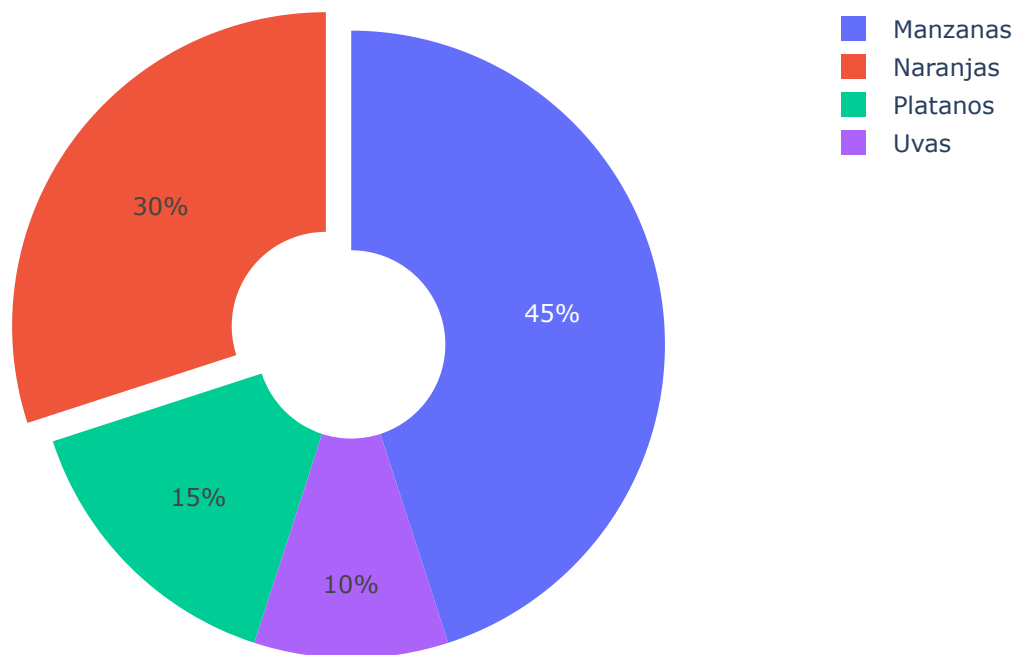
fig.update_layout(
    title='Graifico de Pastel'
)

fig.show(
    config={
        'scrollZoom': True
    }
)

```



Graifico de Pastel



✓ Gráfico de Caja (go.Box)

- El gráfico de caja se utiliza para mostrar la distribución de datos.

```

import numpy as np

np.random.seed(42)
datos = np.random.rand(100)

fig = go.Figure(
    data=[
        go.Box(

```

```

        y=datos,
        boxpoints='all',
        jitter=0.3,
        pointpos=-1.8
    )
]
)

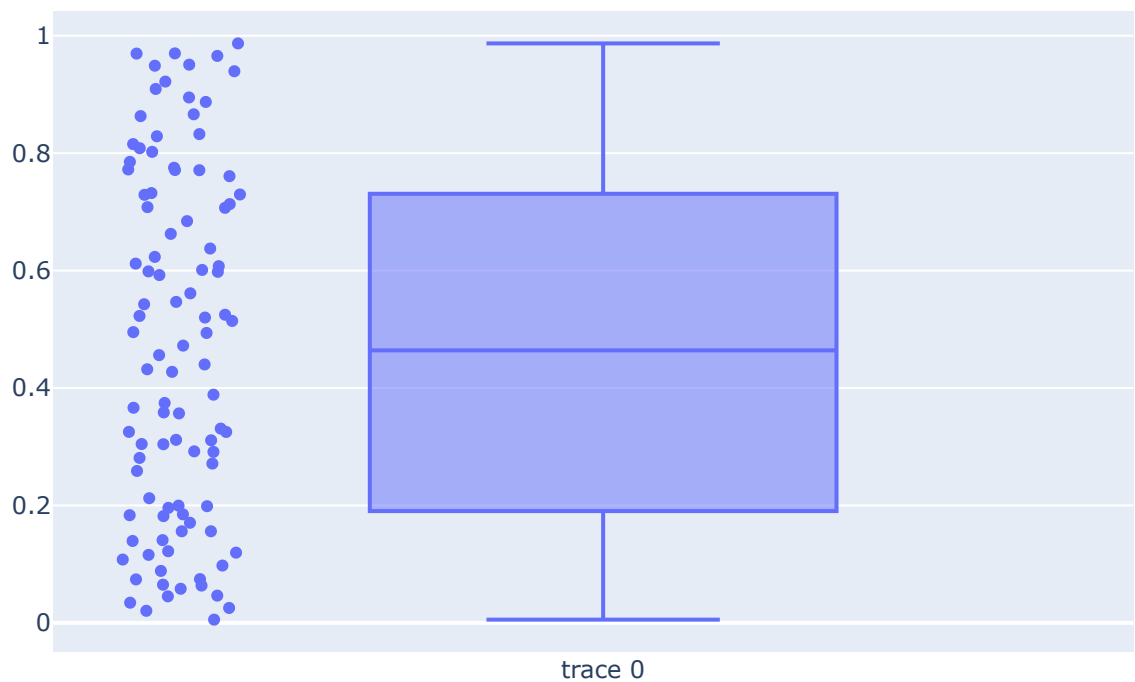
fig.update_layout(
    title='Gráfico de Caja',
    width=700,
    height=500
)

fig.show(
    config={
        'scrollZoom': True
    }
)

```



Gráfico de Caja



✓ Gráfico de Histograma (go.Histogram)

- El histograma muestra la distribución de un conjunto de datos.

```
import numpy as np

np.random.seed(42)
datos = np.random.randn(1000)

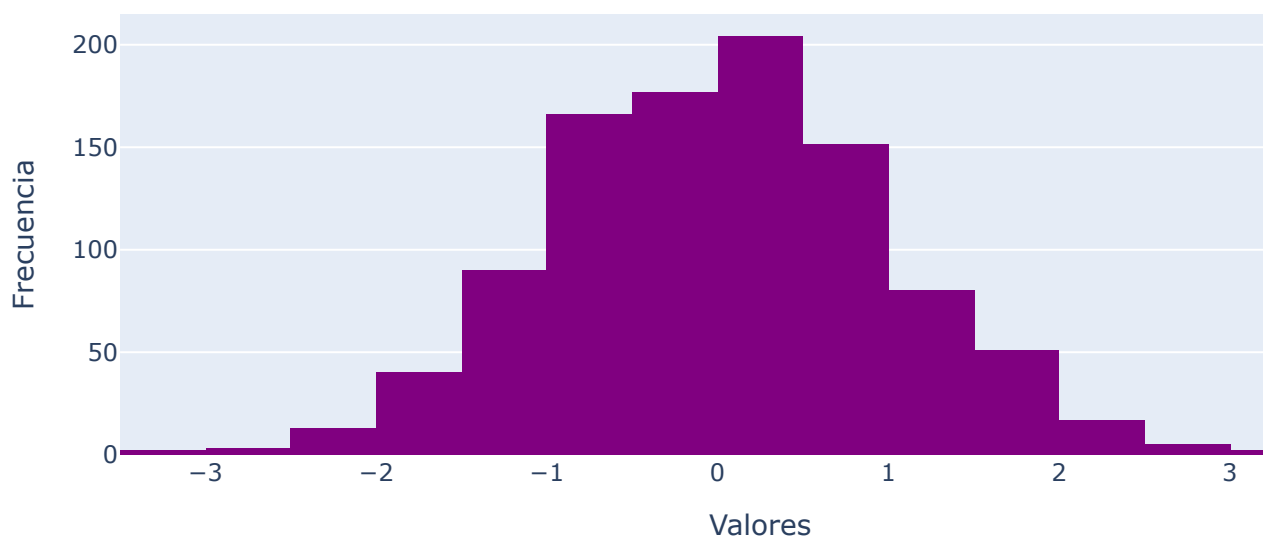
fig = go.Figure(
    data=[
        go.Histogram(
            x=datos,
            nbinsx=30,
            marker=dict(color='purple')
        )
    ]
)

fig.update_layout(
    title='Gráfico de Histograma',
    xaxis=dict(title='Valores'),
    yaxis=dict(title='Frecuencia'),
    width=800,
    height=400
)

fig.show(
    config={
        'scrollZoom': True,
    }
)
```



Gráfico de Histograma



✓ Tabla de elementos clave para gráficos en Plotly

Elemento	Descripción	Ejemplo de uso
x	Eje horizontal (valores, categorías)	x=[1,2,3,4]
y	Eje vertical (valores)	y=[10,15,20]
mode	Define el tipo de visualización	mode='lines+markers'
name	nombre de la serie en la leyenda	name='ventas Q1'
marker	Personaliza los puntos del gráfico (tamaño, color, borde)	marker=dict(size=10, color='red')
line	Personaliza las líneas (color, tipo, grosor)	line=dict(color='blue', width=3,dash='dash')
text	Agrega etiquetas de texto a cada punto del gráfico	text=['A', 'B', 'C']
textposition	Posición de las etiquetas 'top', 'bottom', 'left', 'right'	textposition='top right'
fill	Rellena el área bajo la curva. Opciones: 'tonexty', 'tozeroy'	fill='tozeroy'
opacity	Controla la transparencia del gráfico (valores entre 0 y 1)	opacity=0.5
hoverinfo	Información mostrada al pasar el curso	hoverinfo='x+y+text'
hovertext	Texto personalizado en el hover	hovertext=['Dia 1', 'Dia 2']
width	Anchura de las barras en gráficos de barra	width=0.5
hole	Tamaño del agujero en gráficos de pastel estilo dona	hole=0.4
pull	Resalta porciones en gráficos de pastel	pull=[0,0.1,0,0]
colorscale	Escala de colores para gráficos de dispersión o mapas de calor	coorscale='Viridis'
showlegend	Muestra u oculta la leyenda (True, False)	showlegend=True
orientation	Orientación del gráfico de barras: 'h', 'v'	orientation='v'
bargap	Espacio entre las barras en gráficos de barra	bargap=0.2
barmode	Controla la superposición de las barras: 'stack', 'group'	barmode='group'
autocolorscale	Permite que Plotly elija automáticamente la escala de colores	autocolorscale=True
pattern_shape	Agrega patrones a las barras	



```
import plotly.graph_objects as go
import plotly.io as pio
pio.renderers.default = 'colab'

fig = go.Figure(
    data=[
        go.Bar(
            x=['Producto A', 'Producto B', 'Producto C'],
            y=[350, 420, 280],
            text=['350 PS', '420 PS', '280 PS'],
            textposition='auto',
            marker=dict(
                color=['#FF5733', '#33FF57', '#3357FF'],
                line=dict(color='black', width=2),
                pattern_shape='/'
            )
        )
    ]
)

fig.update_layout(
    title='Ejemplo completo con varios elementos aplicados',
```

```

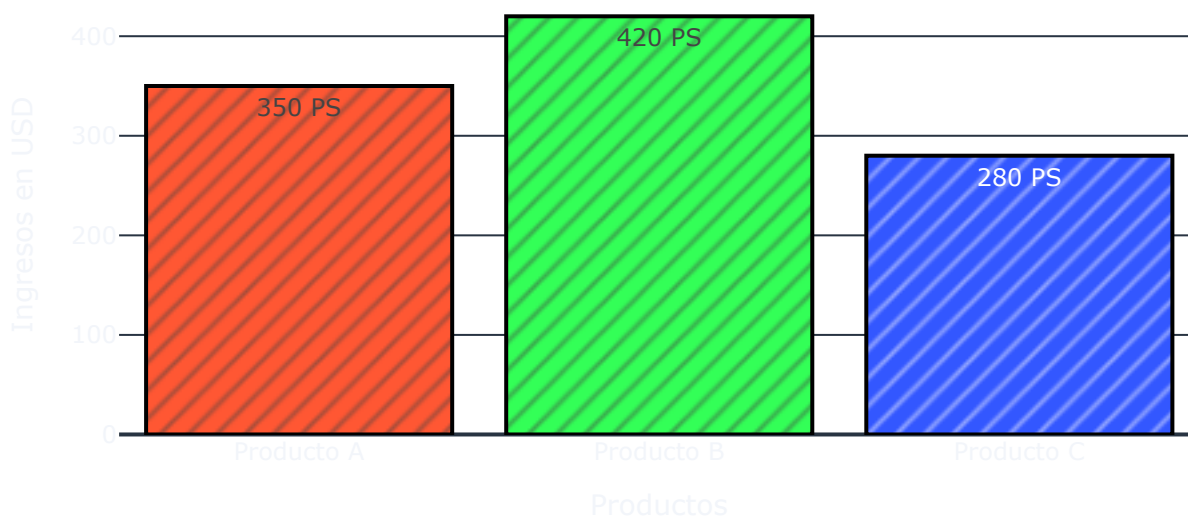
axis=dict(title='Productos'),
axis=dict(title='Ingresos en USD'),
width=700,
height=400,
template='plotly_dark',
bargap=0.15, # Espacio entre barras
barmode='group' # Agrupación de barras
)

# Mostrar el gráfico
fig.show()

```



Ejemplo completo con varios elementos aplicados



✓ Tabla de elementos para fig = go.Figure

Elemento	Descripción	Ejemplo de uso
x	Eje horizontal (valores, categorías)	x=[1,2,3,4]
y	Eje vertical (valores)	y=[10,15,20]
mode	Define el tipo de visualización	mode='lines+markers'
name	nombre de la serie en la leyenda	name='ventas Q1'
marker	Personaliza los puntos del gráfico (tamaño, color, borde)	marker=dict(size=10, color='red')
line	Personaliza las líneas (color, tipo, grosor)	line=dict(color='blue', width=3,dash='dash')
text	Agrega etiquetas de texto a cada punto del gráfico	text=['A', 'B', 'C']
textposition	Posición de las etiquetas 'top', 'bottom', 'left', 'right'	textposition='top right'
fill	Rellena el área bajo la curva. Opciones: 'tonexty', 'tozeroy'	fill='tozeroy'
opacity	Controla la transparencia del gráfico (valores entre 0 y 1)	opacity=0.5
hoverinfo	Información mostrada al pasar el curso	hoverinfo='x+y+text'
hovertext	Texto personalizado en el hover	hovertext=['Dia 1', 'Dia 2']

Elemento	Descripción	Ejemplo de uso
width	Ach de las barras en gráficos de barra	width=0.5

Elementos para fig.update.layout()

Elemento	Descripción	Ejemplo de uso
title	Título principal del gráfico	title='Gráficoico'
xaxis	Configuración del eje X	xaxis=dict(title='Meses')
yaxis	Configuración del eje Y	yaxis=dict(title='Ventas en \$')
width	Ancho del gráfico	width=700
height	Alto del gráfico	height=400
template	Tema visual del gráfico	template='plotly_dark'
bargap	Espacio entre las barras en graficos de barras	bargap=0.2
barmode	Tipo de agrupación de barras 'stack', 'group'	barmode='group'
showlegend	Muestra u oculta la leyenda	showlegend=True
legend	personalización avanzada de la leyenda (posición, orientación)	legend=dict(x=1, y=1)
margin	Ajusta los márgenes del gráfico	margin=dict(l=40, r=40, t=40, b=40)



✓ Elementos para fig.show()

Elemento	Descripción	Ejemplo de uso
pio.renderers.default	Define el entorno de renderizado(colab, Jupyter)	pio.renderers.default='colab'
config	Permite ajustar opciones interactivas del gráfico	fig.show(config={'scrollZoom'})
auto_open	Controla si el grafico se abre automaticamente	

```
import plotly.graph_objects as go
import plotly.io as pio
pio.renderers.default = 'colab'

fig = go.Figure(
    data=[
        go.Scatter(
            x=[1, 2, 3, 4, 5],
            y=[10, 15, 13, 17, 14],
            mode='lines+markers',
            name='Ventas',
            marker=dict(color='red', size=10),
            line=dict(color='blue', width=3, dash='dash')
        )
    ]
)

fig.update_layout(
    title='Gráfico de Ventas, Usando varios elementos',
    xaxis=dict(title='Meses'),
    yaxis=dict(title='Ventas en USD'),
    width=700,
```

```

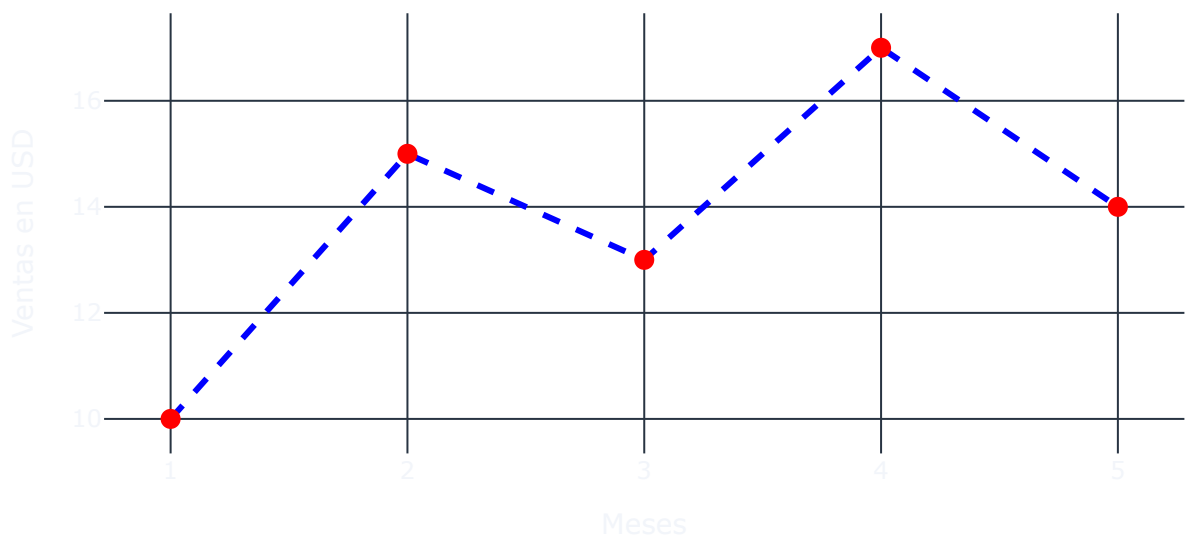
height=400,
template='plotly_dark'
)

fig.show(
    config={'scrollZoom': True}
)

```



Gráfico de Ventas, Usando varios elementos



✓ Opciones disponibles para template

- Plotly tiene una amplia variedad de plantillas predefinidas que se pueden consultar con este comando.
- Disponibles
 1. plotly ---> por defecto
 2. plotly_dark ---> tema oscuro
 3. ggplot2 ---> similar al estilo de ggplot en R
 4. seaborn ---> inspirado en la librería Seaborn
 5. simple_white ---> estilo limpio y minimalista
 6. presentatio ---> ideal para presentaciones
 7. xgridoff, ygridoff, gridon ---> control de líneas de cuadrícula
 8. none ---> sin estilo predefinido

```

import plotly.io as pio
pio.templates

```



Templates configuration

Default template: 'plotly'

Available templates:

```
['ggplot2', 'seaborn', 'simple_white', 'plotly',  
'plotly_white', 'plotly_dark', 'presentation', 'xgridoff',  
'ygridoff', 'gridon', 'none']
```

Opciones disponibles para barmode

- El parámetro barmode se utiliza para definir cómo se agrupan o apilan las barras en gráficos de barras
- Opciones para barmode
 1. group ---> Agrupadas por defecto
 2. stack ---> Apiladas
 3. overlay ---> Superpuestas
 4. relative ---> Similar a stack, pero ajusta valores negativos para iniciar desde cero



Opciones disponibles para mode (en gráficos de dispersión o líneas)


- el parámetro mode define el estilo de visualización en gráficos tipo de Scatter.
- Opciones para mode
 1. lines ---> Solo líneas
 2. markers ---> Solo puntos
 3. lines+markers ---> Combina líneas y puntos
 4. text ---> Solo etiquetas de texto
 5. lines+text, markers+text, lines+markers+text ---> Combinaciones personalizadas



Opciones disponibles para colorscale (en mapas de color y gráficos de dispersión)

- El parametro colorscale define las escalas de color que puedes aplicar.
- Opciones para colorscale
 1. Viridis ---> Escala en tonos morado-amarillo
 2. Cividis ---> Escala azul-amarilla, diseñada para ser legible por personas con daltonismo
 3. Inferno ---> Escala cálida con tonos oscuros y brillantes
 4. Magma ---> Escala intensa en tonos morado-naranja
 5. Plasma ---> Escala morado-amarilla con alto contraste
 6. turbo ---> Escala multicolor vibrante
 7. Jet, hot, Cool ---> Colores clásicos de visualización científica

```
import plotly.express as px
px.colors.named_colorscale()
```



```
'purples',
'purpor',
'rainbow',
'rdbu',
'rdpu',
'redor',
'reds',
'sunset',
'sunsetdark',
'teal',
'tealgrn',
'turbo',
'viridis',
'ylgn',
'ylgnbu',
'ylorbr',
'ylorrd',
'algae',
'amp',
'deep',
'dense',
```



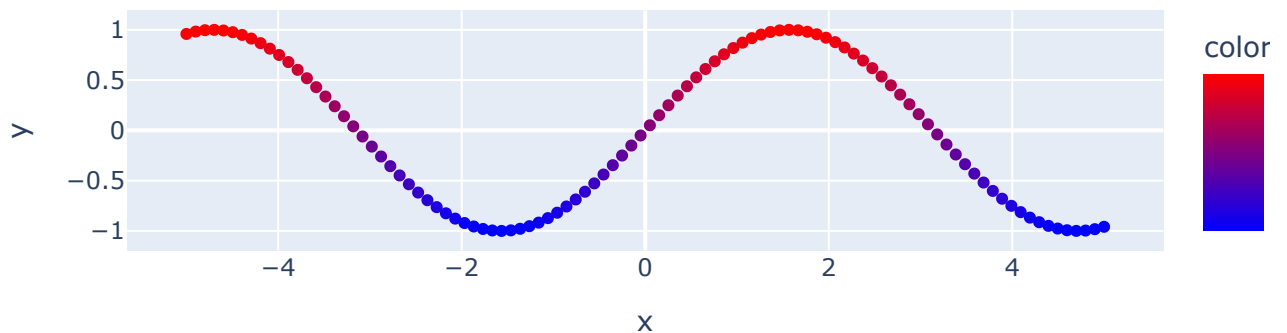
```
import plotly.express as px
import numpy as np

# Datos para el gráfico
x = np.linspace(-5, 5, 100)
y = np.sin(x)

# Gráfico de dispersión con escala de color 'Viridis'
fig = px.scatter(x=x, y=y, color=y, color_continuous_scale='bluered', width=700,
                 height=300, title='Grafico de Dispersion')
fig.show()
```



Grafico de Dispersion



✓ Temas de Personalización de Gráficos

✓ Títulos y etiquetas de ejes

- Se agrega con `fig.update_layout()`
- Personalización de tamaño, color, posición, etc.

```
import plotly.graph_objects as go
import plotly.io as pio
pio.renderers.default = 'colab'
```

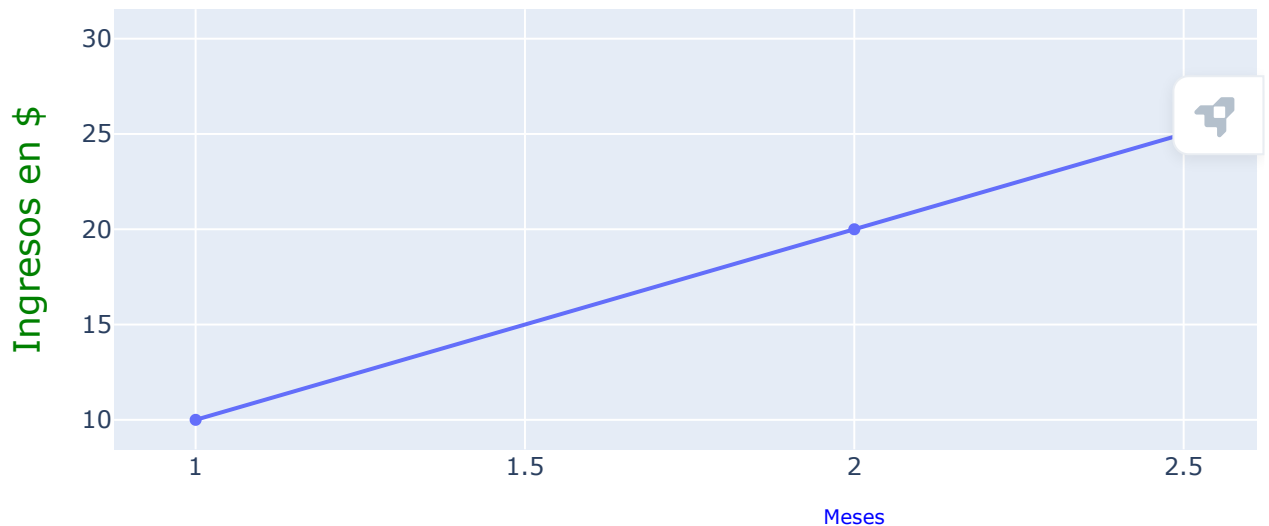
```
fig = go.Figure(data=go.Scatter(x=[1,2,3], y=[10,20,30]))

fig.update_layout(
    title='Ventas mensuales',
    xaxis=dict(title='Meses', title_font=dict(size=10, color='blue')),
    yaxis=dict(title='Ingresos en $', title_font=dict(size=18, color='green')),
    width=900,
```

```
height=400
)
fig.show()
```



Ventas mensuales



Tamaño del gráfico

- Controlando con width y height dentro de fig.update_layout()

fig.update_layout(width=800, height=500)

✓ Colores personalizados

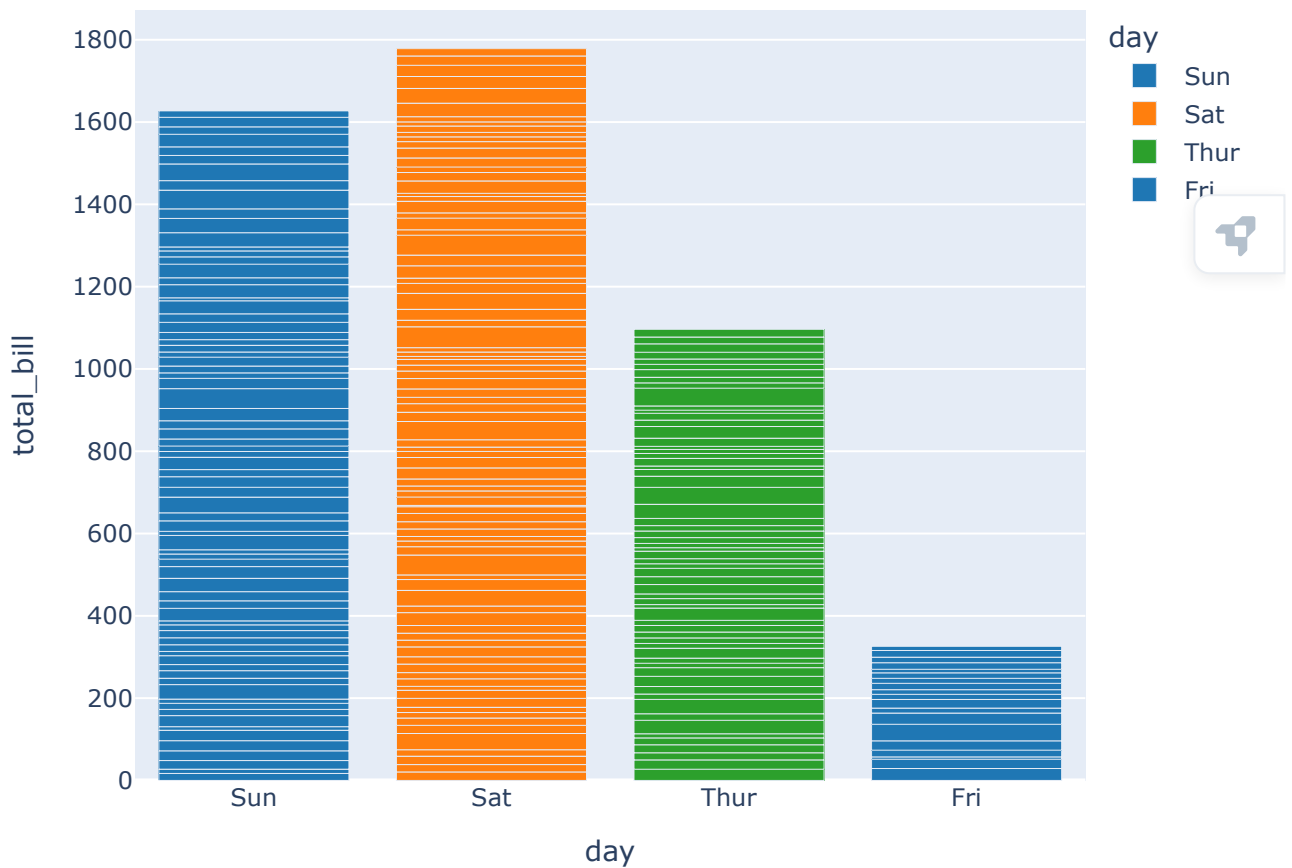
- Se pueden aplicar colores especificos en elementos como:
 - Lineas ---> line=dict(color='red')
 - Marcadores ---> marker=dict(color='green')
 - Barras ---> marker_color='purple'

. Para escalas de color, se usa el paramero colorscale.

```
import plotly.express as px

df = px.data.tips()
fig = px.bar(
    df,
    x='day',
    y='total_bill',
    color='day',
```

```
color_discrete_sequence=['#1f77b4', '#ff7f0e', '#2ca02c']
)
fig.show()
```



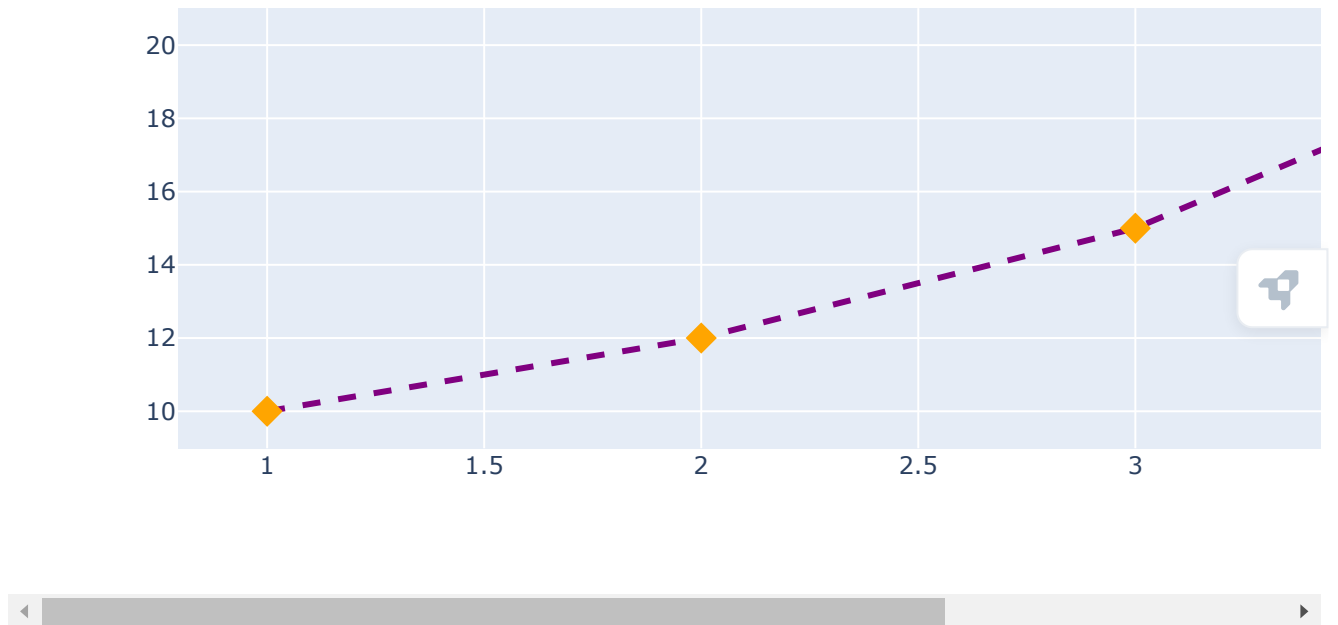
✓ Estilo de líneas y marcadores

- Líneas ---> line=dict(dash='dash', width=3)
- Marcadores ---> marker=dict(size=10, symbol='diamond')

```
fig = go.Figure(data=go.Scatter(
    x=[1,2,3,4],
    y=[10,12,15,20],
    mode='lines+markers',
    line=dict(dash='dash', width=3, color='purple'),
    marker=dict(size=12, color='orange',symbol='diamond')
))

fig.update_layout(
    width=900,
    height=400
)

fig.show()
```



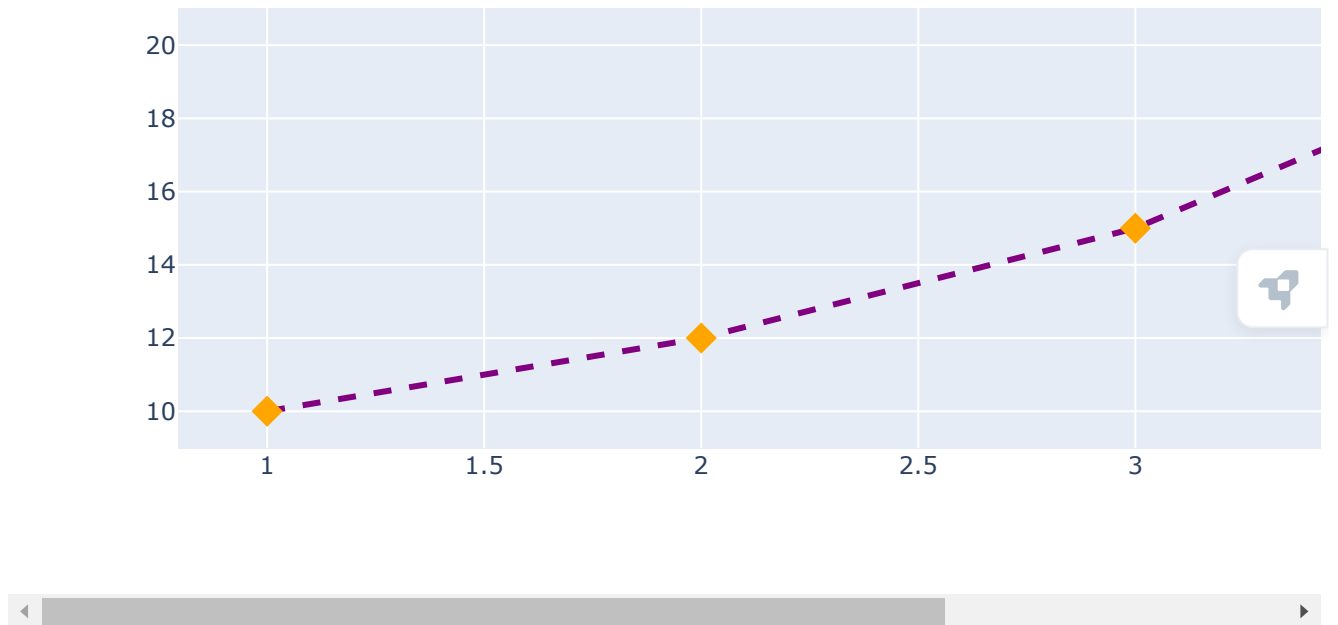
✓ Formato de textos y fuentes

- Se puede personalizar el tipo de letra, tamaño, color, etc.

```
fig.update_layout(  
    title=dict(  
        text='Titulo personalizado',  
        font=dict(family='Arial', size=14, color='darkblue')  
    )  
)
```




Titulo personalizado



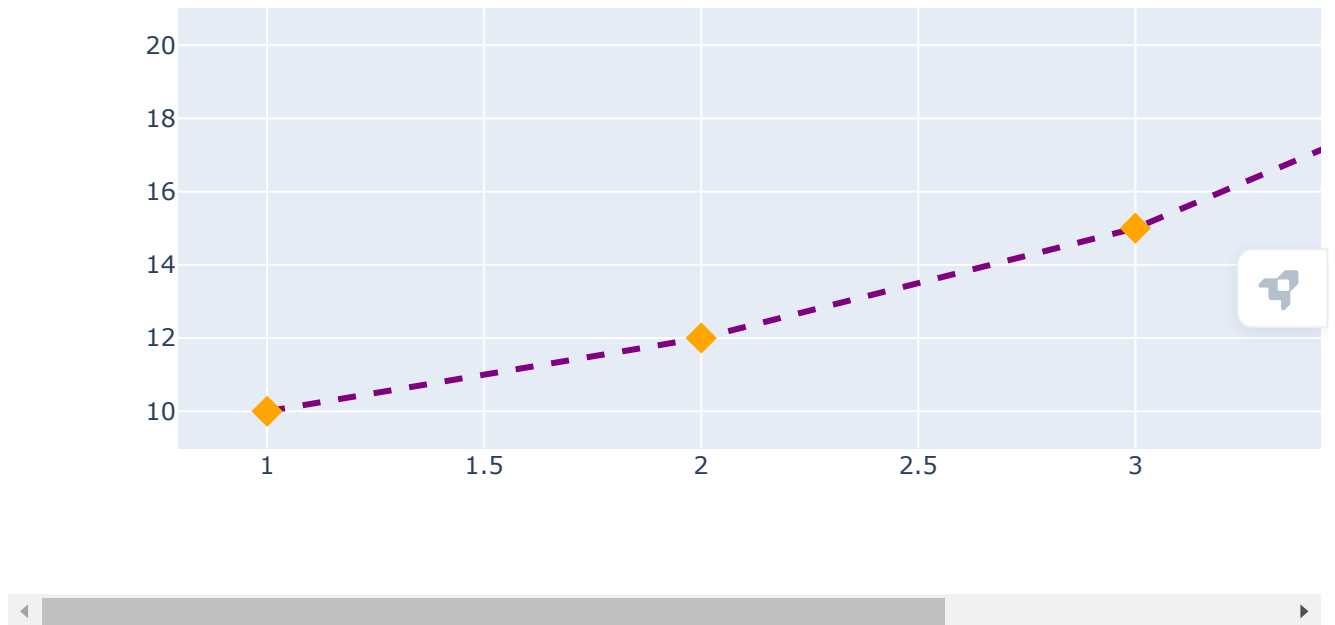
✓ Leyenda personalizada

- Se controla con `fig.update_layout()` usando el parametro `legend`

```
fig.update_layout(  
    legend=dict(  
        title='Categorias',  
        x=1,  
        y=1,  
        bgcolor='lightgray'  
    )  
)
```



Titulo personalizado



✓ Ejes secundarios y escalas logarítmicas.

- Los ejes secundarios se agregan con `secondary_y=True`.
- Escalas logarítmicas se agrega con `type='log'`.

```
from plotly.subplots import make_subplots

fig = make_subplots(specs=[[{'secondary_y':True}]]))

fig.add_trace(go.Bar(
    x=[1,2,3],
    y=[40,60,80],
    name='Ventas'
), secondary_y=False)

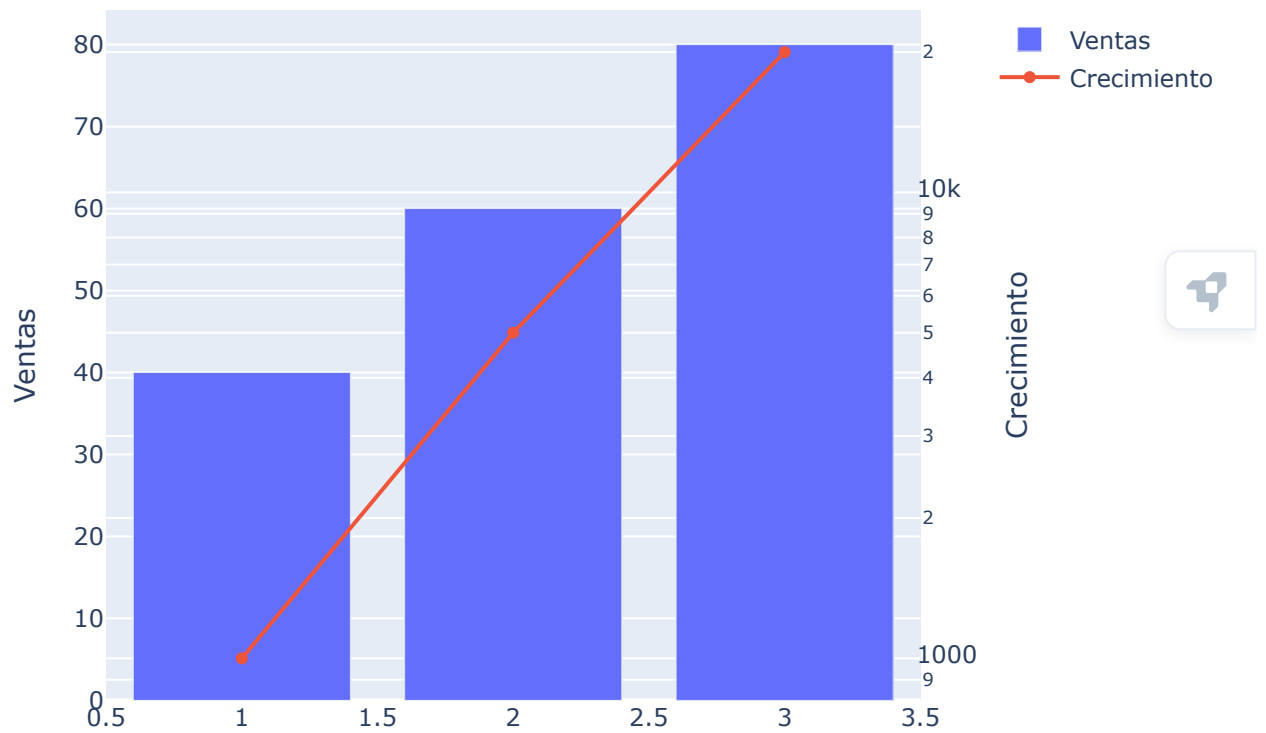
fig.add_trace(go.Scatter(
    x=[1,2,3],
    y=[1000,5000,20000],
    name='Crecimiento',
    mode='lines+markers'
), secondary_y=True)

fig.update_layout(title='Comparación de Datos')
fig.update_yaxes(title_text='Ventas', secondary_y=False)
fig.update_yaxes(title_text='Crecimiento', secondary_y=True, type='log')

fig.show()
```



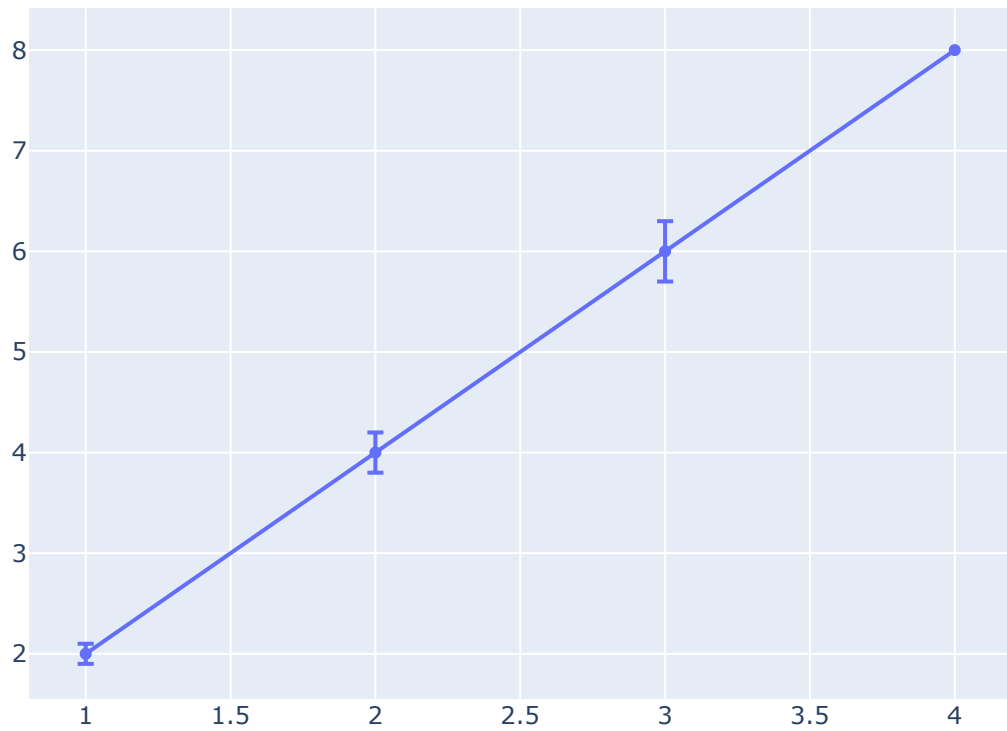
Comparación de Datos



✓ Barras de error y visualización avanzada

- Se usa `error_y` o `error_x` para agregar márgenes de error.

```
fig = go.Figure(  
    data=go.Scatter(  
        x=[1,2,3,4],  
        y=[2,4,6,8],  
        error_y=dict(type='data', array=[0.1, 0.2, 0.3])  
    )  
)  
fig.show()
```



✓ Anotaciones y formas (shapes)

- Se agrega con annotations y shapes

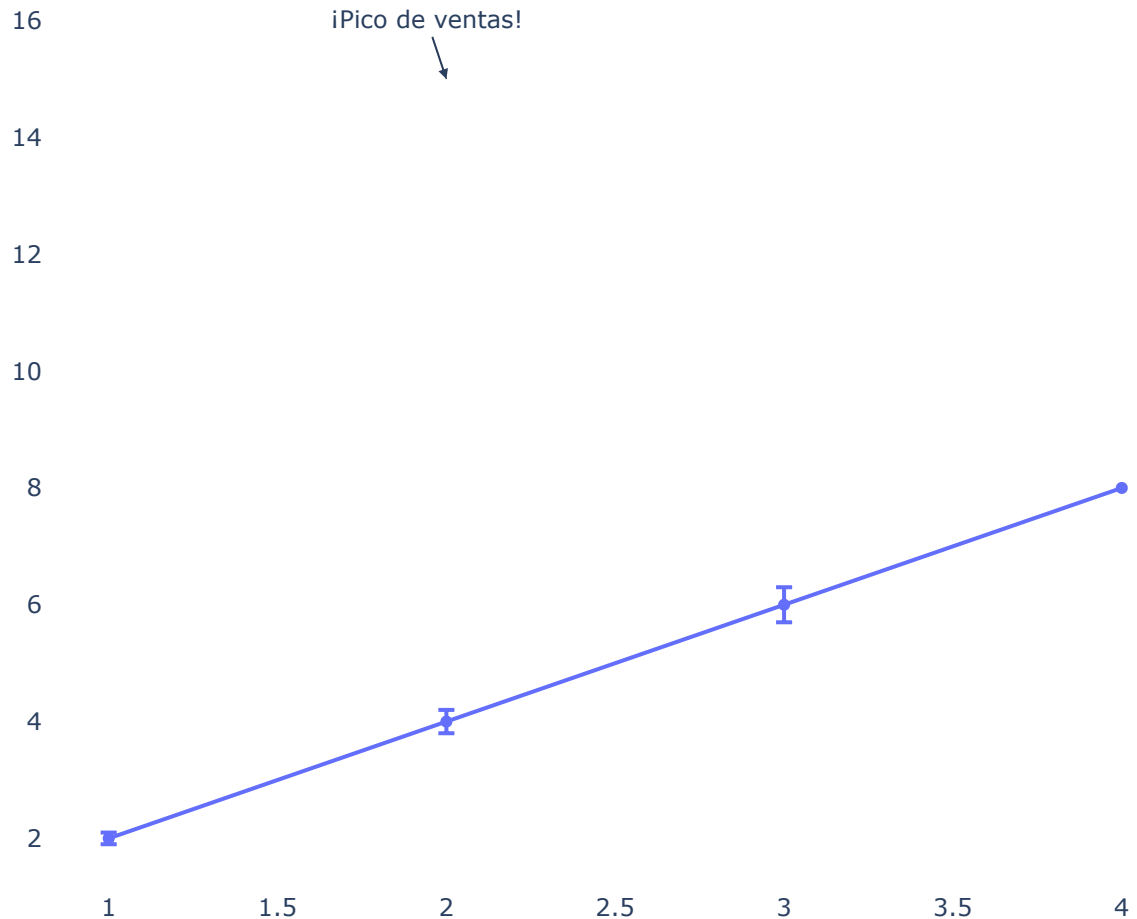
```
fig.update_layout(  
    annotations=[  
        dict(  
            x=2, y=15, text='¡Pico de ventas!', showarrow=True, arrowhead=2  
        )  
    ]  
)
```



✓ Configuración del fondo y márgenes

- Se configura con `plot_bgcolor`, `paper_bgcolor` y `margin`

```
fig.update_layout(  
    plot_bgcolor='rgba(0, 0, 0, 0)',  
    paper_bgcolor='lightblue',  
    margin=dict(l=40, r=40, t=40, b=40)  
)
```



✓ Temas de gráficos Avanzados con Plotly Express

1. Gráficos de dispersión en 3D.
2. Gráficos de líneas avanzados.
3. Mapas de calor (heatmaps).
4. Gráficos de caja (box plot) para análisis estadístico.
5. Gráficos de violín para distribución de datos.
6. Gráficos de pastel (pie charts) y de dona (donut charts).
7. Gráficos geoespaciales (mapas interactivos).
8. Gráficos de facetas (facet grids) para comparaciones multiples.
9. Gráficos animados para mostrar evolución temporal.
10. Combinación de gráficos avanzados.

✓ Gráficos de dispersión en 3D

- Los gráficos 3D permiten visualizar datos con tres variables.

```
import plotly.express as px
import numpy as np
```

```
import pandas as pd

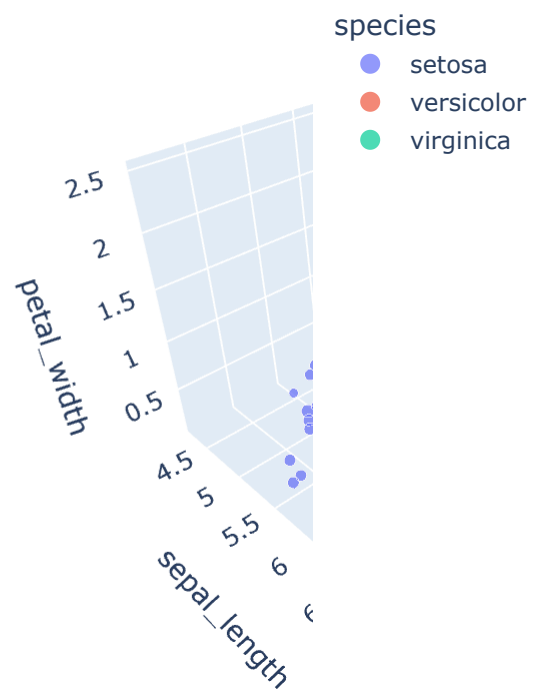
df = px.data.iris()

fig = px.scatter_3d(
    df,
    x='sepal_width',
    y='sepal_length',
    z='petal_width',
    color='species',
    size='petal_length',
    title='Dispesión 3D de flores Iris'
)

fig.show()
```



Dispesión 3D de flores Iris



✓ Gráficos de líneas avanzadas

- Puedes agregar múltiples líneas, sombrear áreas y controlar el estilo

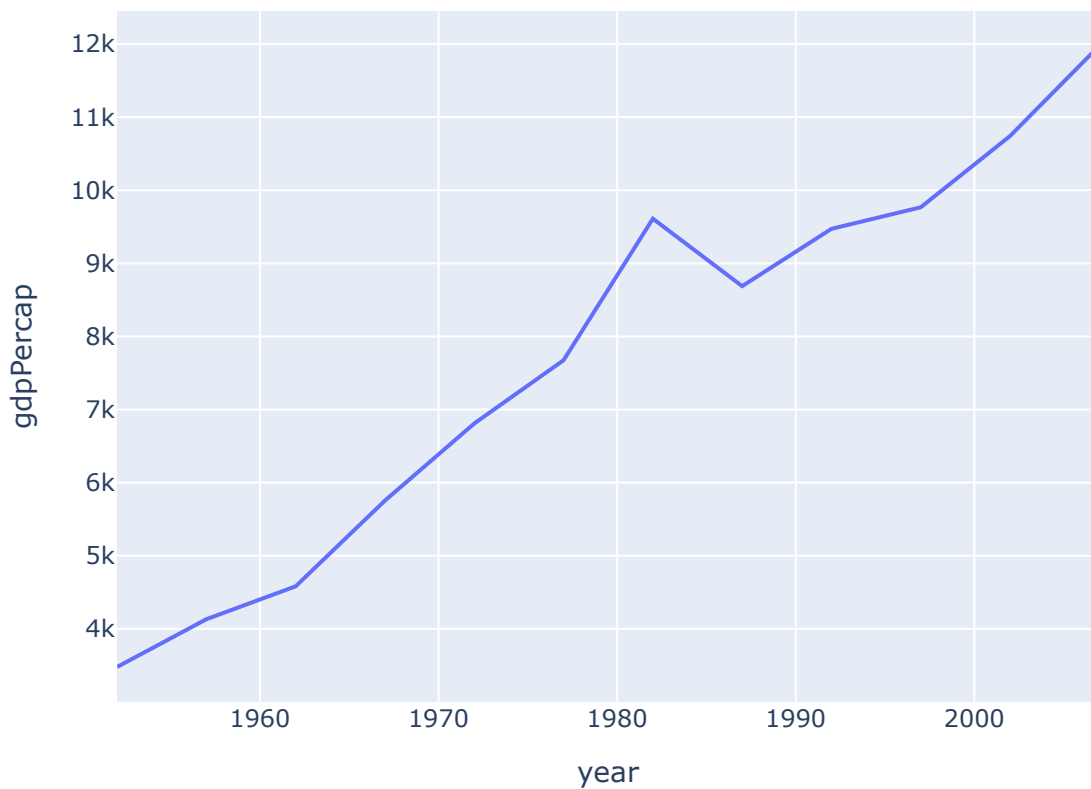
```
import plotly.express as px

df1 = px.data.gapminder()
```

```
fig = px.line(
    df1[df1['country'] == 'Mexico'],
    x='year', y='gdpPercap',
    title='Crecimiento del PIB en México'
)
fig.show()
```



Crecimiento del PIB en México



Mapas de calor (Heatmaps)

- Los mapas de calor son ideales para identificar patrones en datos matriciales.

```
import plotly.express as px
import numpy as np
import pandas as pd

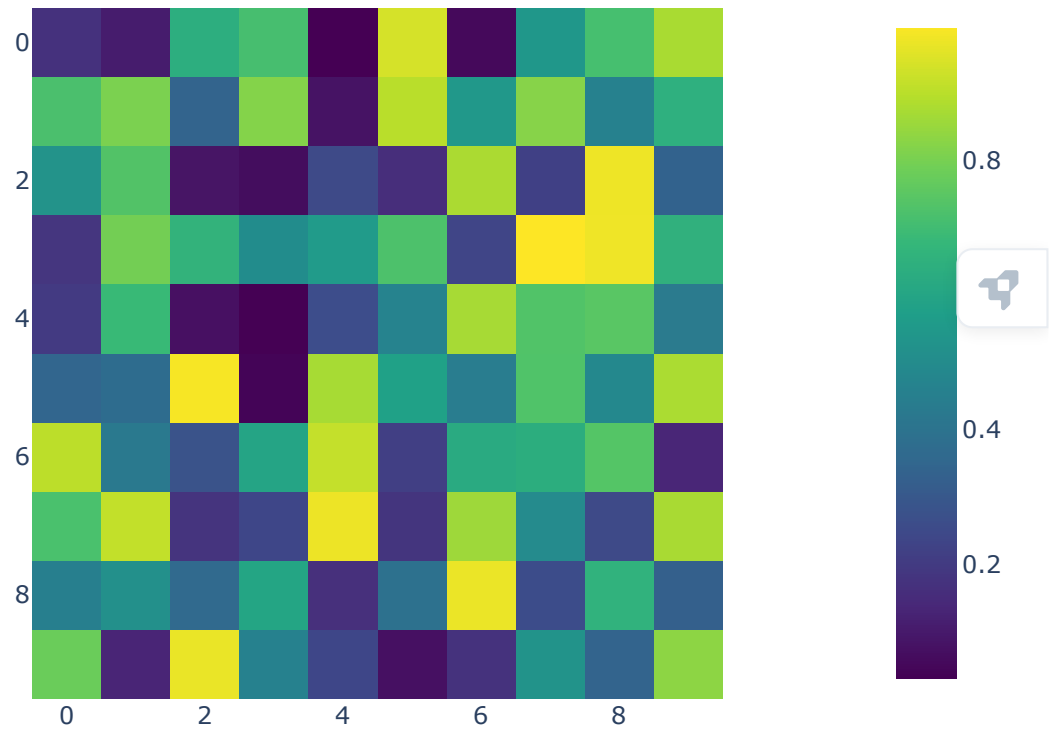
data = np.random.rand(10,10)

fig = px.imshow(
    data,
    color_continuous_scale='Viridis',
    title='Mapa de calor'
)

fig.show()
```




Mapa de calor



✓ Gráficos de caja (Box Plot)

- Ideales para detectar outliers y distribuciones de datos.

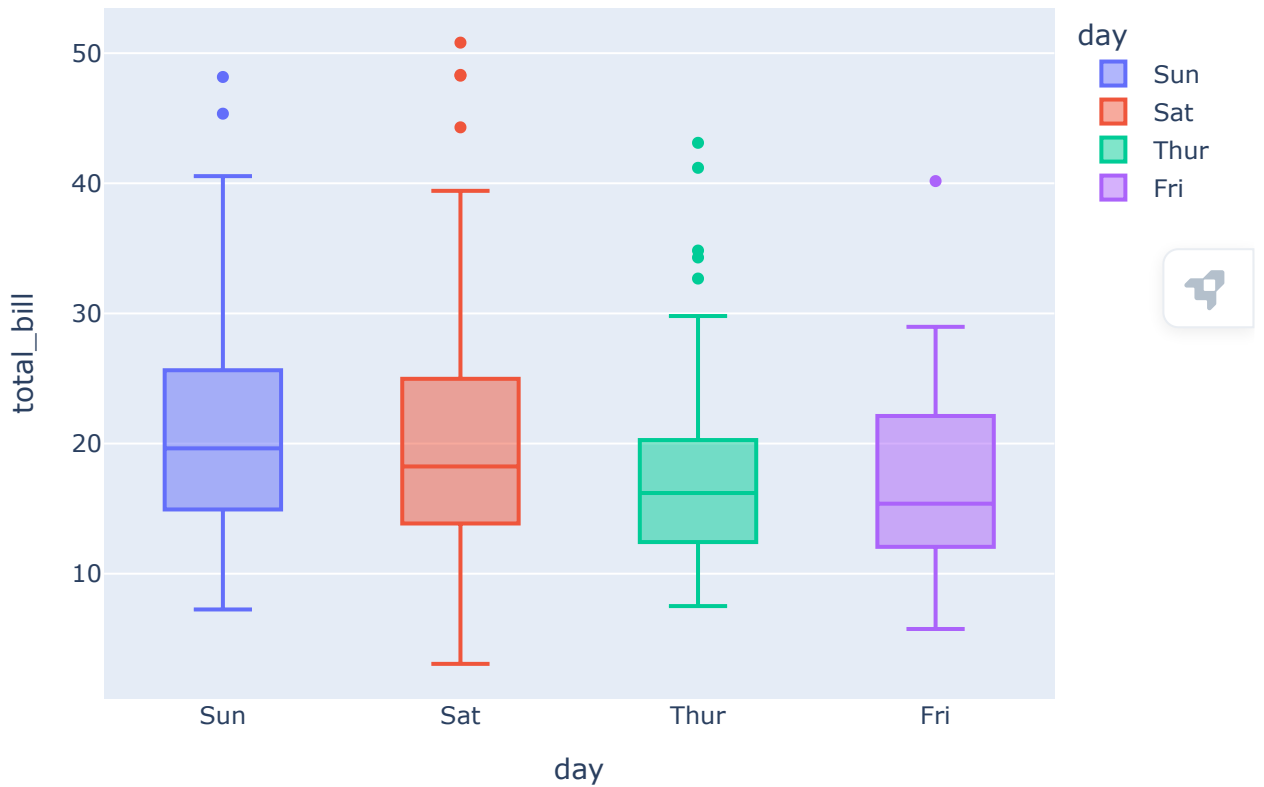
```
df2 = px.data.tips()

fig = px.box(
    df2,
    x='day',
    y='total_bill',
    color='day',
    title='Distribución de cuentas por día'
)

fig.show()
```



Distribución de cuentas por día



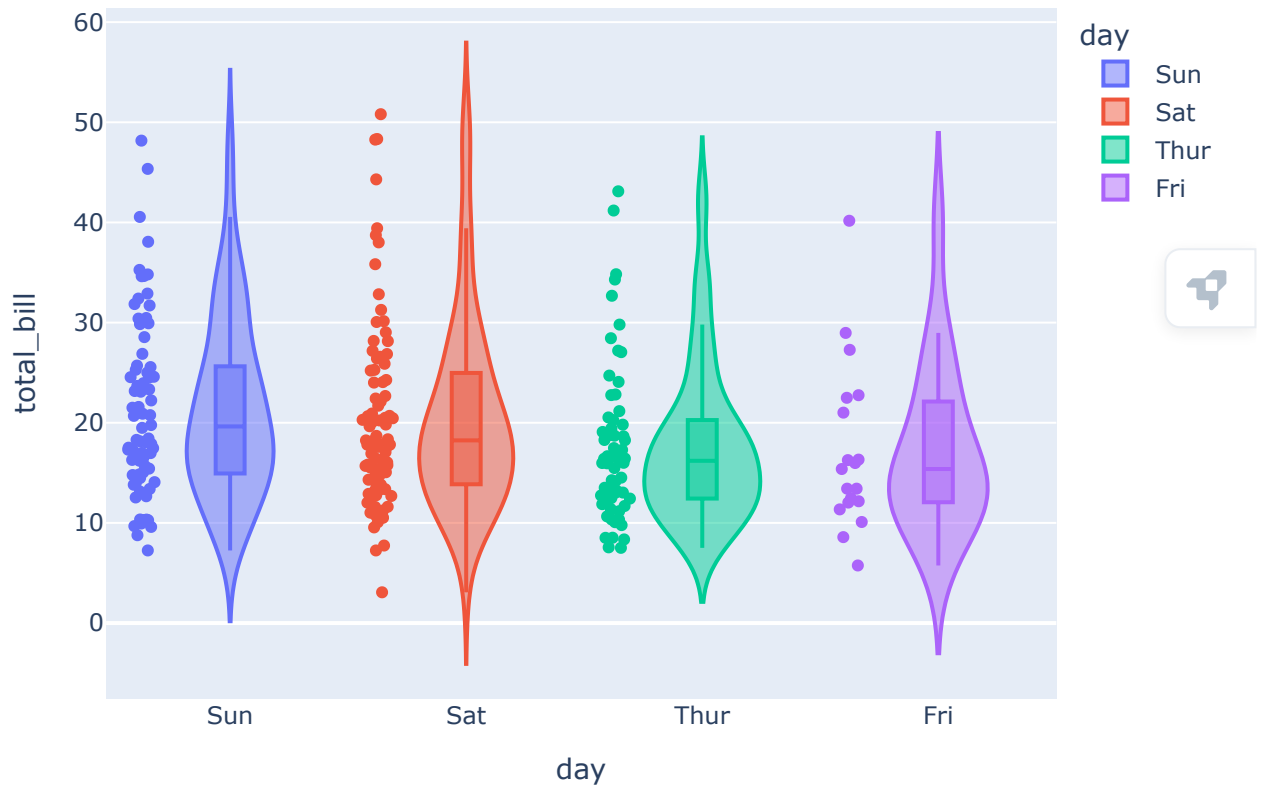
✓ Gráficos de violín

- parecidos a los box plots pero cuna una mejor representación de densidad

```
fig = px.violin(df2,  
                x='day',  
                y='total_bill',  
                color='day',  
                box=True,  
                points='all',  
                title='Gráfico de violín')  
  
fig.show()
```



Gráfico de violín



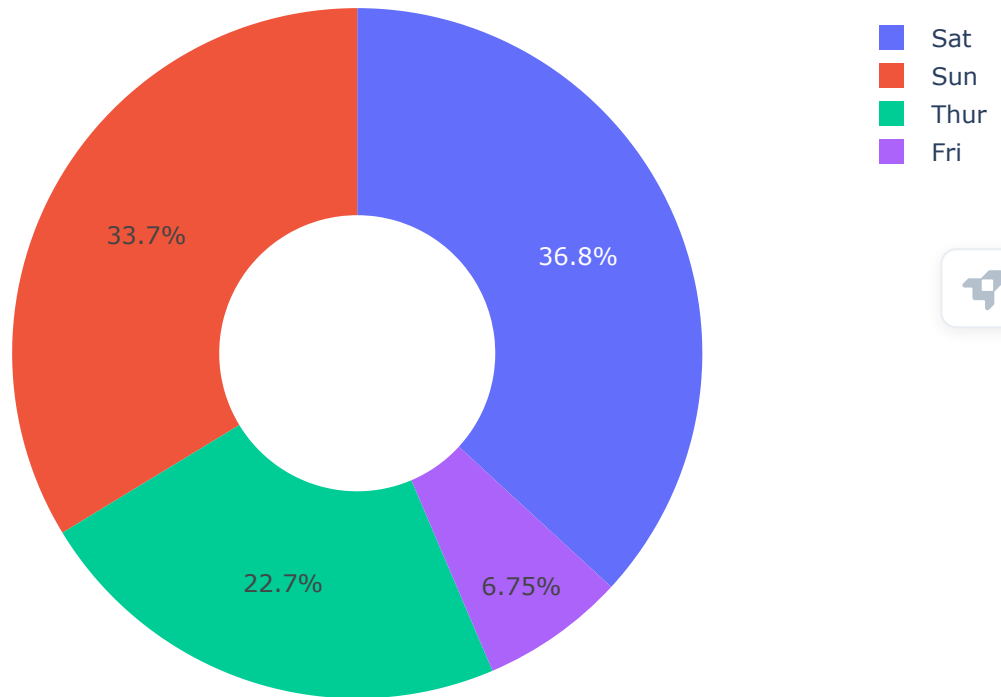
✓ Gráficos de pastel (Pie Charts) y de dona (Donut charts)

- Visualizan la proporción de cada categoría en un conjunto de datos.

```
fig = px.pie(df2,  
             values='total_bill',  
             names='day',  
             title='Distribución de cuentas por día',  
             hole=0.4)  
fig.show()
```



Distribución de cuentas por día



Gráficos geoespaciales (Mapas Interactivos)

- Pltly permite crear mapas interactivos que pueden mostrar datos geográficos.

```
import plotly.express as px

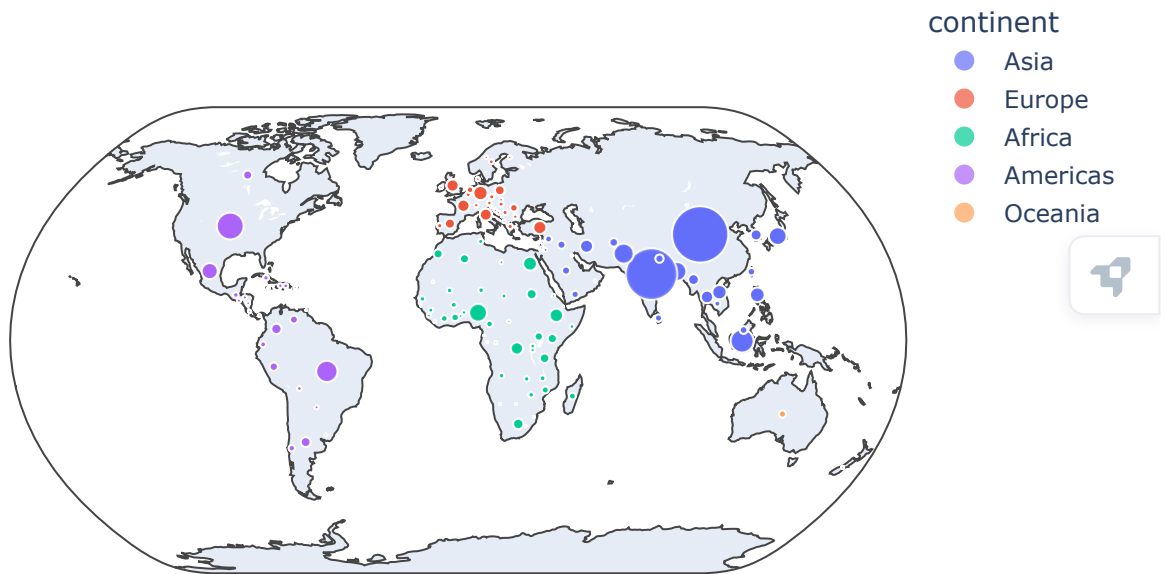
df3 = px.data.gapminder()

fig = px.scatter_geo(df3, locations='iso_alpha',
                    color='continent',
                    hover_name='country',
                    size='pop',
                    projection='natural earth',
                    title='Distribución poblacional mundial')

fig.show()
```



Distribución poblacional mundial



```
import pandas as pd

us_cities = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/us-cities")
us_cities = us_cities.query("State in ['New York', 'Ohio']")

import plotly.express as px

fig = px.line_map(us_cities, lat="lat", lon="lon", color="State", zoom=3, height=300)

fig.update_layout(map_style="open-street-map", map_zoom=4, map_center_lat = 41,
    margin={"r":0,"t":0,"l":0,"b":0})

fig.show()
```



State
— Ohio
— New York



© OpenStreetMap contributors

```
us_cities.head(10)
```



	City	State	Population	lat	lon	
2	Cleveland	Ohio	390113	41.499320	-81.694361	
36	Buffalo	New York	258959	42.886447	-78.878369	
39	Lima	Ohio	38355	40.742551	-84.105226	
44	Mount Vernon	New York	68224	40.912599	-73.837079	
86	Lancaster	Ohio	39325	39.713675	-82.599329	
101	Schenectady	New York	65902	42.814243	-73.939569	
106	Kettering	Ohio	55870	39.689504	-84.168827	
126	Columbus	Ohio	822553	39.961176	-82.998794	
156	Binghamton	New York	46444	42.098687	-75.917974	
164	Cuyahoga Falls	Ohio	49267	41.133945	-81.484558	

Próximos
pasos:

[Generar código con us_cities](#)[Ver gráficos recomendados](#)[New interactive sheet](#)

```
import plotly.graph_objects as go

fig = go.Figure(go.Scattermap(
    fill = "toself",
    lon = [-99.06224], lat = [19.35529],
    marker = { 'size': 10, 'color': "orange" }))

fig.update_layout(
    map = {
        'style': "open-street-map",
        'center': {'lon': -73, 'lat': 46 },
        'zoom': 5},
```

```
showlegend = False)
```

```
fig.show()
```



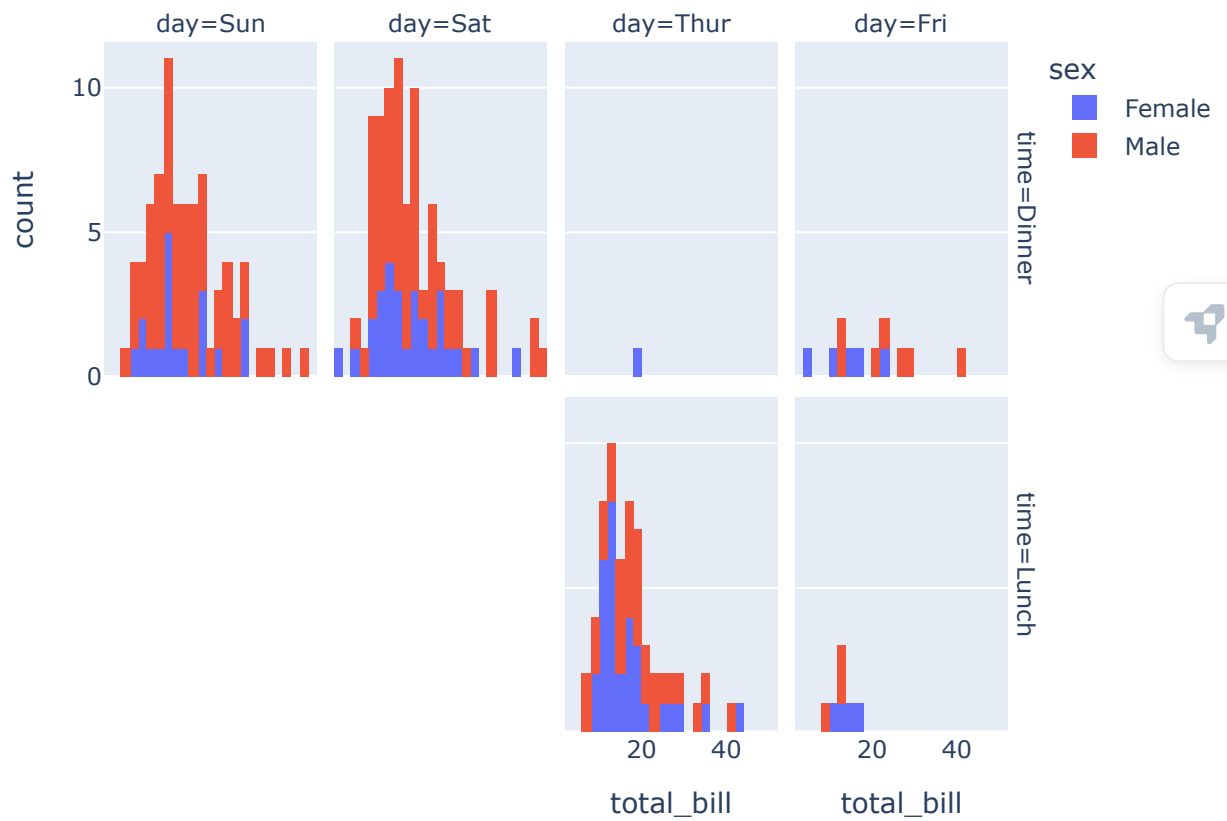
✓ Gráficos de facetas (Facet Grids)

- Permiten dividir los gráficos por categorías para comparar datos en múltiples subconjuntos.

```
fig = px.histogram(df2,  
                   x='total_bill',  
                   color='sex',  
                   facet_row='time',  
                   facet_col='day',  
                   title='Distribución de cuentas por día y sexo')  
fig.show()
```



Distribución de cuentas por día y sexo



✓ Gráficos animados

- Perfectos para mostrar la evolución de datos a lo largo del tiempo.

```
fig = px.scatter(df1, x='gdpPercap', y='lifeExp',  
                 animation_frame='year', animation_group='country',  
                 size='pop', color='continent',  
                 hover_name='country',  
                 log_x=True, size_max=60,  
                 title='Evolución del PIB y esperanza de vida')  
fig.show()
```



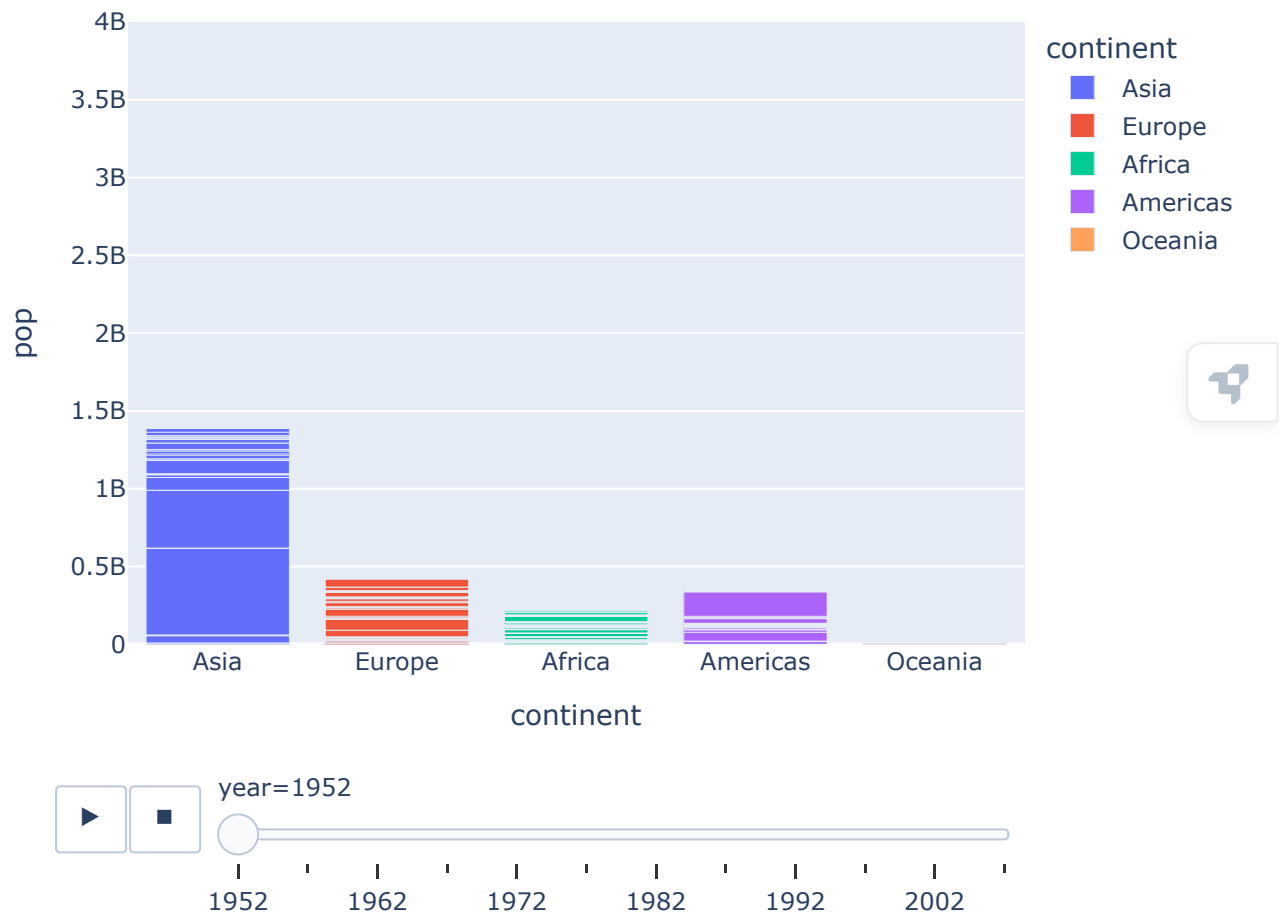

Evolución del PIB y esperanza de vida



```
import plotly.express as px

df = px.data.gapminder()

fig = px.bar(df, x="continent", y="pop", color="continent",
             animation_frame="year", animation_group="country", range_y=[0,4000000000])
fig.show()
```



✓ Combinación de gráficos avanzados

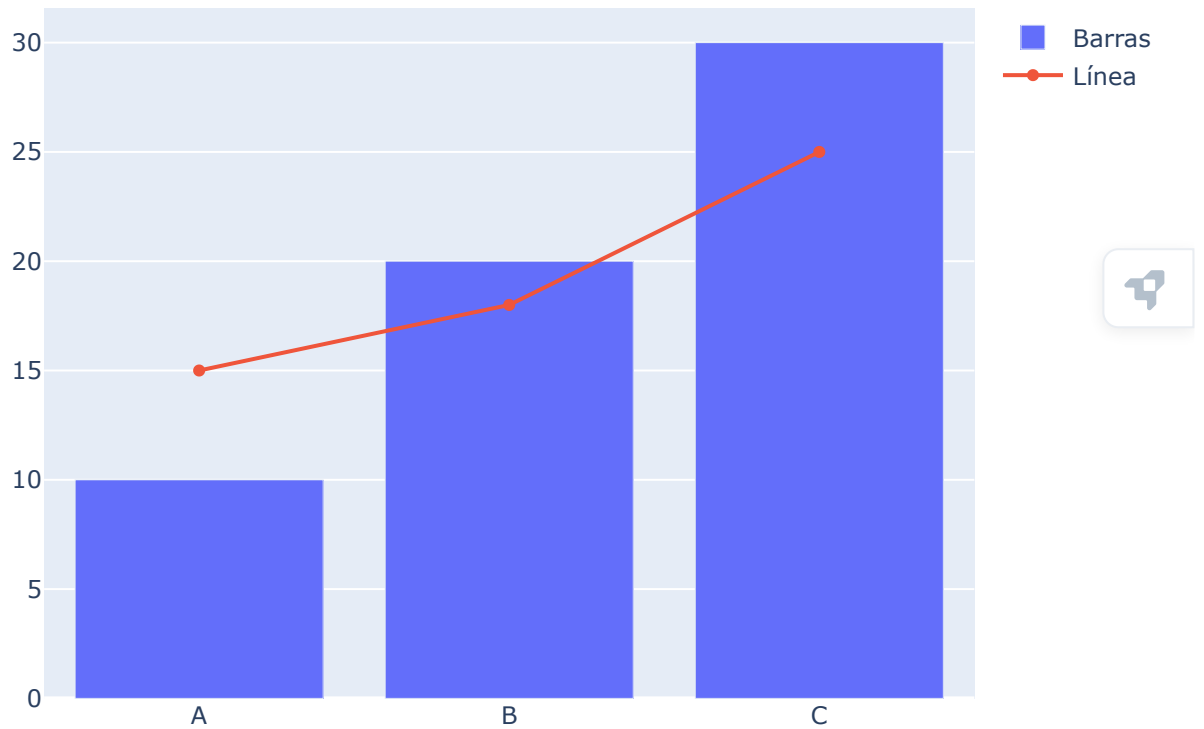
- Plotly permite combinar diferentes tipos de gráficos en una misma visualización

```
import plotly.graph_objects as go

# Gráfico combinado con barras y líneas
fig = go.Figure()

fig.add_trace(go.Bar(x=['A', 'B', 'C'], y=[10, 20, 30], name='Barras'))
fig.add_trace(go.Scatter(x=['A', 'B', 'C'], y=[15, 18, 25],
                        mode='lines+markers', name='Línea'))

fig.show()
```



Gráficos personalizados con Graph Objects.

1. Estructura de `go.Figure()` y `go.Layout()`
2. Manejo de múltiples ejes (secundarios y suspuestos)
3. Combinación de diferentes tipos de gráficos en una sola figura.
4. trazas personalizadas con diferentes estilos y capas.
5. Gráficos de redes interactivas: conexiones entre nodos.
6. Edición manual de ticks, etiquetas y fondos.
7. Uso de imágenes y marcas de agua en gráficos.

✓ 1. Estructura de `go.Figure()` y `go.Layout()`

La clase `go.Figure()` es la base para crear gráficos con Plotly Graph Objects. Su estructura se compone principalmente de dos elementos.

- `data` → Las trazas del gráfico (gráficoso de líneas, barras, etc)
- `layout` → Configuración del diseño (títulos, ejes, colores, etc)

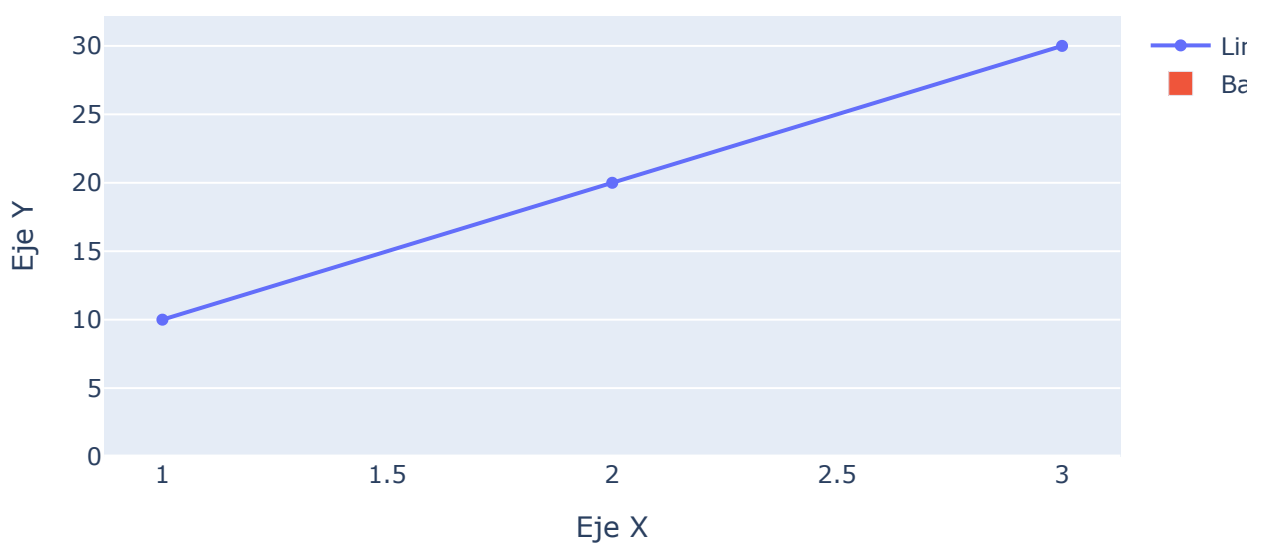
```
import plotly.graph_objects as go
import plotly.io as pio
```

```
pio.renderers.default = 'colab'
```

```
fig = go.Figure(  
    data=[  
        go.Scatter(x=[1,2,3], y=[10,20,30], mode='lines+markers', name='Linea 1'),  
        go.Bar(x=['A', 'B', 'C'], y=[15, 20, 25], name='Barra 1')  
    ],  
    layout=go.Layout(  
        title = 'Gráfico de ejemplo combinado',  
        xaxis =dict(title = 'Eje X'),  
        yaxis =dict(title = 'Eje Y'),  
        width=700,  
        height=400  
    )  
)  
fig.show()
```



Gráfico de ejemplo combinado



✓ 2. Manejo de múltiples ejes (secundarios y superpuestos)

- Permite agregar ejes adicionales para representar diferentes escalas o datos en la misma visualización.
- Se utiliza el parámetro `secondary_y = True` en las trazas.

```
from plotly.subplots import make_subplots  
  
fig = make_subplots(specs=[[{'secondary_y':True}]])  
  
fig.add_trace(go.Bar(  
    x=[1,2,3],  
    y=[15, 20, 25],  
    name='Barra 1'
```

```
x=['Enero', 'Febrero', 'Marzo'],
y=[100, 200, 300], name='Ventas'), secondary_y = False)

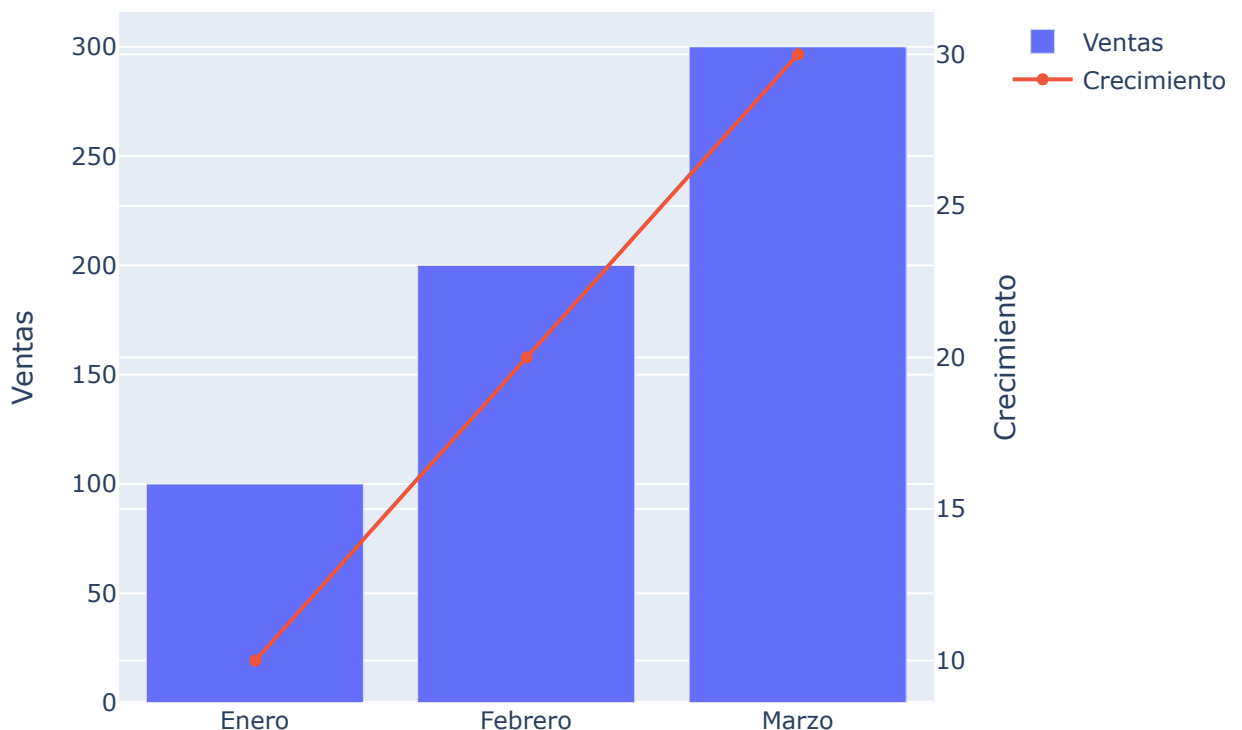
fig.add_trace(go.Scatter(
    x=['Enero', 'Febrero', 'Marzo'],
    y=[10, 20, 30], name='Crecimiento', mode='lines+markers'), secondary_y = True)

fig.update_layout(title='Grafico con Ejes Secunadarios')
fig.update_yaxes(title_text='Ventas', secondary_y=False)
fig.update_yaxes(title_text='Crecimiento', secondary_y=True)

fig.show()
```



Grafico con Ejes Secunadarios



✓ 3. Combiación de diferentes tipos de gráficos.

Plotly permite combinar gráficos de barras, líneas, dispersión en una sola figura.

```
fig = go.Figure()

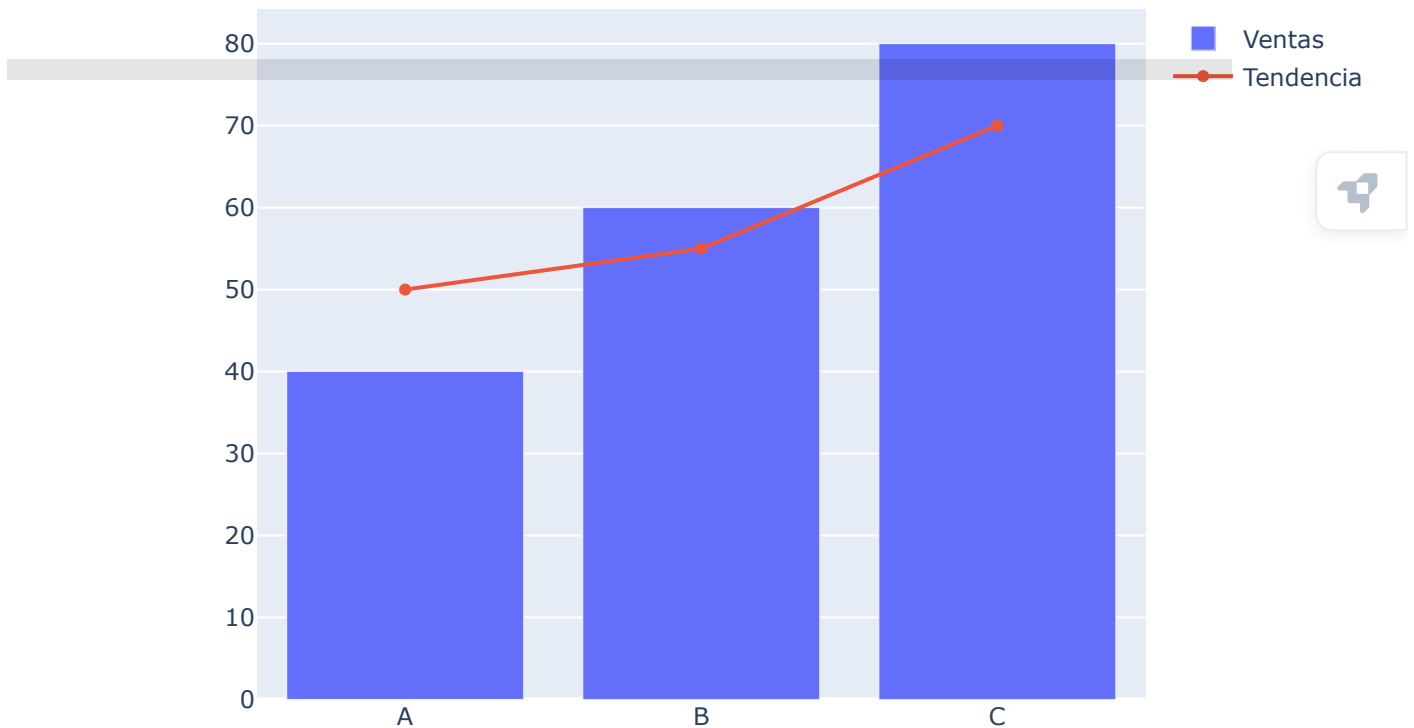
fig.add_trace(go.Bar(x=['A', 'B', 'C'], y=[40, 60, 80], name='Ventas'))
fig.add_trace(go.Scatter(x=['A', 'B', 'C'], y=[50, 55, 70], mode='lines+markers', name='T

fig.update_layout(title='Combinación de gráficos en una sola figura')
```

```
fig.show()
```



Combinación de gráficos en una sola figura



✓ 4. Trazas personalizadas con diferentes estilos y capas.

Podemos personalizar cada traza con colore, formas, tamaños y estilos específicos.

```
fig = go.Figure()

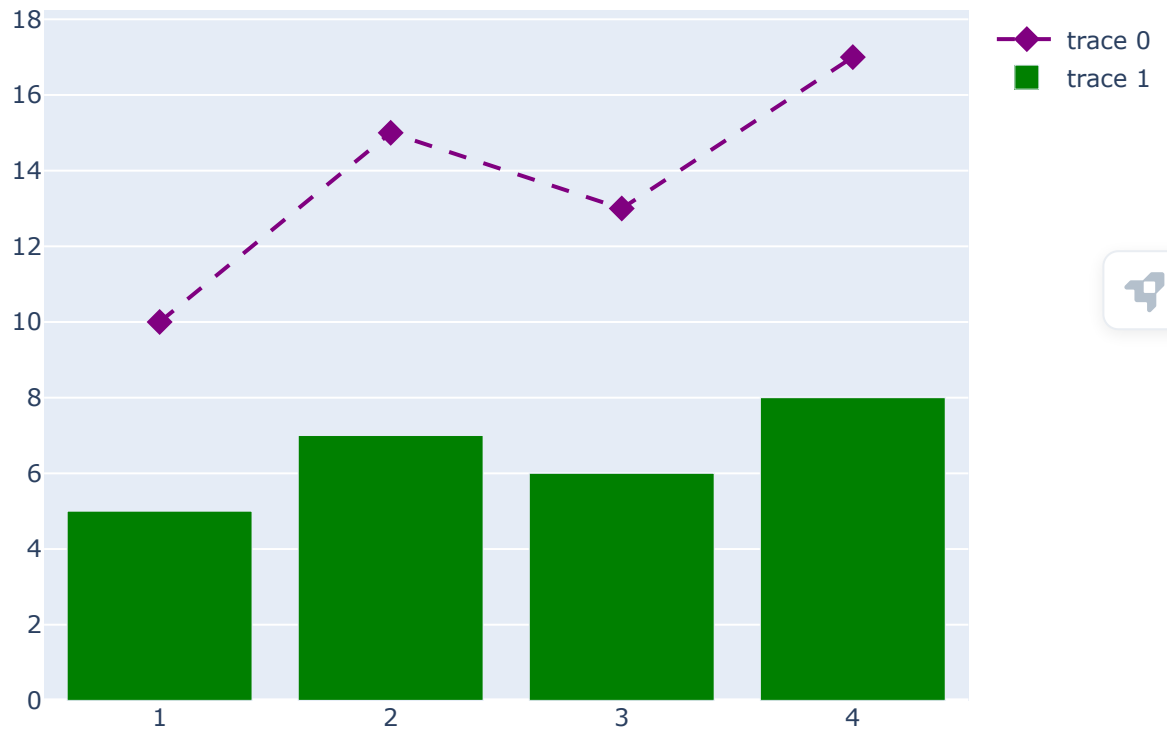
fig.add_trace(go.Scatter(
    x=[1, 2, 3, 4],
    y=[10, 15, 13, 17],
    mode='lines+markers',
    line=dict(color='purple', dash='dash'),
    marker=dict(size=10, symbol='diamond')
))

fig.add_trace(go.Bar(x=[1, 2, 3, 4], y=[5, 7, 6, 8], marker_color='green'))

fig.update_layout(title='Trazas Personalizadas')
fig.show()
```



Trazas Personalizadas



✓ 5. Gráficos de redes interactivas (nodos y conexiones)

Plotly permite crear gráficos interactivos que representas redes complejas.

```
import networkx as nx

# Crear un grafo
G = nx.complete_graph(5)

# Obtener posiciones
pos = nx.spring_layout(G)

# Crear el gráfico
edge_x = []
edge_y = []

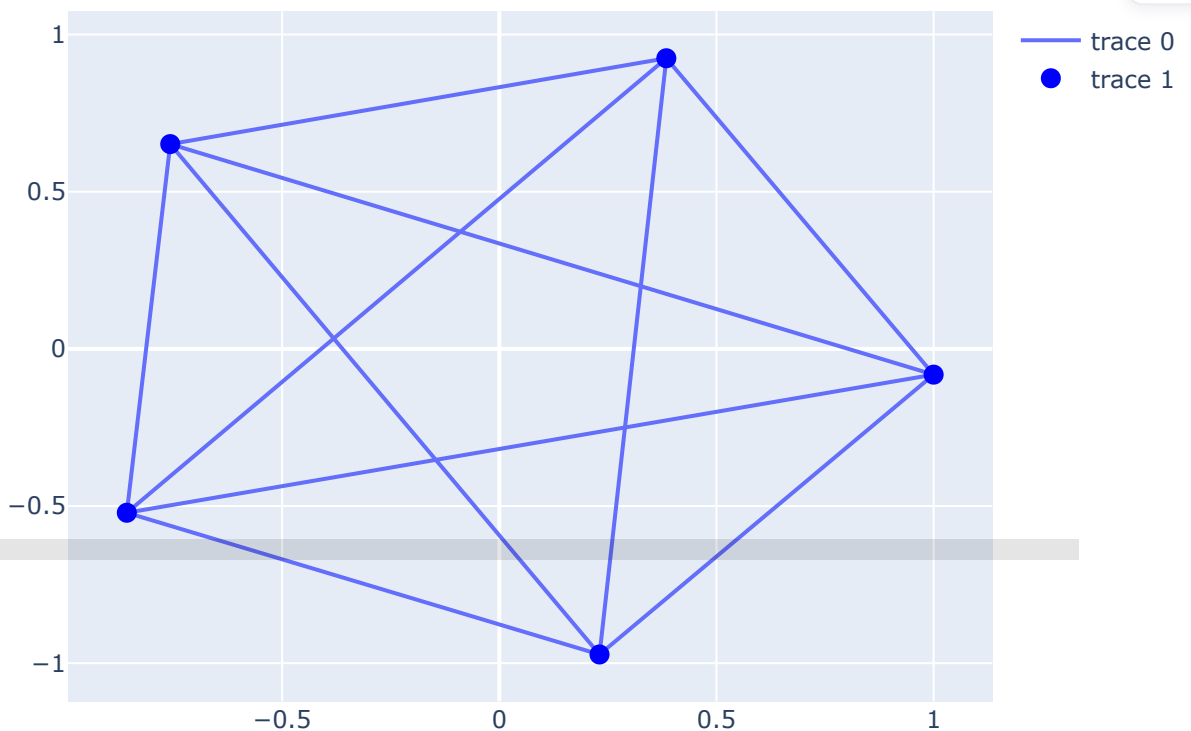
for edge in G.edges():
    x0, y0 = pos[edge[0]]
    x1, y1 = pos[edge[1]]
    edge_x.extend([x0, x1, None])
    edge_y.extend([y0, y1, None])

# Líneas de conexión
fig = go.Figure(go.Scatter(x=edge_x, y=edge_y, mode='lines'))
```

```
# Nodos
node_x, node_y = zip(*pos.values())
fig.add_trace(go.Scatter(x=node_x, y=node_y, mode='markers', marker=dict(size=10, color='
fig.update_layout(title='Red de nodos interactiva')
fig.show()
```



Red de nodos interactiva



✓ 6. Edición manual de ticks, etiquetas y fondos.

Plotly permite personalizar el aspecto de los ejes, ticks y fondos.

```
import plotly.graph_objects as go
import plotly.io as pio
pio.renderers.default = 'colab'

fig = go.Figure()

fig.add_trace(go.Bar(
    x=[1, 2, 3, 4, 5],
    y=[10, 20, 15, 30, 25],
    marker=dict(color=['red', 'blue', 'green', 'purple', 'orange'])
```



```

))

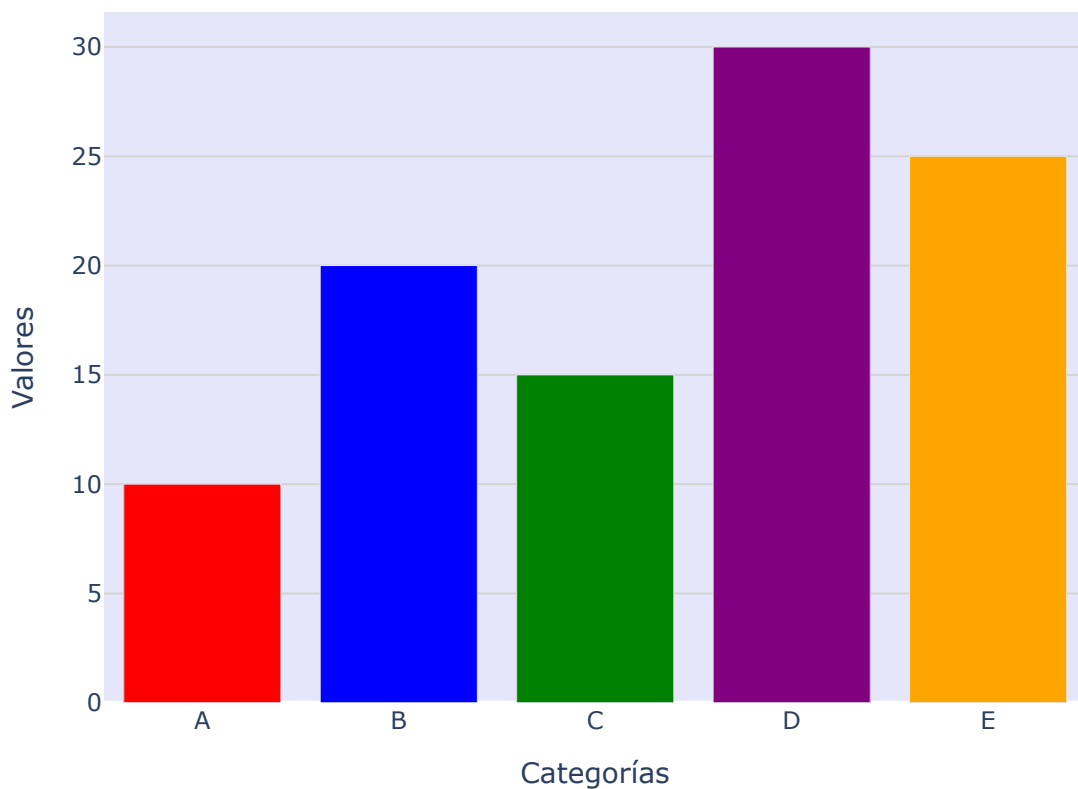
fig.update_layout(
    title="Personalización de ticks y fondo",
    xaxis=dict(
        tickmode='array',
        tickvals=[1, 2, 3, 4, 5],
        ticktext=['A', 'B', 'C', 'D', 'E'],
        title='Categorías'
    ),
    yaxis=dict(
        title='Valores',
        showgrid=True,
        gridcolor='lightgray'
    ),
    plot_bgcolor='rgba(230, 230, 250, 1)',
    paper_bgcolor='rgba(200, 200, 200, 1)'
)

fig.show()

```



Personalización de ticks y fondo



7. Uso de imágenes y marcas de agua.

Podemos agregar imágenes y marcas de agua en los gráficos usando `images` en el `layout`.

```
fig = go.Figure()

fig.add_trace(go.Scatter(
    x=[1, 2, 3, 4, 5],
    y=[5, 10, 8, 15, 12],
    mode='markers',
    marker=dict(size=15, color='red', opacity=0.8),
    name="Puntos"
))

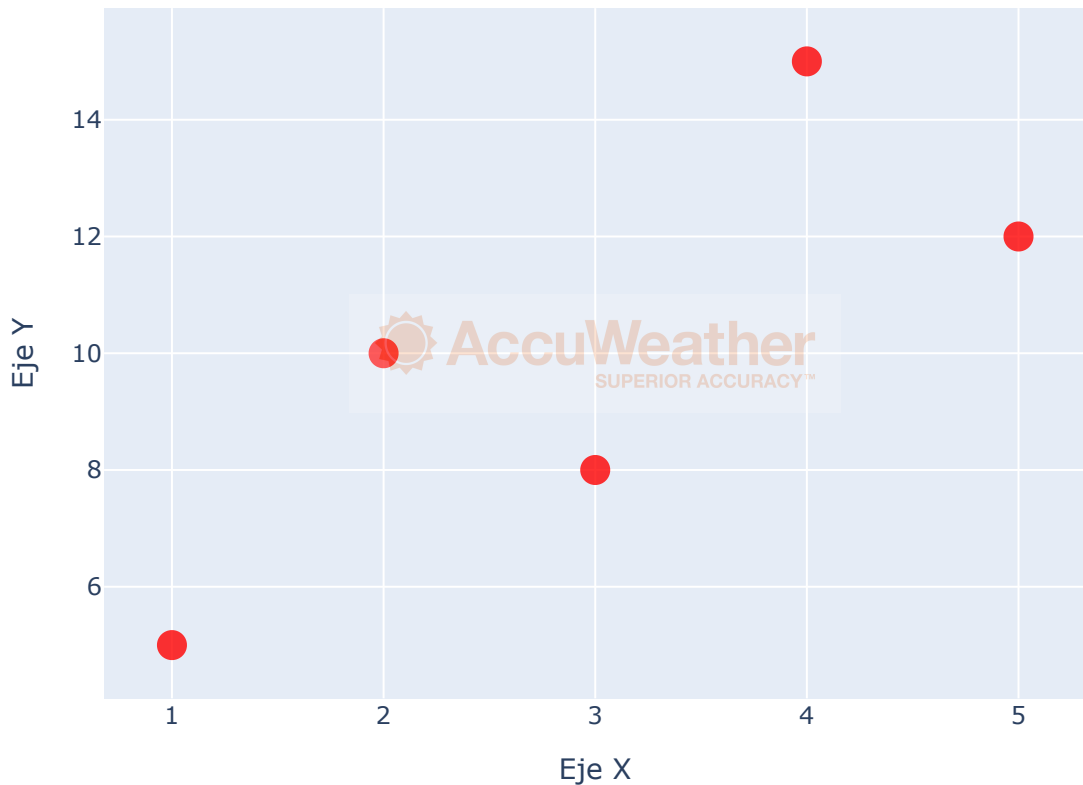
fig.update_layout(
    title="Gráfico con Marca de Agua",
    xaxis=dict(title="Eje X"),
    yaxis=dict(title="Eje Y"),
    images=[
        dict(
            source='https://raw.githubusercontent.com/cldougl/plot_images/add_r_img/accuwatermark.png',
            xref='paper', yref='paper',
            x=0.5, y=0.5,
            sizex=0.5, sizey=0.5,
            xanchor='center', yanchor='middle',
            opacity=0.2
        )
    ]
)

fig.show()
```





Gráfico con Marca de Agua



Interactividad co Plotly

1. Activar y desactivar elementos con hover effects y tooltips personalizados.
2. Zoom y selección de datos dinámicamente.
3. Uso de sliders y botones para cambiar visualizaciones en tiempo real.
4. Filtrado y actualización de gráficos sin necesidad de recargar la página.
5. Eventos personalizados: cómo capturar clics y selecciones en gráficos.
6. Uso de dropdowns para cambiar entre diferentes visualizaciones.
7. Exportación y descarga de gráficos interactivos

✓ Activar y desactivar elementos con hover y tooltips personalizados.

- Gráfico de dispersion con hover y tooltips personalizados.
- Se muestra información personalizada al pasar el mouse por cada punto. con hoverinfo
- hovermode='x=unified' agrupa los tooltips cuando los puntos comparten la misma coordenada.

Comienza a programar o [generar](#) con IA.

```
import plotly.graph_objects as go
import plotly.io as pio
pio.renderers.default = 'colab'

x = [1, 2, 3, 4, 5]
y = [5, 10, 8, 15, 12]
nombres = ['A', 'B', 'C', 'D', 'E']

figura = go.Figure()

figura.add_trace(go.Scatter(
    x=x,
    y=y,
    mode='markers',
    marker=dict(size=15, color='blue', opacity=0.8),
    text=[f'Punto{n}: {v}' for n, v in zip(nombres, y)],
    hoverinfo='text'
))

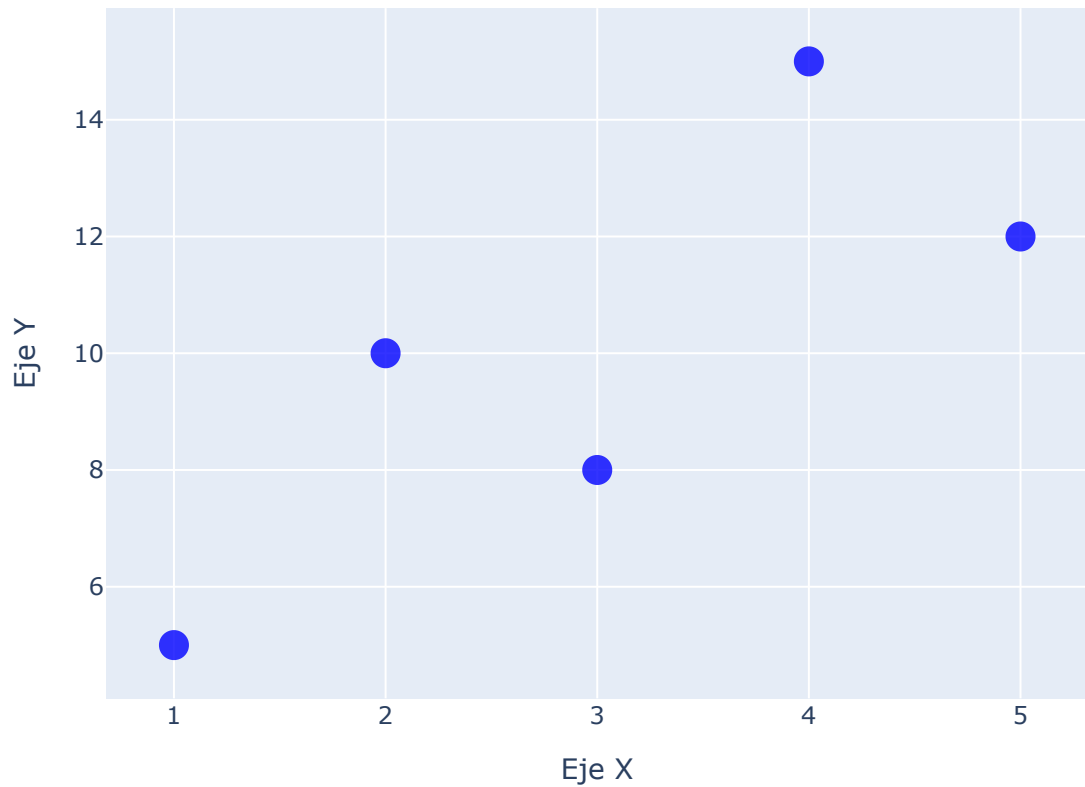
figura.update_layout(
    title="Gráfico con Hover y Tooltips Personalizados",
    xaxis_title="Eje X",
    yaxis_title="Eje Y",
    hovermode='x unified'
)

figura.show()
```





Gráfico con Hover y Tooltips Personalizados



✓ Zoom y selección de datos dinámicamente.

- Se agrega un range slider en el eje x para facilitar el zoom.
- Se activa con la función de zoom interactivo con `dragmode='zoom'`

```
import numpy as np

x = np.linspace(0, 10, 100)
y = np.sin(x)

figura1 = go.Figure()

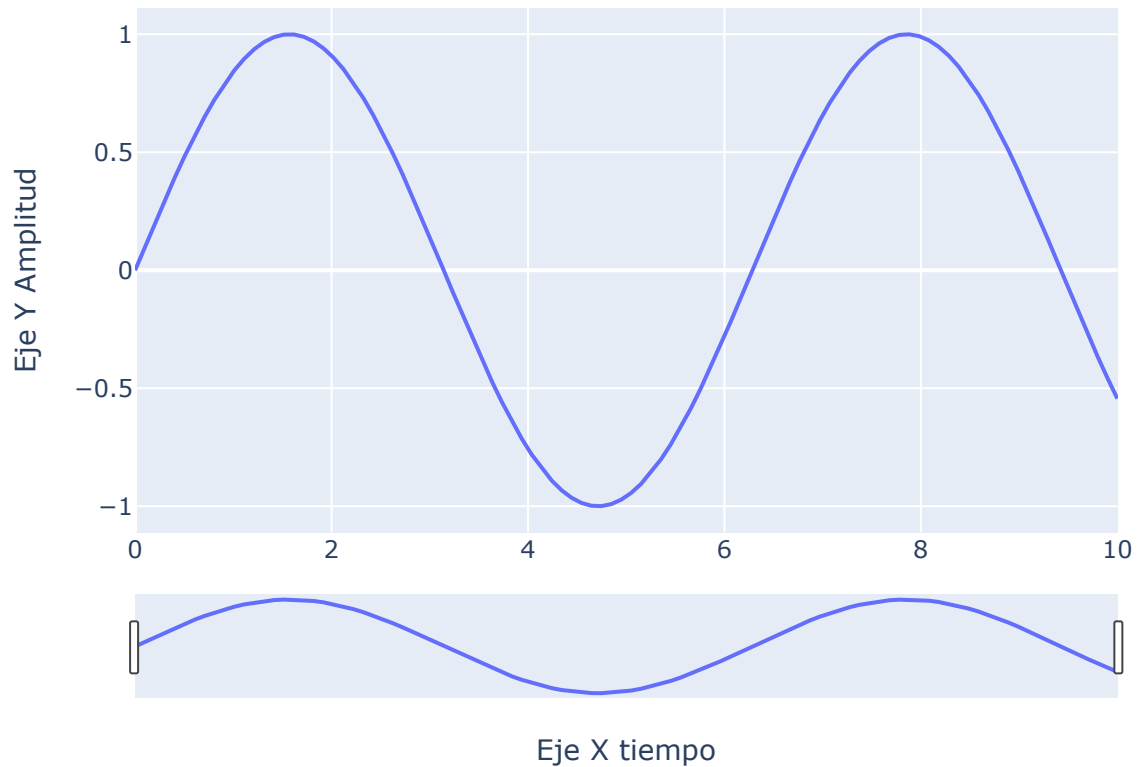
figura1.add_trace(go.Scatter(x=x, y=y, mode='lines', name='Sin(x)'))

figura1.update_layout(
    title="Gráfico con Zoom y Selección de Datos",
    xaxis=dict(title="Eje X tiempo", rangeslider=dict(visible=True)),
    yaxis=dict(title="Eje Y Amplitud"),
    dragmode='zoom'
)

figura1.show()
```



Gráfico con Zoom y Selección de Datos



✓ Uso de sliders y botones para cambiar visualizaciones en tiempo real.

- Slider para cambiar la frecuencia una onda senoidal.

```
import plotly.express as px
from ipywidgets import interact

def update_plot(frecuencia=1):
    x = np.linspace(0, 10, 100)
    y = np.sin(frecuencia * x)

    figsen = go.Figure()
    figsen.add_trace(go.Scatter(x=x, y=y, mode='lines', name='Sin(x)'))

    figsen.update_layout(
        title='Onda Senoidal'
    )
    figsen.show()

interact(update_plot, frecuencia=(1,10,1))
```



frecuencia



1



```
update_plot
def update_plot(frecuencia=1)
```

```
<no docstring>
```

✓ Filtrado y actualización de gráficos sin recargar la página.

- se usa `updatemenus` para crear un dropdown interactivo.
- Se pueden ocultar o mostrar ciertas categorías sin necesidad de recargar la página.

```
import pandas as pd

df = pd.DataFrame({
    "Categoría": ["A", "A", "B", "B", "C", "C"],
    "Valor": [10, 15, 8, 12, 20, 25]
})

fig = go.Figure()

for cat in df["Categoría"].unique():
```

```
subset = df[df["Categoría"] == cat]
fig.add_trace(go.Bar(x=[cat]*len(subset), y=subset["Valor"], name=f"Categoría {cat}"))

fig.update_layout(
    title="Filtrado con Dropdown",
    updatemenus=[{
        "buttons": [
            {"label": "Todos", "method": "update", "args": [{"visible": [True, True, True]}]},
            {"label": "Categoría A", "method": "update", "args": [{"visible": [True, False, False]}]},
            {"label": "Categoría B", "method": "update", "args": [{"visible": [False, True, False]}]},
            {"label": "Categoría C", "method": "update", "args": [{"visible": [False, False, True]}]}
        ]
    }]
```

