**Clase 13 – Visualización II**

**Librería Bokeh:** En japonés, Bokeh es desenfoque. Mientras que Matplotlib genera gráficos estáticos útiles para una visualización rápida y simple, Bokeh crea visualizaciones para **mostrar en la web**, **altamente interactivas**.

Da una construcción concisa de gráficos en forma versátil, ofreciendo interactividad de alto rendimiento sobre grandes volúmenes de datos o streaming.

Con **bokeh.plotting** se obtienen los elementos para crear los gráficos interactivos. Se recomiendan los siguientes pasos:

1. Preparar el **dataset**. Lista, con tipos de datos de Numpy y Pandas.
2. Informar **dónde se genera la salida del gráfico**. Generalmente se pueden usar la función **output\_file**, que genera archivos html; o bien **output\_notebook** que las muestra como celdas de Jupyter Notebook. A menos que se quiera modificar la salida luego, esto se define sólo una vez.
3. Usar la función **figure()**, análogamente a matplotlib, es el contenedor donde se define el estilo general, títulos, grillas (grids), labels de ejes y también los **tools** (**barra con botones** para actuar de **forma interactiva**).
4. Agregar los **Glyphs**. Son los puntos, líneas, áreas y otras figuras geométricas que representan los datos.
5. Informar cómo mostramos los resultados. **show()** muestra el gráfico en el browser; **save()** lo guarda en el archivo que definamos.

IE:

**from** bokeh.plotting **import** figure, output\_notebook, show

**from** bokeh.io **import** output\_notebook

**from** bokeh.resources **import** INLINE

#dataset

circle\_x = [1, 2.5, 3, 2]; circle\_y = [2, 3, 1, 1.5]

triangle\_x = [1, 3, 2]; triangle\_y = [3, 1, 1.5]

#salida del gráfico

output\_notebook(INLINE)

#figure

P = figure(plot\_width=250, plot\_height=150, tools=”pan, reset, save”)

#glyphs

p.triangle(x=triangle\_x, y=triangle\_y, color=“red”, size=20)

p.circle(xcircle\_x, y=circle\_y, radius=0.3, alpha=0.5)

Se pueden obtener Datasets a partir de Bokeh:

**Import** bokeh.sampledata

bokeh.sampledata.download()

IE:

**from** bokeh.sampledata.gapminder **import** fertility, life\_expectancy, population, regions

anios = [“1970”, “1980”, “1990”, “2000”, “2010”]

s\_argentina = life.expectancy.loc[“Argentina”,anios]

Es posible agregar en **figure** el título, etiquetas de ejes y tamaño del gráfico. Con el parámetro **tools** se describen las acciones interactivas permitidas, las cuales pueden gruparse por:

* **drag tools** (arrastrando el mouse).  **box\_select** (rectángulo donde hacer zoom); **lasso\_select** (lazo); **pan** (genera una panorámica, moviendo el mouse mientras hacemos click).
* **click tools** (con un click). t**ap** (selecciona un punto)
* **scroll tools**. **wheel\_zoom** (zoom con la rueda del mouse); **box\_zoom**.
* **Acciones**. **reset** (Vuelve a los valores originales); **sabe** (guarda el archivo como un PNG).

IE:

anios = [“1970”, “1980”, “1990”, “2000”, “2010”]

output\_notebook(INLINE)

#Figure

fig = figure(title=”Expectativa de vida”, x\_axis\_label=”Años”, y\_axis\_level=”Años de vida”, tools=”box\_select,box\_zoom,lasso\_select,tap,wheel\_zoom,pan,reset,save”, toolbar\_location=”above”, plot\_width=400, plot\_height=250)

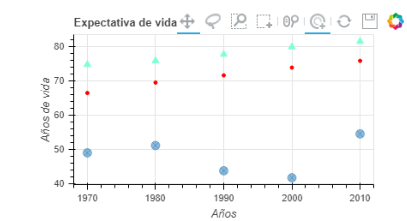
#tres Glyphs

fig.circle(x=anios, y=s\_argentina, color=”red”, size=4)

fig.triangle(x=anios, y=s\_suecia, color=”aquamarine”, size=8)

fig.circle\_x(x=anios, y=s\_zambia, size=10, alpha=0.5)

show(fig)



**PALETTE:** Bokeh ofrece una paleta de colores predefinidas que se puede ver en:

<https://docs.bokeh.org/en/latest/docs/reference/palettes.html>

Con colores de **Set2** se pueden colorear las barras de este gráfico. Hay que informar cuántos colores vamos a usar, en este caso 5.

**from** bokeh.palettes **import** Set2

colors = Set2[5]

**BARRAS:** Es posible graficar barras verticales, horizontales o apiladas. Para barras verticales, **vbar()** definimos el rango de valores en el eje x, **x=**; y el rango de valores en el eje y **top=**. También se puede agregar un valor **bottom=.** El proceso es parecido para barras horizontales. Es necesario definir un **offset** entre varias barras para que no se superpongan entre sí. El método **p.add\_layout** sirve para dejar las leyendas afuera del gráfico. Se agrega en **figure** el parámetro **x\_range =** anios para que el eje x permita valores categóricos.

IE:

**import** numpy **as** np

**from** bokeh.models **import** Legend

p = figure(x\_range=anios, plot\_width=600, plot\_height=230, title=”Comparación de países”, x\_axis\_label=”Años”, y\_axis\_label=”Años de vida”)

r0 = p.vbar(x=anios, tops=s\_zambia, width=0.1, bottom = 0, color=colors[0])

offsets = [0.1 , 0.1, 0.1, 0.1, 0.1]

anios\_off=list(zip(anios, offsets))

r1 = p.vbar(x=anios\_off, tops=s\_suecia, width=0.1, bottom = 0, color=colors[1])

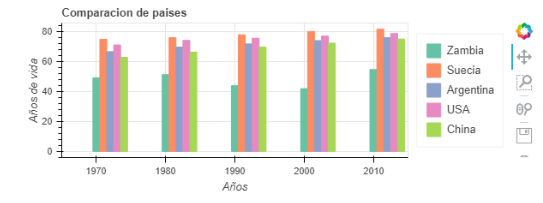
anios\_off=list(zip(anios, offsets)\*2)

Misma lógica para r3 y r4.

legend = Legend(ítems=[(“Zambia”, [r0]), (“Suecia”, [r1]), (etc.)], location =”center”)

p.add.layout(legend, “right”)

show(p)



**AREAS**: Se grafican pintando la región entre dos series que compartan un eje x o un índice en común.

IE:

output\_notebook(INLINE)

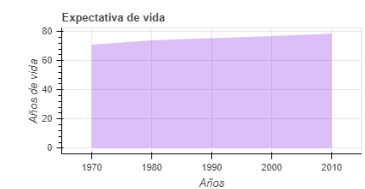
#Figure

fig = figure(title = “Expectativa de vida”, x\_axis\_label = “Años”, y\_axis\_label = “Años de vida”, tools = “pan, reset, save, Wheel\_zoom”, toolbar\_location = “below”, x\_range = anios, plot\_width = 400, plot\_height = 250)

#Area

fig.varea(x = anios, y1 = 0, y2 = s\_usa, fill\_color = “blueviolet”, alpha = 0.3);

show(fig)



**RECTÁNGULOS, ELIPSES, POLÍGONOS:** **rect()**, **square()** generan rectángulos y cuadrados basado en las coordenadas del centro de la figura. En **rect()** las dimensiones se definen con ancho y largo. En **square()** las dimensiones se definen con **size**.

Con **ellipse()** pueden crearse glyphs con las coordenadas **x**, **y** y el **ancho** y **largo**.

IE:

#salida del gráfico

output\_notebook(INLINE)

#figure

fig = figure(plot\_width=300, plot\_height=150)

#glyphs

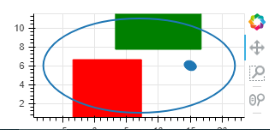
fig.rect(x=10, y=10, width=100, height=50, color=”green”, width\_units=”screen”, height\_units=”screen”)

fig.square(x=2, y=3, size=80, color=”red”)

fig.ellipse(x=7,y=6,width=30, height=10, fill\_color=None, line\_width=2)

fig.ellipse(x=15, y=6, width=2, height=1, angle=-0.4)

show(fig)



**COLUMNDATASOURCE – GRIDPLOT:** **ColumnDataSource** se usa mucho en gráficos Bokeh; permite compartir fácilmente datos entre múltiples gráficos.

Si se usa el mismo objeto **ColumnDataSource**, la selección del origen de datos también es compartida.

Con **gridplot** se crea una grilla de plots; con una única **toolbar** compartida entre todos los plots de dicha grilla. Cada gráfico deberá tener su figure y su glyph; y luego se unirán con **gridplot**.

IE:

**from** bokeh.models **import** ColumnDataSource

**from** bokeh.layouts **import** gridplot

#data de autos

**from** bokeh.sampledata.autompg **import** autompg **as** data\_cars

data\_bokeh = ColumnDataSource(data\_cars)

options = {“plot\_width”: 300, “plot\_height”:150, tools: “pan, box\_select, lasso\_select, reset”}

p1 = figure(title=”MPG por Año”, x\_axis\_label = “MPG”, y\_axis\_label = “Año”, \*\*options)

p1.circle(“yr”, “mpg”, color = “blue”, source = data\_bokeh)

p2 = figure(title=”HP vs Desplazamiento”, x\_axis\_label = “HP”, y\_axis\_label = “Desp”, \*\*options)

p2.cross(“hp”, “displ”, color = “green”, source = data\_bokeh)

p3 = figure(title=”MPG vs Desplazamiento”, x\_axis\_label = “MPG”, y\_axis\_label = “Desp”, \*\*options)

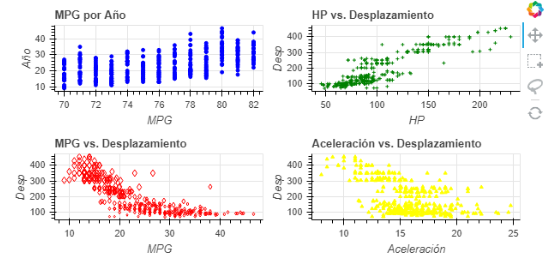
p3.diamond(“mpg”, “displ”, size = “cyl”, line\_color = “red”, fill\_color = None, source = data\_bokeh)

p4 = figure(title=”Aceleración vs Desplazamiento”, x\_axis\_label = “Aceleración”, y\_axis\_label = “Desp”, \*\*options)

p4.triangle(“accel”, “displ”, color = “yellow”, source = data\_bokeh)

p = gridplot([[p1, p2], [p3, p4]], toolbar\_location=”right”)

show(p)



**HOVERTOOL:** Herramienta de isnpección pasiva; permite asociar a cada dato que se visualiza en el gráfico una tabla que lo complementa.

IE:

Hover = HoverTool(tooltips=[(“index”,”$index”), (“(x,y)”, “($x,$y)”), (“desc”, “@desc”)])

Las etiquetas y valores se asignan en una lista de tuplas (*etiqueta, valor*).

Los nombres de **campos que comienzan con $** son **campos especiales**. A menudo se trata de valores intrínsecos del gráfico, como las coordenadas del mouse en la pantalla:

1. $index: Índice del punto seleccionado del dataset.
2. $x: Coordenada x del cursor en el espacio de datos.
3. $y: Coordenada y del cursor en el espacio de datos.

Los nombres de **campos que comienzan con @** están asociados a columnas definidas en **ColumnDataSource**.

IE:

**from** bokeh.plotting **import** ColumnDataSource, figure, output\_file, show

**from** bokeh.models **import** HoverTool

output\_notebook(INLINE)

#Dataset

source = ColumnDataSource(data=dict(x=[1, 2, 3, 4, 5], y = [2, 5, 8, 2, 7], desc=[“A”, “b”, “C”, “d”, “E”],))

#Figure

P = figure(plot\_width=400, plot\_height=200, tools = [“pan”, “box\_zoom”, “reset”], title = “Mouse over the dots”)

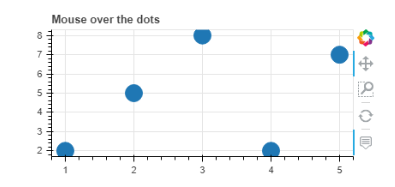
p.circle(“x”, ”y”, size=20, source = source, hover\_color = “pink”, hover\_alpha = 0.6)

# Tooltips indica qué valores muestra al pasar el mouse por los puntos

hover = HoverTool(tooltips=[(“index”,”$index”), (“(x, y)”, “($x, $y)”), (“desc”, “@desc”)])

p.add\_tools(hover)

show(p)



**ARCHIVOS HTML:** Bokeh permite guardar plots en archivos HTML. Para esto, se usa la función **output\_file** y **save** en vez de **output\_notebook()** y **show**. También es posible cominar **output\_file** y **show** para abrir una nueva pestaña con el gráfico guardado.

IE:

**From** bokeh.plotting **import** output\_file, save

output\_file(“test\_save\_plot.html”, mode=”inline”)

save(p);

Si queremos resetear los settings de output, podemos usar **reset\_output**

IE:

**from** bokeh **import** reset\_output

reset\_output()

**PLOTLY:** **Librería de Python** que genera **gráficos interactivos** de una manera sencilla: diagramas de líneas, barras, histogramas, etc. Y diagramas de burbujas, usando grilla de gráficos, hovertool y animaciones. Se pueden mostrar en Jupyter Notebooks, o volcarlos en archivos html e incluso en aplicaciones web usando la librería ***dash***.

**PLOTLY EXPRESS**: Función de Plotly con una interfaz de alto nivel. Opera sobre una variedad de tipos de datos y genera figuras fáciles de trabajar en su estilo. Las funciones de Plotly Express dan como resultado un objeto de **tipo graph\_objects.Figure**. Los datos y Layout dependerán de lo que le demos como argumentos. Se puede mostrar en la Notebook con el método **show**. Se puede guardar en un archivo con **write\_image**.

Esquema general para crear gráficos:

fig = px.chart\_type(df, parameters) (*chart\_type:* bar, scatter, etc. *df:* Dataframe)

fig.update\_layout(layout\_parameters or add annotations)

fig.update\_traces(further graph parameters)

fig.update\_xaxis() # or update\_yaxis

fig.show()

IE:

**import** numpy **as** np

**import** pandas **as** pd

**import** plotly.express **as** px

**import** plotly **as** pl

pl.offline.init\_notebook\_mode(connected=**True**)

fig = px.bar(x=[“DS”, “DE”, “SE”], y=[20, 14, 23], width=400, height=250)

fig.update\_layout(title=”sample\_figure”)

fig.update\_traces(marker\_color”red”, width=0.1)

fig.update\_xaxes(title=”Profesiones”)

fig.update\_yaxes(title=”Total Salario”, titlefont\_size=8, tickfont\_size=14)

fig.show()

fig.write\_image(“fig1\_offline.png”)



**Diagrama de Barras**

Con **px.bar** podemos representar cada fila de una DataFrame con una **barra**. Podemos graficar **diagramas de barras (bar)**, **barras apiladas (stacked bar)** y agrupadas por **variables categóricas**. Se definen los datos y ajustes generales con **px.bar**; podemos definir una tercer variable con **color**. Y además podemos agregar información de estilo con **update\_layout**. Pasando el mouse por los datos podemos ver sus valores.

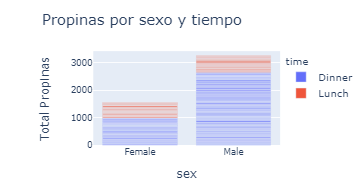
IE:

df\_tips = px.data.tips #dataset de propinas

fig = px.bar(data\_frame = df\_tips, x = “sex”, y = “total\_bill”, color = “time”, width = 400, height = 250)

fig.update\_layout(title=”propinas por sexo y tiempo”, xaxis\_tickfont\_size=10, yaxis\_dict(title=”Total Propinas”, titlefont\_size=14, tckfont\_size=10))

fig.show()



**Diagrama de Burbujas (Bubble Chart):** Diagrama Scatter donde el tamaño de cada punto representa el valor de una columna del DataFrame. Se representa con el parámetro **size**. La burbuja o tamaño de cada punto representa el valor de la variable **pop**.

IE: Dado el DataFrame data\_gap; se va a representar un gráfico donde veremos un Scatterplot entre la expectativa de vida (lifeExp) vs PBI per cápita (gdpPercap), para 2007, para todos los países. Las burbucas crecen en función del tamaño de cada población. Cada continente se representa cada continente con un color distinto. Al pasar el mouse por cada burbuja, podremos ver un texto (hover) con el nombre del país (country). Crearemos una barra para seleccionar el rango del eje x que queremos ver **fig.update\_xaxes(rangeslider\_visible = True)**

IE:

data\_year\_mask = data\_gap[“year”] == 2007

data\_year = data\_gap.loc[data\_year\_mask, :]

fig = px.scatter(data\_frame = data\_year, x = “gdpPercap”, y = “lifeExp”, size = “pop”, color = “continent”, opacity = 0.6, hover\_name=”country”, width=700, height=500)

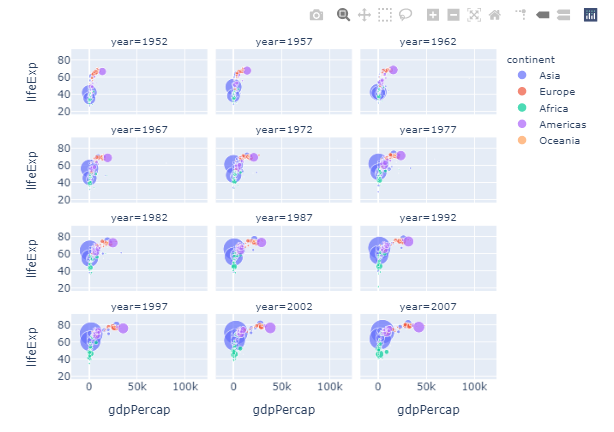
fig.update\_xaxes(rangeslider\_visible = True)

fig.show()

**FACET:** Es la función análoga a gridplot en Bokeh: Sirve para generar múltiples gráficos vinculados. Dichos gráficos vinculados se crean con el argumento **facet\_col** (o bien **facet\_row**)**.** Las distintas columnas de la grilla de gráficos corresponderán a los diferentes valores de la columna indicada por dicho argumento. Se puede especificar el número máximo de columnas que tendrá el gráfico con **facet\_col\_wrap**.

En el siguiente ejemplo, con la columna year se indica que cada columna de la grilla corresponderá a un año.

fig = px.scatter(data\_gap, x= “gdpPercap”, y = lifeExp”, color = “continent”, size = “pop”, facet\_col = “year”, facet\_col\_wrap= 3)



**ANIMACIONES**: Se trata de gráficos que muestran un movimiento según los valores de alguna columna. Con el argumento **animation\_frame** se indica cuál es la columna sobre la cual el gráfico deberá evolucionar, es decir, se **define cada cuadro (*frame*) de la animación**. En general se usa con **variables de *tiempo****.* Con el argumento **animation\_group** se puede indicar la **columna que da la consistencia entre los frames**. Es decir, aquellos objetos que coincidan en el animation\_group representan el mismo objeto en todos los frames. Country sería el animation\_group en el caso que estamos viendo. Con **range\_x** y **range\_y** puede definirse el **rango de valores de los ejes** x e y respectivamente. Se puede definir una **escala logarítmica** a los ejes dándole True a los argumentos **log\_x** o **log\_y**. Queremos que los rangos abarquen todos los datos para poder ver toda la evolución; si algo queda afuera nos lo vamos a perder.

IE:

fig = px.scatter(data\_gap, x =”gdpPercap”, y = “lifeExp”, animation\_frame = “year”, animation\_group = “country”, size = “pop”, color = “continent”, hover\_name = “country”, log\_x = True, size\_max = 55, range\_x =[100, 100000], range\_y = [25,90])

fig.show()

