



VISUALIZACIÓN DE DATOS Y STORYTELLING

Visualización de Datos con Python



Si a los datos no los estamos visualizando para comprender mejor el mundo en su interior, nos estamos perdiendo de muchas cosas. Es decir, los datos pueden tener algún sentido como números, pero la magia ocurre cuando intentas visualizarlos. Tienen más sentido y de repente se vuelven más perceptibles.

Existen muchas herramientas para visualizar los datos, pero el lenguaje Python se ha vuelto cada vez más popular; por lo que a continuación analizaremos

dos de las bibliotecas más populares para la visualización de datos en este lenguaje.

1. Matplotlib y Seaborn

Matplotlib

Matplotlib se creó teniendo en cuenta el estilo de graficación de MATLAB, aunque también tiene una interfaz orientada a objetos, la cual puede usarse de la siguiente forma:

```
import matplotlib.pyplot as plt
figure, axes = plt.subplots(2) # para dos subgraficos

# Ahora se puede configurar un gráfico usando
# funciones disponibles para estos objetos.
```

Esta es una **biblioteca de bajo nivel** en donde es posible tener el control total del gráfico.

Seaborn

Seaborn es una **biblioteca de nivel superior** para visualización, desarrollada sobre matplotlib. Se utiliza principalmente para hacer trazados rápidos y atractivos sin mucha complejidad. Aunque seaborn trata de dar algo de control sobre los gráficos de una manera elegante, aún así no es posible obtener todo el control. Para eso, se debe usar la **funcionalidad de matplotlib**.



2. Gráficas de distribución

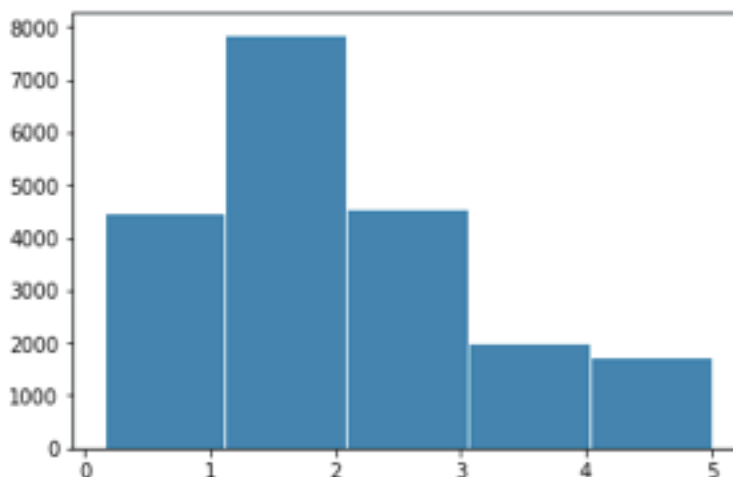
Las **gráficas de distribución** (o gráficas de probabilidad) nos dicen cómo se distribuye una variable. Nos da la **probabilidad de encontrar una variable en un rango particular**. Es decir. Si tuviéramos que seleccionar aleatoriamente un número del rango total de una variable, nos da probabilidades de que esta variable esté en **diferentes rangos**.

Las gráficas de distribución deben estar normalmente distribuidas, para obtener mejores resultados. Este es uno de los supuestos de todos los modelos lineales, es decir, Normalidad.

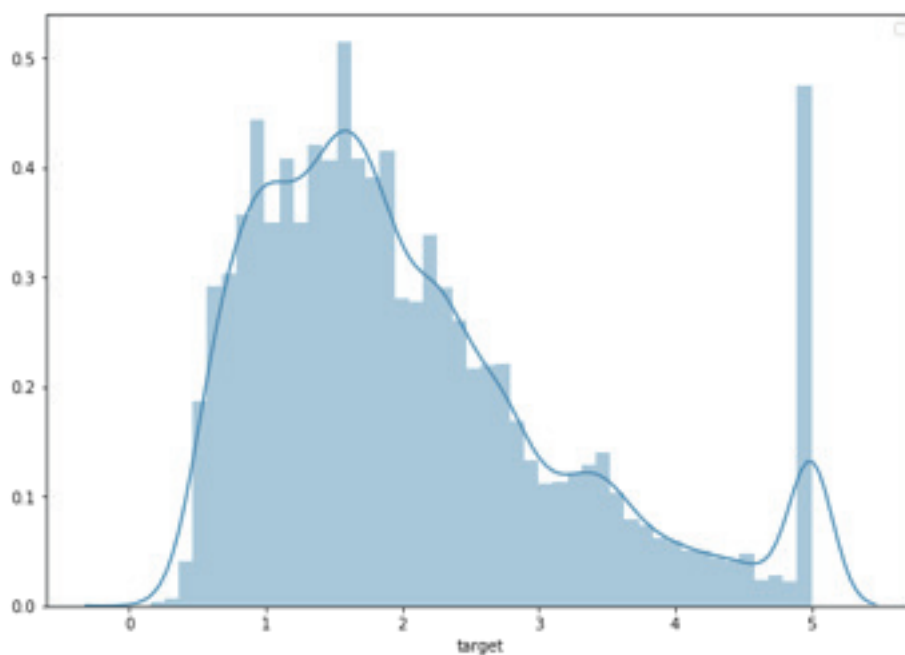
La distribución normal parece una joroba mediana en el centro con colas ligeras.

1. Histogramas y gráficos de estimación de densidad del núcleo (KDEs)

```
# Simple Histograma  
_ = plt.hist(train_df['target'], bins=5, edgecolors='white')
```



```
# con seaborn  
_ = sns.distplot(train_df['target'])
```



3. Gráficas relacionales

Las gráficas relacionales son muy útiles para obtener relaciones entre dos o más variables. Estas relaciones pueden ayudarnos a comprender más nuestros datos y probablemente nos ayuden a

crear nuevas variables a partir de las variables existentes.

Este es un paso importante en la exploración de datos y la ingeniería de datos ya que nos permiten visualizar los siguientes tipos de gráficos:

a) Gráficos lineales

b) Gráficos de dispersión

c) Histogramas 2D, diagramas hexadecimales y diagramas de contorno

d) Diagramas pares

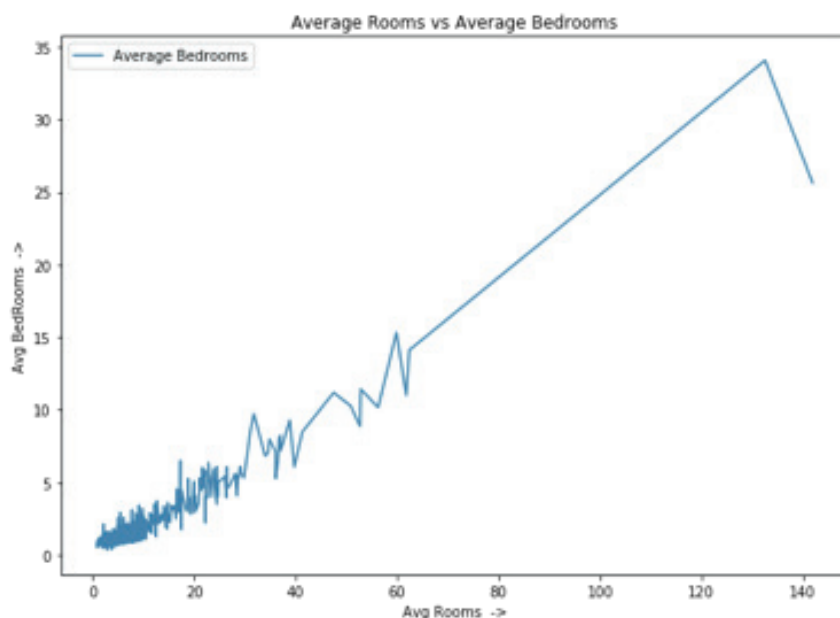


Revisemos a continuación cada uno de ellos con mayor detalle:

a) Gráficos lineales

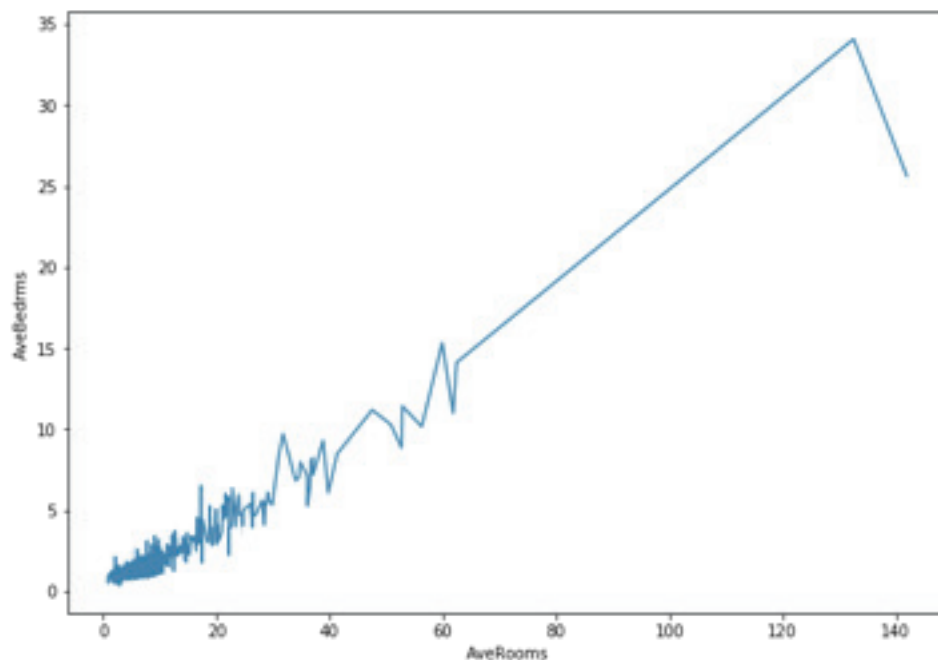
Los gráficos de líneas son útiles para verificar la relación lineal, e incluso las relaciones cuadráticas, exponenciales y similares, entre dos variables.

```
plt.plot('AveRooms', 'AveBedrms', data=data,
         label="Average Bedrooms")
plt.legend() # Para mostrar la etiqueta de la variable del eje y dentro
de la gráfica
plt.title("Average Rooms vs Average Bedrooms")
plt.xlabel("Avg Rooms ->")
plt.ylabel("Avg BedRooms ->");
```



con seaborn

```
_ = sns.lineplot(x='AveRooms', y='AveBedrms', data=train_df)
```



b) Gráficos de dispersión

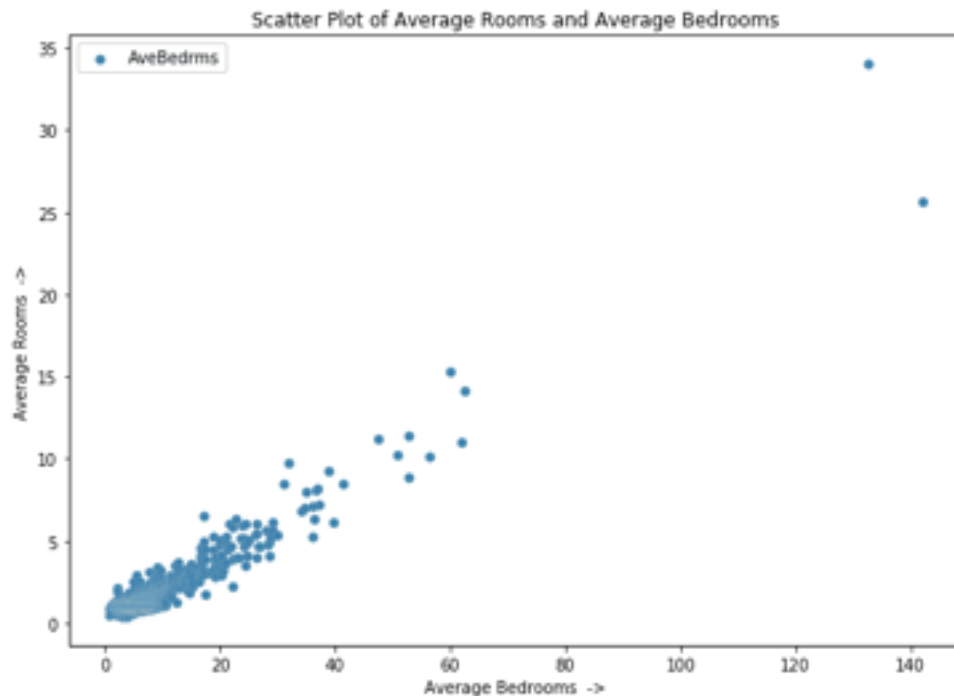
No todas las relaciones entre dos variables son lineales, en realidad solo unas pocas lo son. Estas variables también tienen algún componente aleatorio que las hace casi lineales, y otros casos tienen una relación totalmente diferente que habríamos tenido dificultades para mostrar con gráficos lineales.

Además, si tenemos muchos puntos de datos, el diagrama de dispersión puede ser útil para **verificar si la mayoría de los puntos de datos están concentrados en una región o no, ¿hay valores atípicos** con respecto a estas dos o tres variables, etc.?

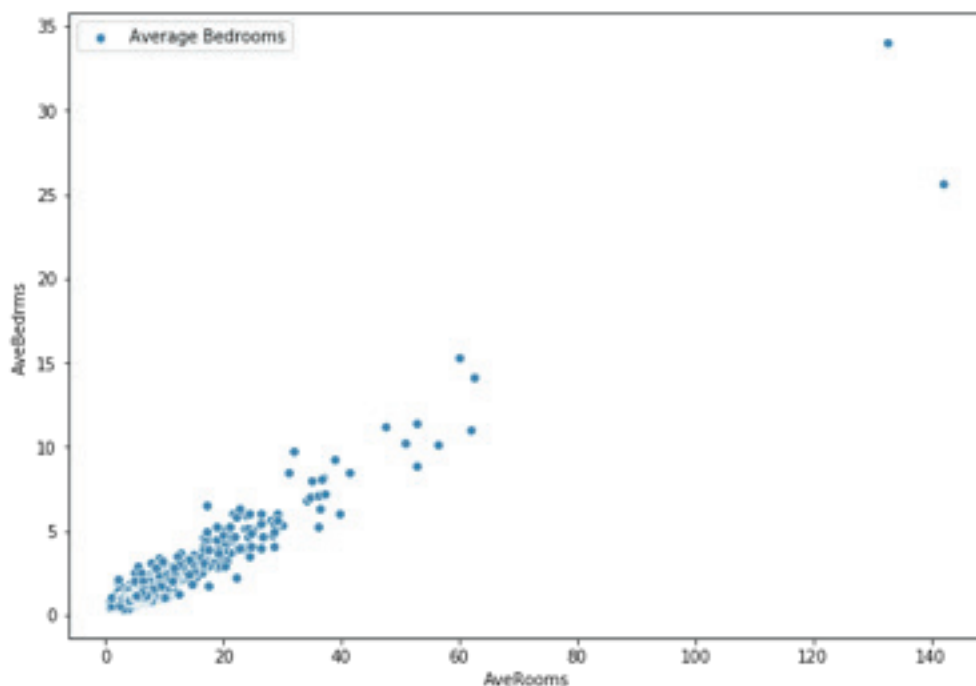
```
from matplotlib.pyplot import figure
figure(figsize=(10, 7))

plt.scatter('AveRooms', 'AveBedrms', data=data,
            edgecolors='w', linewidths=0.1)

plt.title("Scatter Plot of Average Rooms and Average Bedrooms")
plt.xlabel("Average Bedrooms ->")
plt.ylabel("Average Rooms ->");
```

```
# con seaborn
from matplotlib.pyplot import figure
figure(figsize=(10, 7))
sns.scatterplot(x='AveRooms', y='AveBedrms', data=train_df,
               label="Average Bedrooms");
```



c) Histogramas 2D, diagramas hexadecimales y diagramas de contorno

Los histogramas 2D y los gráficos hexadecimales se pueden usar para verificar la densidad

relativa de datos en una posición particular.

Los **gráficos de contorno** se pueden usar en las siguientes acciones:

- Para graficar datos **3D en 2D**, o para graficar datos **4D en 3D**.
- Una línea de contorno (o franja de color en el contorno relleno) nos dice la ubicación donde la función tiene un valor constante.
- Nos familiariza con todo el panorama de las **variables utilizadas** en el gráfico.
- Por ejemplo, **se puede usar para graficar la función de costos con respecto a diferentes thetas en aprendizaje profundo**. Pero para hacerlo necesita muchos datos, para ser precisos.
- En cuanto a **graficar todo el paisaje**, necesitarás datos para todos los puntos en ese paisaje. Y si tienes una función para ese paisaje, puedes hacer fácilmente estos gráficos calculando los valores manualmente.



```
from matplotlib.pyplot import figure
figure(figsize=(10, 7))
plt.hist2d('MedInc', 'target', bins=40, data=train_df)
plt.xlabel('Median Income ->')
plt.ylabel('Target ->')
plt.suptitle("Median Income vs Target", fontsize=18);
```

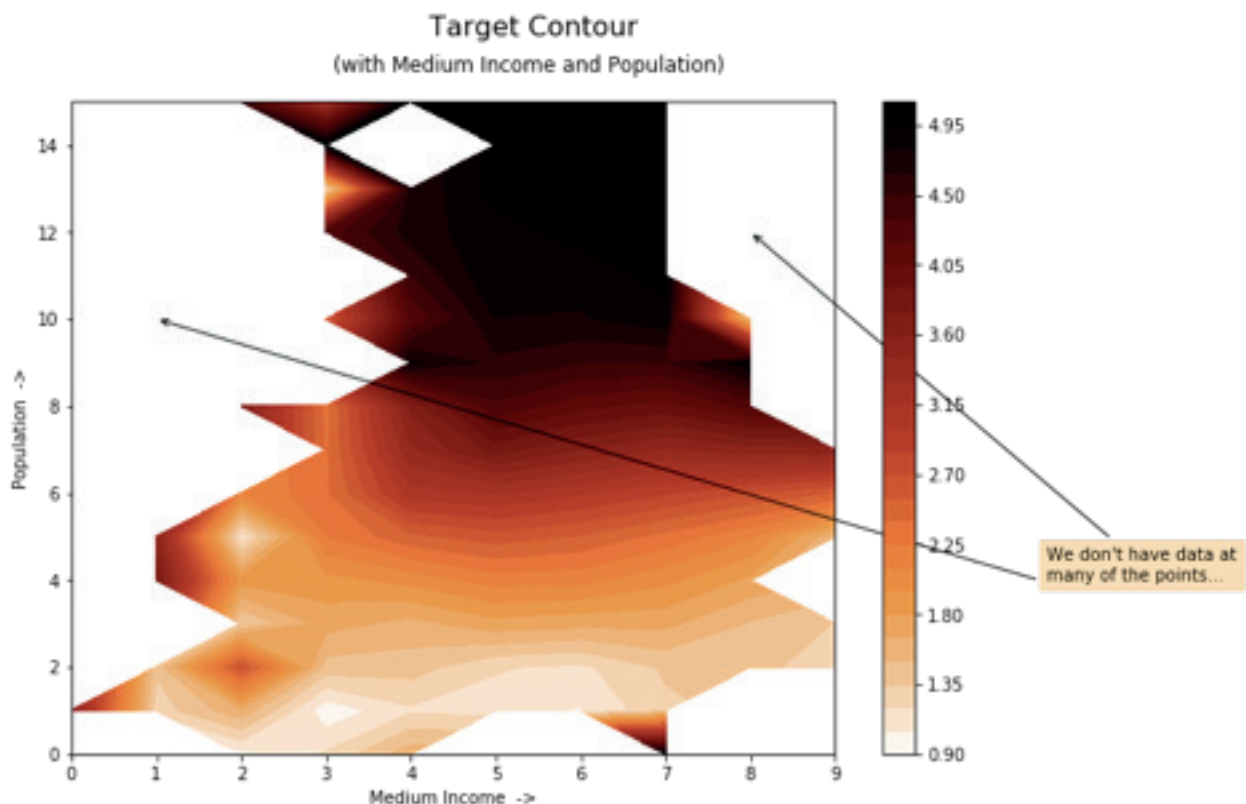


Nota: Este tipo de gráficas no se pueden obtener directamente en seaborn.

c) Gráfica de contorno

Un diagrama de contorno es una forma de visualizar datos 3D en un diagrama 2D. En matplotlib hay dos métodos disponibles, a saber `.contour` y `.contourf`. El primero hace contornos de línea y el segundo hace contornos rellenos. Puede pasar una matriz 2D de valores z o pasar en dos matrices 2D X, Y para valores x y valores y y una matriz 2D para todos los valores z correspondientes.

```
# Para la gráfica de contorno
from matplotlib.pyplot import figure
figure(figsize=(10, 7))
plt.contourf(Z, levels=30, cmap="gist_heat_r")
plt.colorbar()
plt.suptitle("Target Contour", fontsize=16)
plt.title("(with Medium Income and Population)",
          position=(0.6, 1.03))
plt.xlabel("Medium Income ->")
plt.ylabel("Population ->")
```

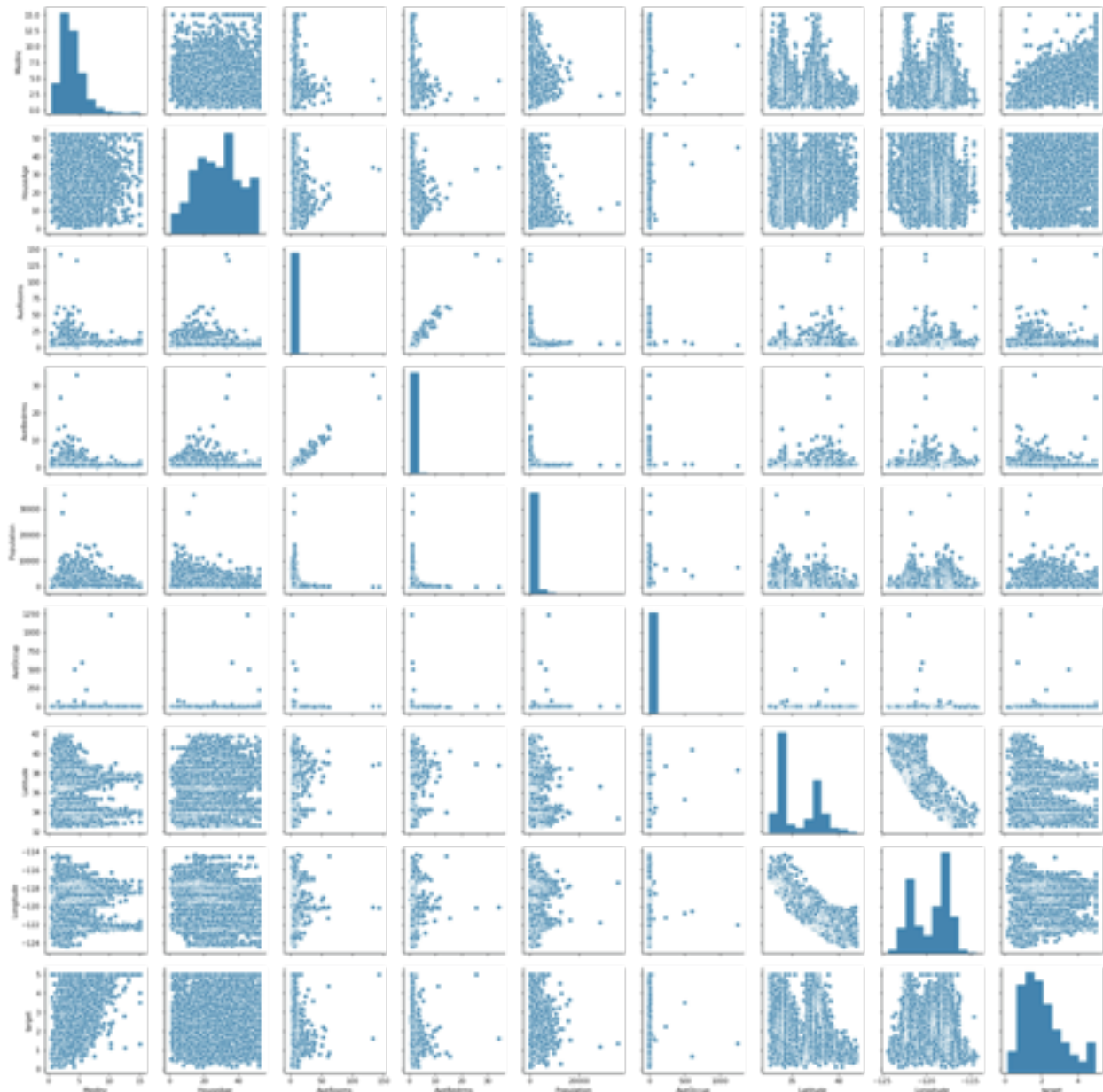


d) Diagramas pares

seaborn proporciona un **diagrama de pares de métodos** con el que puede trazar todos los diagramas relacionales posibles de una sola vez. Se puede usar para ver rápidamente la relación

entre todas las variables en sus datos, y también la distribución de cada variable.

```
_ = sns.pairplot(train_df)
```



4. Gráficos categóricos

Los gráficos categóricos también son necesarios en el paso de Exploración de datos, ya que nos informan sobre cómo se distribuyen las diferentes clases de una variable en el conjunto de datos. Si tenemos datos suficientes, podemos sacar conclusiones de estos gráficos para diferentes clases de esa variable.

Son ejemplos de estos gráficos los siguientes:

a) Gráfica de barra

b) Diagrama de caja

c) Gráfica de violín

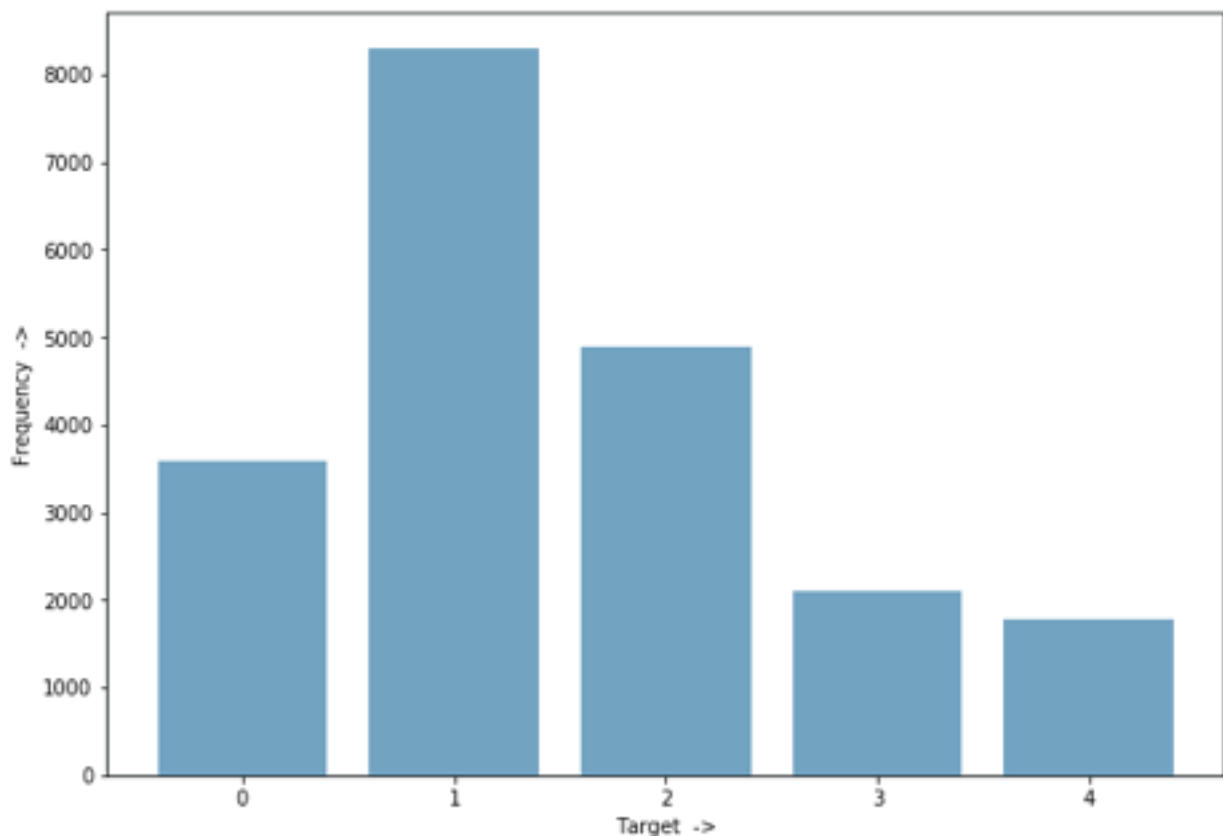
Revisemos a continuación cada uno de ellos con mayor detalle:

a) Gráfico de barras

Los gráficos de barras se pueden usar para contrastar entre categorías donde sus alturas representan algún valor específico para esa categoría.

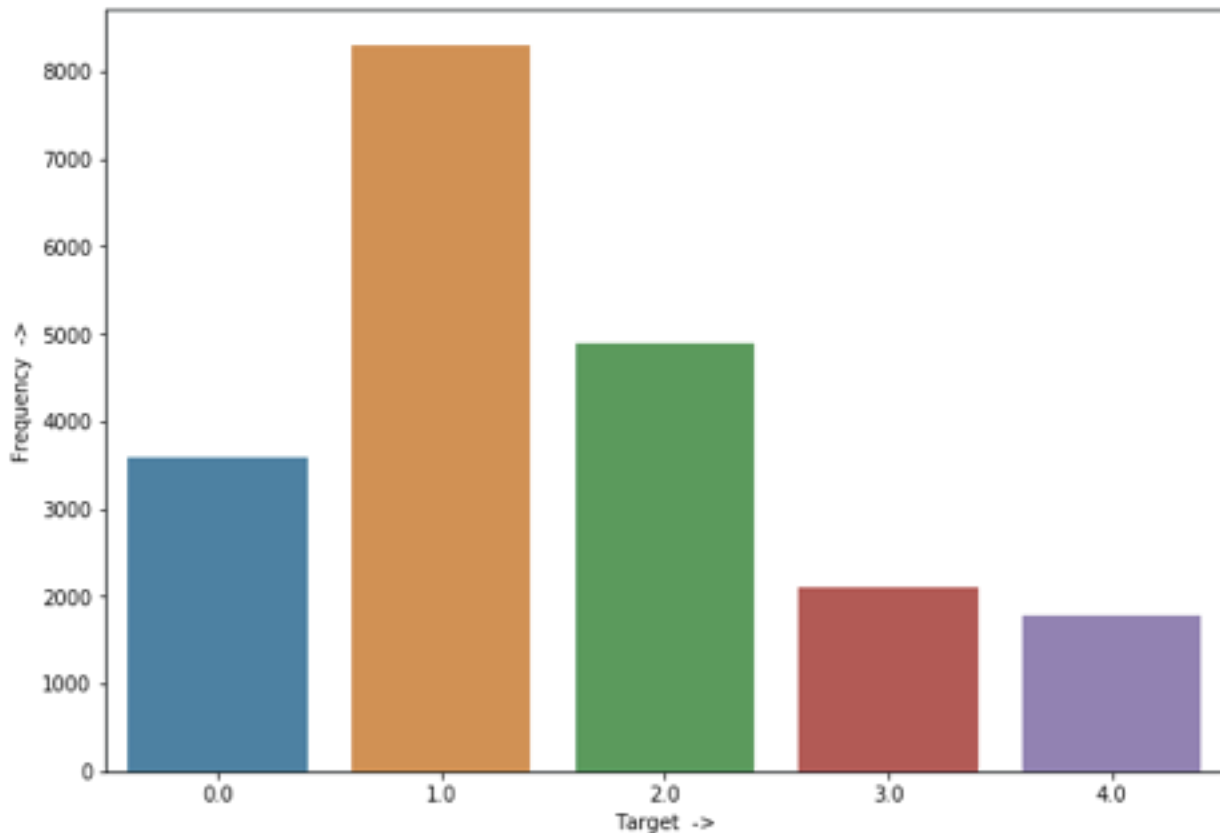
```
from matplotlib.pyplot import figure
figure(figsize=(10, 7))

plt.bar(np.sort(data.unique()), data.value_counts().sort_index(),
        alpha=0.7) # Es posible que deba ordenar;
                  # Tenga cuidado con los valores que se trazan entre sí
plt.xlabel("Target ->")
plt.ylabel("Frequency ->");
```



Seaborn

```
from matplotlib.pyplot import figure
figure(figsize=(10,7))
sns.barplot(np.sort(data.unique()),data.value_counts().sort_index())
plt.xlabel("Target ->")
plt.ylabel("Frequency ->");
```



b) Diagrama de caja

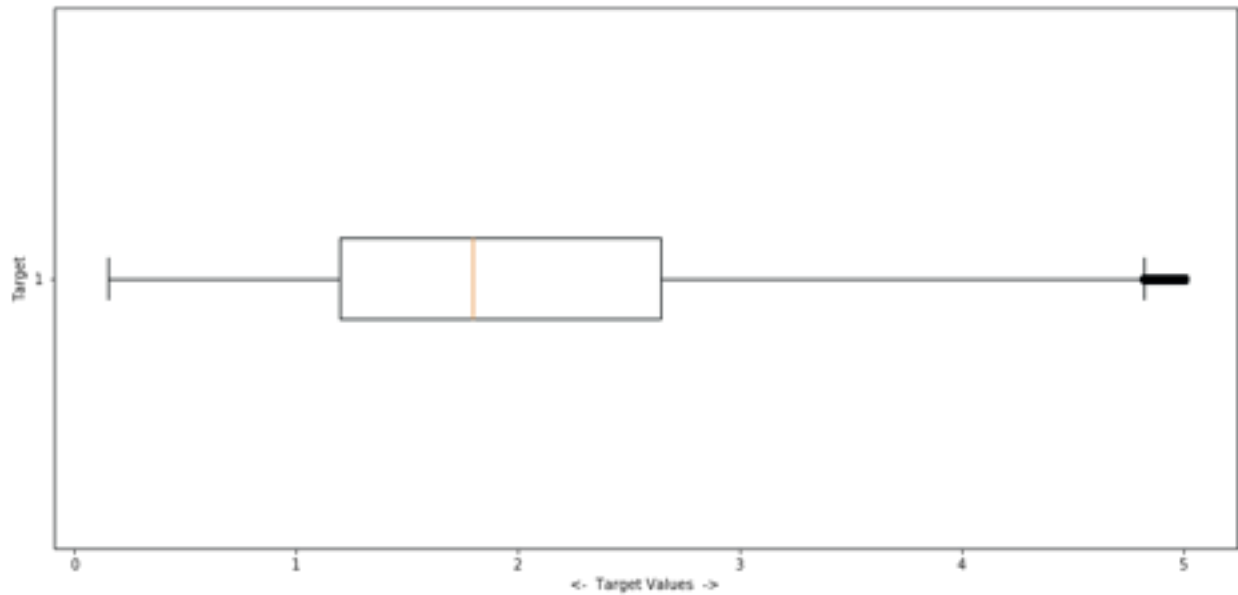
El diagrama de caja es una **versión estadística del diagrama de distribución**. Nos da una gama de diferentes cuartiles, medias y extremos. Algunos casos de uso posibles pueden ser que con él puede identificar variables en las que puede encontrar valores atípicos si algunos puntos están fuera del rango de los bigotes de caja, o puede verificar la distorsión en la distribución mediante la ubicación relativa de la caja del medio en la gráfica.



```
from matplotlib.pyplot import figure
figure(figsize=(15, 7))

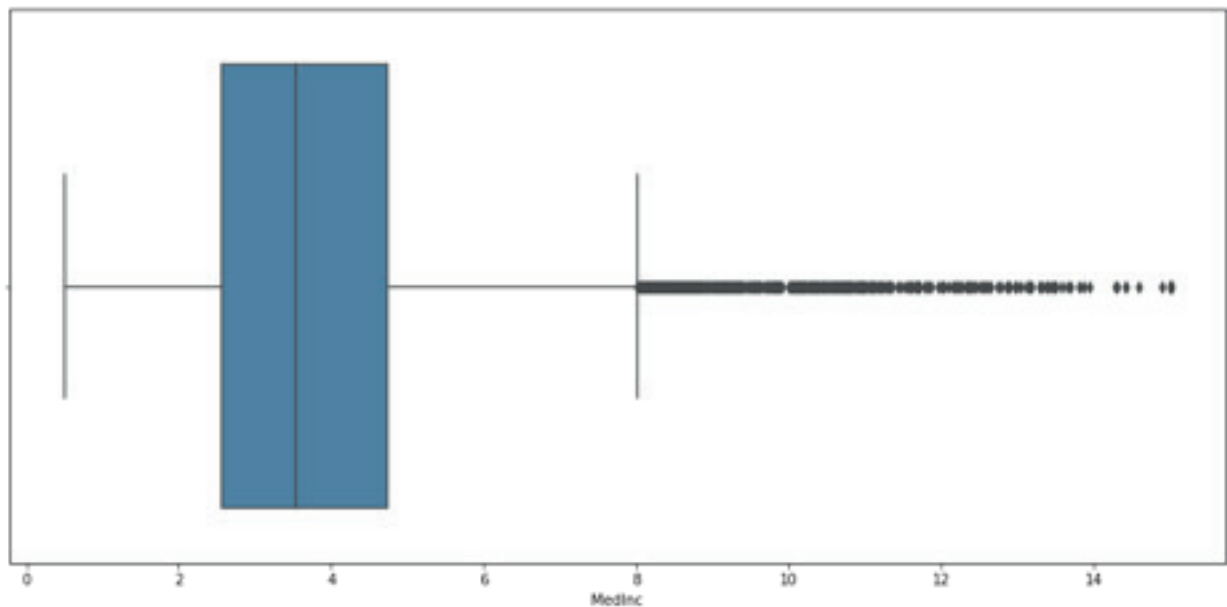
plt.boxplot(train_df['target'], vert=False)
plt.xlabel("<- Target Values ->")
```

```
plt.ylabel("Target");
```



```
# Com Seaborn:
from matplotlib.pyplot import figure
figure(figsize=(15, 7))

sns.boxplot(train_df['MedInc']);
```



c) Gráfica de violín

La **gráfica de violín** es una extensión de la trama de caja. También tiene **indicadores de medias, extremos y posiblemente diferentes cuartiles también**. Además de estos, también muestra la

distribución de probabilidad de la variable, en ambos lados.

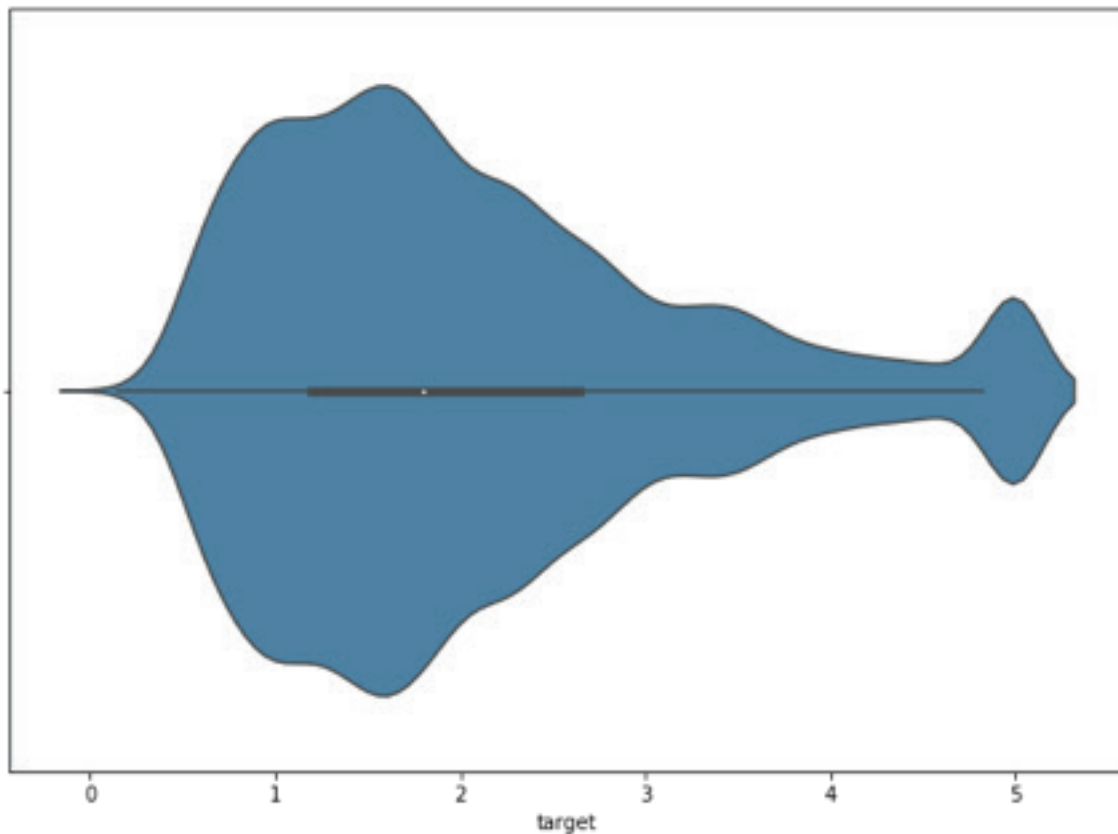
```
from matplotlib.pyplot import figure
figure(figsize=(10, 7))

plt.violinplot(train_df['target'])
plt.title("Target Violin Plot")
plt.ylabel("Target values ->");
```



```
# Con Seaborn
from matplotlib.pyplot import figure
figure(figsize=(10, 7))

sns.violinplot(train_df['target']);
```



5. Gráficas múltiples

Se pueden hacer tantos gráficos como se necesiten utilizando el método `plt.subplots` o agregando manualmente los ejes a la figura especificando sus coordenadas de caja, o utilizando el método `plt.GridSpec` (). Es decir:

1. Ya sea usando: `fig, axess = plt.subplots (ncols = 2, nrows = 4)` y luego puede dibujar en cualquiera de estos ejes accediendo a ellos como `axess [col_num] [row_num]`, y luego usar cualquiera de los métodos de los ejes para dibujar en ellos.
2. O utilizando el método `plt.axes` () que proporciona una lista de valores de cuatro por ciento que da [izquierda, abajo, ancho, alto] de ejes para hacer en la figura. Por ejemplo: `plt.axes ([0.1, 0.1, 0.65, 0.65])`.
3. O utilizando el método `plt.GridSpec` (). Como `grid = plt.GridSpec (n_row, n_col)`. Y ahora, al hacer los ejes mediante el método `plt.subplot` (), puede usar esta cuadrícula como una matriz 2D para seleccionar cuántas y qué cuadrículas usar para hacer los ejes actuales, uno. Por ejemplo, `plt.subplot (grid [0 , :])` seleccionará toda la primera fila como un eje.

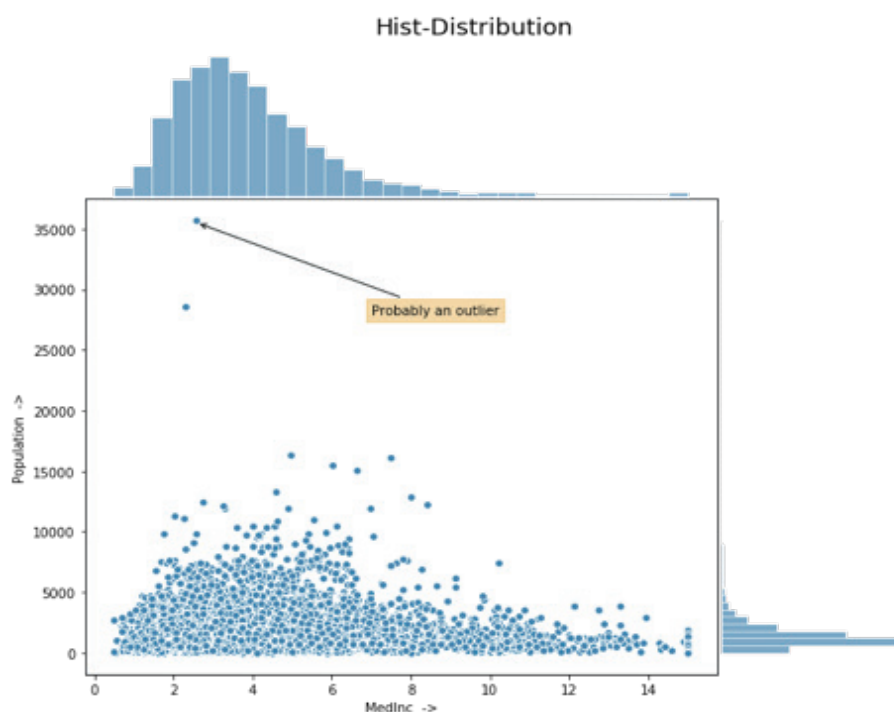

```

plt.figure(1, figsize=(10, 8))
plt.suptitle("Hist-Distribution", fontsize=18, y=1)# Ahora hagamos algunos ejes en esta figura
axScatter = plt.axes([0.1, 0.1, 0.65, 0.65])
    # [izquierda, abajo, ancho, alto] en valores porcentuales
axHistx = plt.axes([0.1, 0.755, 0.65, 0.2])
axHisty = plt.axes([0.755, 0.1, 0.2, 0.65])
axHistx.set_xticks([])
axHistx.set_yticks([])
axHisty.set_xticks([])
axHisty.set_yticks([])
axHistx.set_frame_on(False)
axHisty.set_frame_on(False)
axScatter.set_xlabel("MedInc ->")
axScatter.set_ylabel("Population ->")

# Vamos a trazar en estos ejes:
axScatter.scatter(x, y, edgecolors='w')
axHistx.hist(x, bins=30, ec='w', density=True, alpha=0.7)
axHisty.hist(y, bins=60, ec='w', density=True, alpha=0.7,
    orientation='horizontal')
axHistx.set_ylabel("")

# Agregar anotaciones:
axScatter.annotate("Probably an outlier", xy=(2.6, 35500),
    xytext=(7, 28000),
    arrowprops={'arrowstyle':'->'},
    bbox={'pad':4, 'facecolor':'orange', 'alpha':
        0.4, 'edgecolor':'orange'});

```



A continuación, revisaremos una de las librerías más populares dentro de Python

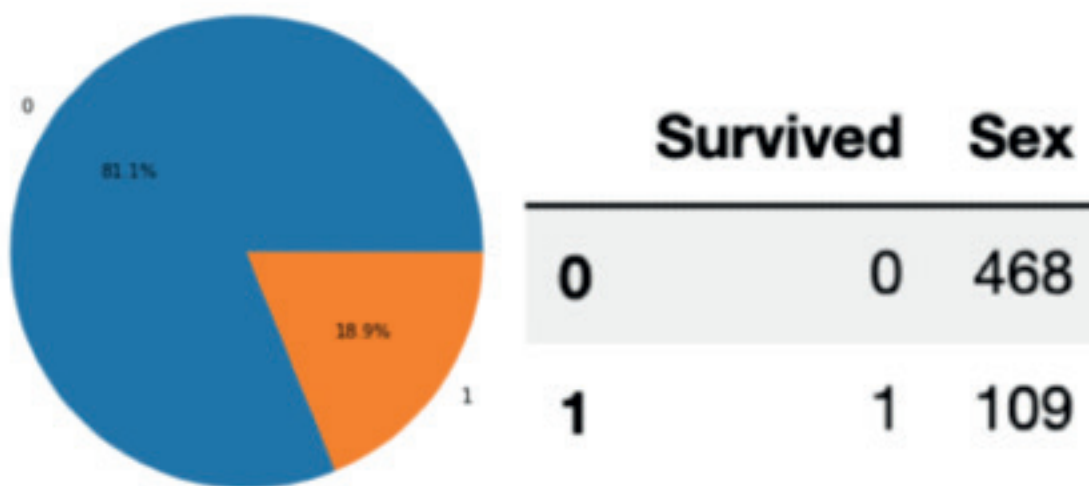
Una guía rápida para la visualización de datos con Plotly

Cuando se necesita contar una historia, expresar su opinión y convencer tal vez a un colega, supervisor o incluso a un CEO la **visualización de datos** puede ser un gran aliado; y aunque existen muchas herramientas, actualmente algunas empresas se han inclinado por **herramientas open source** (código abierto) y el **lenguaje Python** se ha vuelto un referente. Aunque existen muchas librerías en este lenguaje, hay uno que ha empezado a cobrar importancia y nos referimos a Plotly.



En este apartado, nos enfocaremos a presentar un ejemplo usando **el conjunto de datos de la tragedia del Titanic usando Plotly**.

Imagina que estás tratando de explicar que menos del 20% de los hombres sobrevivieron a la tragedia del Titanic. ¿Cómo se vería?

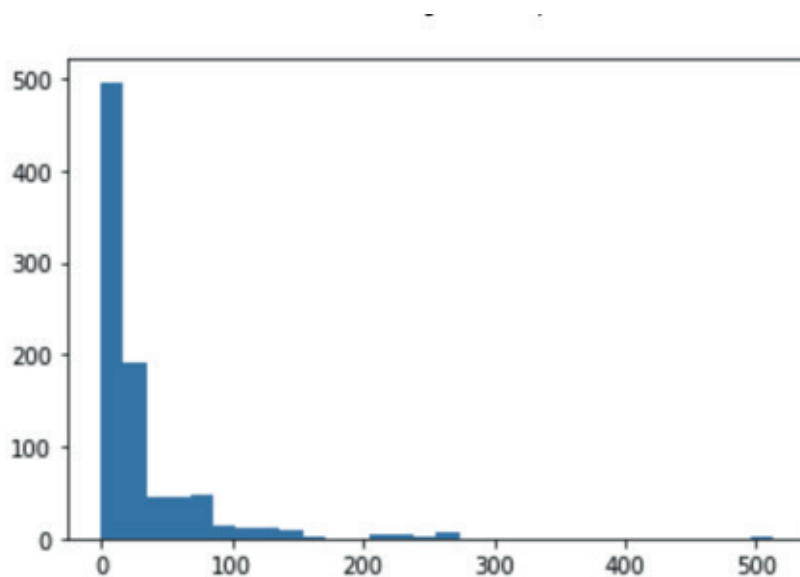


Supervivencia masculina (conjunto de datos del Titanic)

Como puedes ver, **las visualizaciones** juegan un papel importante en la explicación de sus datos. Mostrar números en una pantalla no se aplica realmente a algunas personas. Las imágenes hacen la diferencia.

Las **visualizaciones** ayudan al científico de datos a comprender y descubrir historias ocultas en los datos. Un histograma simple puede decirle más en una instancia que tratar de averiguar la distribución usted mismo.

Aquí hay una distribución de las tarifas del conjunto de datos del Titanic.



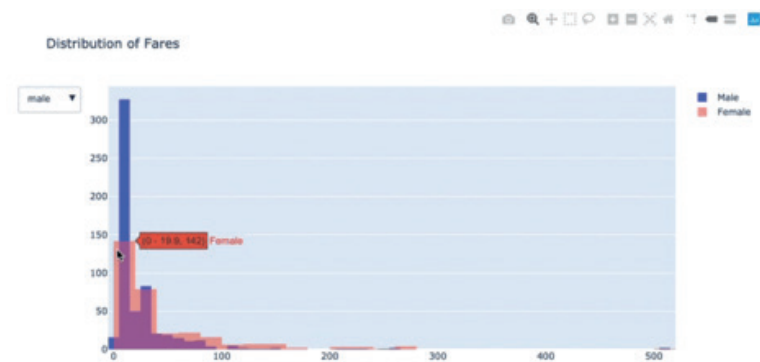
Distribución de tarifas (conjunto de datos del Titanic)

Graficar esto directamente nos dice que la mayoría de las tarifas de los clientes oscilan por debajo de la marca de 100 dólares. Esta es una buena información, pero no nos cuenta toda la historia.

“La visualización le da respuestas a preguntas que no sabía que tenía.”
- **Ben Schneiderman**

Las **visualizaciones interactivas** son populares para agregar una tonelada de información en la parte superior de las gráficas que se están presentando, desbloquear posibilidades y hacer que se vea 10 veces más genial. Es difícil ponerlo en palabras, visualicemos lo que estamos tratando de decirle.

Aquí está el mismo diagrama de distribución para las tarifas en el conjunto de datos del Titanic una vez más.



Distribución de tarifas (conjunto de datos del Titanic)

Observa cómo podemos pasar el cursor sobre el histograma y saber qué **distribución** (hombre o mujer) a los que nos referimos, conoce los **valores** sobre los que estamos pasando el cursor, **acerca y aleja** para profundizar en los valores y **filtra** para el género masculino o femenino. Ya puedes imaginar cuán poderoso es esto y cómo esto contribuirá a tu **narración de historias** en los datos. La librería que hizo esto posible se llama **Plotly**.

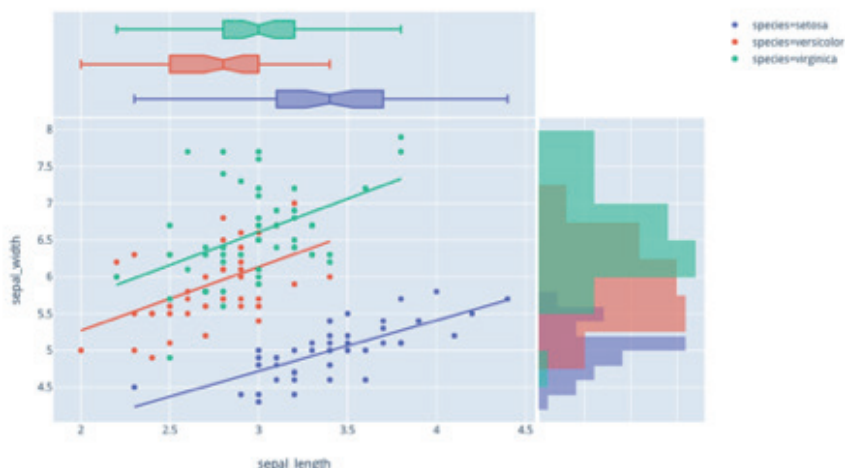
Plotly es una librería de Python que se utiliza para las **visualizaciones interactivas** de datos. **Plotly** te permite trazar gráficos interactivos superiores a lo que se puede graficar con las librerías Matplotlib o Seaborn.

¿Qué tipo de gráficos traza gráficamente?

- Todos los **gráficos de Matplotlib y Seaborn**.
- **Gráficos estadísticos** que incluyen, entre otros, gráficas categóricas y gráficos de árbol de probabilidad.
- **Gráficos científicos** que nunca pensó, que van desde Gráficos de Red a Gráficos de Radar.
- **Gráficos financieros** que son útiles para el análisis de series de tiempo, los ejemplos incluyen velas, embudos y gráficos de viñetas.
- **Mapas geológicos y diagramas tridimensionales** que le permiten interactuar con ellos.



```
px.scatter(iris, x="sepal_width", y="sepal_length", color="species", marginal_y="histogram",  
           marginal_x="box", trendline="ols")
```



¿Por qué es Plotly tan popular?

- Las **gráficas** son interactivas.
- Las **gráficas** son más atractivas que Matplotlib / Seaborn.
- Ofrece una **visualización más detallada** que ayuda a explorar, comprender y comunicar los datos.

- Proporciona la **máxima personalización para las gráficas**, incluidas: Agregar controles deslizantes y filtros.
- Una **base de código** mucho más limpia y comprensible
- **Respaldado por una compañía llamada Plotly**, que hace visualizaciones interactivas basadas en web y aplicaciones web

¿Cómo se instala Plotly?

Tener Python instalado,

```
pip install plotly
pip install cufflinks
```

Trabajaremos con **Jupyter Notebook** en esta guía. Este es un ejemplo:

```
from plotly.offline import init_notebook_mode, iplot
import plotly.graph_objects as go
import cufflinks as cf
init_notebook_mode(connected=True)
```

Debido a cómo opera Plotly, guarda la gráfica en un archivo html separado y lo abre en una ventana diferente directamente. Esto sucederá cuando se ejecute el código en la consola ó terminal. Por lo tanto, usamos el modo `plotly.offline`, `iplot` y `init_notebook` para ayudarnos a trazar los gráficos en el propio Jupyter Notebook.

En esta guía, nos centraremos en la sintaxis de gráficas originales porque proporciona la máxima personalización en el gráfico. **Plotly Express** y **Cufflinks** proporcionan una mejor alternativa para graficar fácilmente el código, pero no ofrecen muchas herramientas como lo hace la sintaxis original.

Definiendo lo que necesitas para graficar

Antes de graficar algo, se necesita saber lo que se está tratando de graficar.

Realiza las siguientes preguntas cómo:

- ¿Qué tipo de información estoy tratando de transmitir?
- ¿Estoy graficando valores numéricos o valores categóricos?
- ¿Cuántas variables estoy tratando de graficar?



Después de responder algunas de estas preguntas, puedes comenzar a planificar la gráfica. Vamos a ver cómo realizar una **visualización con Plotly** usando el conjunto de datos del Titanic.

Nuevamente, el **objetivo principal** de este conjunto de datos es estudiar cuáles son los factores que afectaron la supervivencia de una persona a bordo del Titanic.

Lo primero que hay que **considerar es mostrar cuántos pasajeros sobrevivieron al accidente**. Por lo tanto, visualizar la columna Survived (Sobrevivido) en sí será un buen comienzo.

Datos, Diseño y Figura

En Plotly, definimos los objetos de gráfico que se graficarán. Los 3 parámetros principales necesarios para un gráfico son los parámetros **Datos, Diseño y Figura**.

Por lo tanto, necesitamos definirlos de una manera clara y concisa donde alguien más pueda entender lo que estamos tratando de graficar. Como estamos graficando solo la columna Survived, nuestros datos serán los siguientes:

Gráfico circular

```
#etiquetas
lab = df["Survived"].value_counts().keys().tolist()
#valores
val = df["Survived"].value_counts().values.tolist()
trace = go.Pie(labels=lab,
               values=val,
               marker=dict(colors=['red']),
               # Establecer valores a
               hoverinfo="value"
               )
data = [trace]
```

Diseño

El diseño significa literalmente lo que significa, el diseño de la gráfica. Aquí, puede definir títulos de la gráfica, títulos de eje x e y, mostrar **leyendas**, activar **filtros / controles** deslizantes y muchas más personalizaciones para el gráfico. Como solo estamos dibujando un gráfico circular para una sola columna, no lo haremos demasiado elegante.

```
layout = go.Layout (title = "Survived Distribution")
```

Estamos creando un **objeto** de diseño que contiene solo el parámetro de título aquí.

Figura

La figura es lo que está tratando de graficar, toma los datos y el parámetro de diseño por defecto que ya se habían definido.


```
fig = go.Figure(data = data,layout = layout)
```

Entonces podemos mostrar esta gráfica literalmente dibujándola con Plotly.

```
ipplot(fig)
```

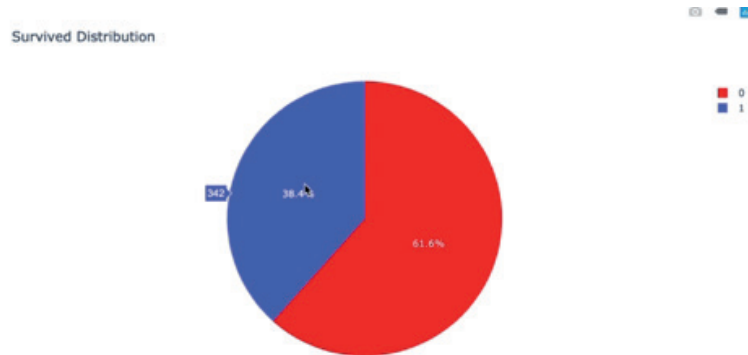


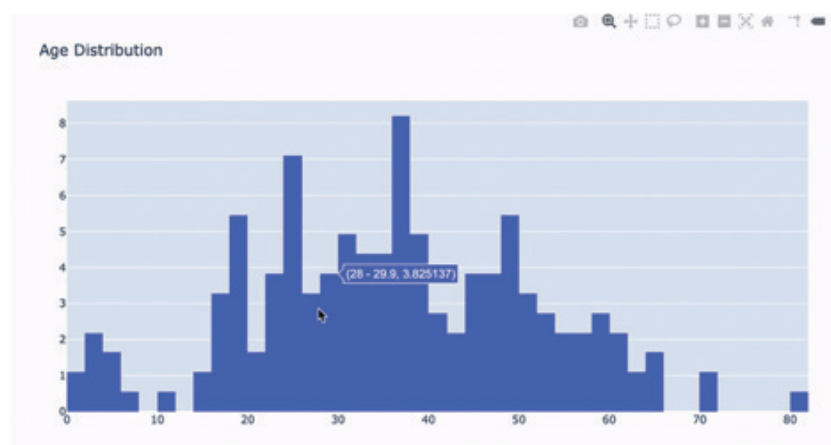
Gráfico circular

Voila, habías planeado con éxito un **gráfico interactivo**. Observa cómo se pueden ver los valores reales al pasar el cursor sobre cada sección. Esto es extremadamente útil cuando se presentan visualizaciones de manera limpia y profesional.

A continuación, exploraremos algunas de las columnas **numéricas** que son las columnas Edad (Age) y Tarifa (Fare). Al graficar columnas numéricas solas, pensamos en un diagrama de distribución. En este caso, usaremos un **histograma**.

Histograma

```
# definiendo los datos
trace = go.Histogram(x=df['Age'],nbinsx=40,histnorm='percent')
data = [trace]# defining layout
layout = go.Layout(title="Age Distribution")# definir figura y graficar
fig = go.Figure(data = data,layout = layout)
ipplot(fig)
```



Histograma

Perfecto. Aquí podemos ver claramente cómo se distribuye la edad de los pasajeros. Ten en cuenta que podemos ajustar 2 parámetros útiles para Histogramas que son los siguientes:

- **Histnorm**: valor para trazar el histograma, se estableció como “porcentaje” para que se muestre el porcentaje de las ubicaciones que contribuyen a la distribución. Si se deja en blanco, muestra el recuento del contenedor por defecto.
- **nbinsx**: número de contenedores para los valores a distribuir. Un mayor número de contenedores tiende a darle una distribución más detallada.

Hasta ahora, solo habíamos dibujado **gráficos con una variable**. Ahora estudiaremos la relación entre esas variables graficándolas unas contra otras.

Veamos la relación entre nuestras **columnas Edad (Age) y Tarifa (Fare)**. Esto sería útil para responder la siguiente pregunta: ¿las personas mayores tienden a comprar boletos de tarifa más caros?

Al trazar 2 valores numéricos uno contra el otro, un diagrama de dispersión será un buen lugar para comenzar.

Diagrama de dispersión

```
#definiendo los datos
trace = go.Scatter(x = df['Age'],y=df['Fare'],text = df['Survived'],mode='markers')
data=[trace]#defining layout
layout = go.Layout(title='Fare Vs Age Scatter
Plot',xaxis=dict(title='Age'),yaxis=dict(title='Fare'),hovermode='closest')#defining figure and
plotting
figure = go.Figure(data=data,layout=layout)
iplot(figure)
```

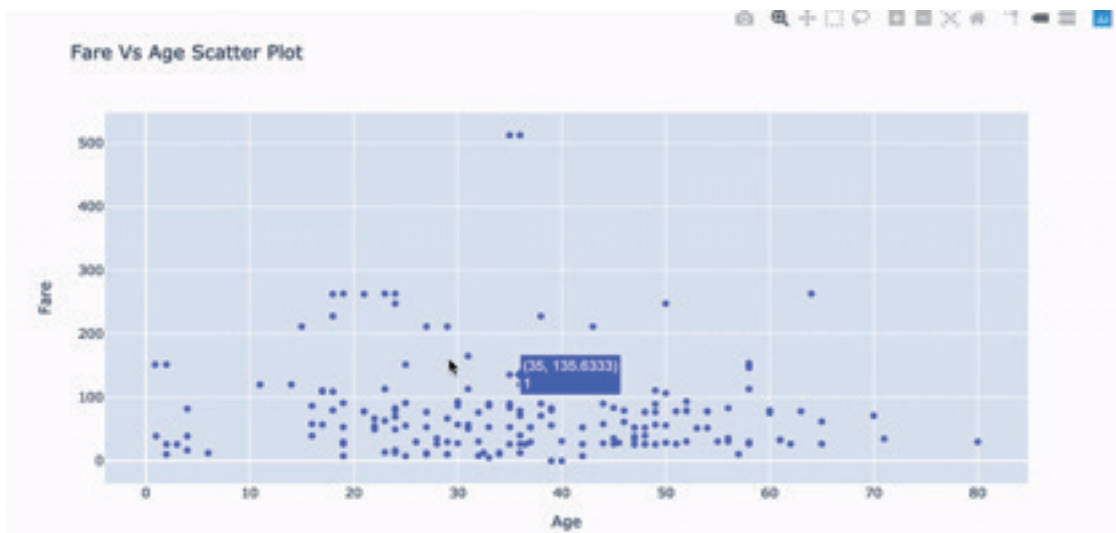



Gráfico de dispersión



Ten en cuenta que **se agregaron algunas características sutiles** en este gráfico, que son los títulos de los ejes “X e “Y”, así como el valor que se mostrará al pasar el mouse sobre el punto. Se puede personalizar el valor de visualización cambiando el parámetro de texto en el código.

Observando la gráfica, realmente no vemos una relación lineal entre las columnas. Las **tarifas** tienden a alcanzar un precio máximo por debajo de los 300 dólares.

También observamos que las **personas mayores** también compran tarifas baratas. Además de eso, no podemos decir mucho simplemente confiando solo en este gráfico.

Se prohíbe la reproducción total o parcial de esta obra por cualquier medio sin previo y expreso consentimiento por escrito del Instituto Tecnológico y de Estudios Superiores de Monterrey.

D.R. © Instituto Tecnológico y de Estudios Superiores de Monterrey, México. 2020 Ave. Eugenio Garza Sada 2501 Sur Col. Tecnológico C.P. 64849 Monterrey, Nuevo León | México



**Tecnológico
de Monterrey**