Hipótesis y correlación

Hipótesis y correlación **1**
¿Qué aprenderás? 2
Introducción 2
Refactorizando nuestros gráficos 4
FacetGrid 9
Iniciar un objeto FacetGrid 10
Aplicar gráficos a nuestro objeto 11
Scatterplots 14
Refactorizando el gráfico 15
Comprensiones de Lista 16
Refactorización con seaborn 19
Boxplot 23
Preguntas de cierre 26

¡Comencemos!

¿Qué aprenderás?

- Reconocer las funcionalidades avanzadas de gráficos estáticos mediante seaborn.
- Aprender a realizar gráficos que muestren de forma estratificada el comportamiento de subconjuntos de elementos en la muestra.

Introducción

A lo largo de esta lectura, aprenderemos sobre el proceso inferencial de la estadística. Por inferencial haremos referencia a los mecanismos que nos permiten aprender sobre los datos cuando tenemos información incompleta, situación que regularmente es la norma.

Anteriormente aprendimos sobre las variables aleatorias, leyes matemáticas que nos ayudan a entender el comportamiento de la muestra en base a una función probabilística. También aprendimos a generar puntajes z que miden la ubicación de una observación respecto a la media, medida en desviaciones estándares.

Por consiguiente, tenemos todos los elementos necesarios para generar inferencias y pruebas de hipótesis, solo nos falta incluirlos.
También aprenderemos a refactorizar gráficos mediante `seaborn`, una librería que sintetiza las buenas prácticas del análisis en una serie de funciones que trabajan considerando `pandas`, `numpy` y `matplotlib`.
Con los gráficos de `seaborn` abordaremos los diagramas de dispersión y la correlación, piedra angular del trabajo estadístico moderno.

Para ello seguiremos trabajando con la base de datos *Quality of Government*.

```
%matplotlib inline
# importamos la triada de Data Science
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# importamos scipy.stats que ayudará a generar distribuciones
import scipy.stats as stats
# importamos seaborn, siguiendo la convención de renombrarlo como sns
import seaborn as sns
# archivo con funciones de visualización
import lec4_graphs as gfx
# evitar warnings y deprecaciones
import warnings
warnings.filterwarnings(action="ignore")
plt.style.use('seaborn') # gráficos estilo seaborn
```

```
plt.rcParams["figure.figsize"] = (6,4) # Tamaño gráficos
plt.rcParams["figure.dpi"] = 200 # resolución gráficos
```

```
# importamos la base de datos
df = pd.read_csv('qog_std_cs_jan18.csv')
```

¡Vamos con todo!

Refactorizando nuestros gráficos

Retomemos el ejemplo trabajado durante unidades anteriores, donde analizamos el Índice de Desarrollo Humano (IDH).

Una de las primeras cosas que observamos es que la medición `undp_hdi` presentaba valores perdidos. Para limpiar nuestra columna, utilizabamos el método `dropna()`.

Posteriormente utilizamos un histograma para observar la distribución de los casos. Generemos la figura de nuevo.

```
plt.hist(df['undp_hdi'].dropna());
```

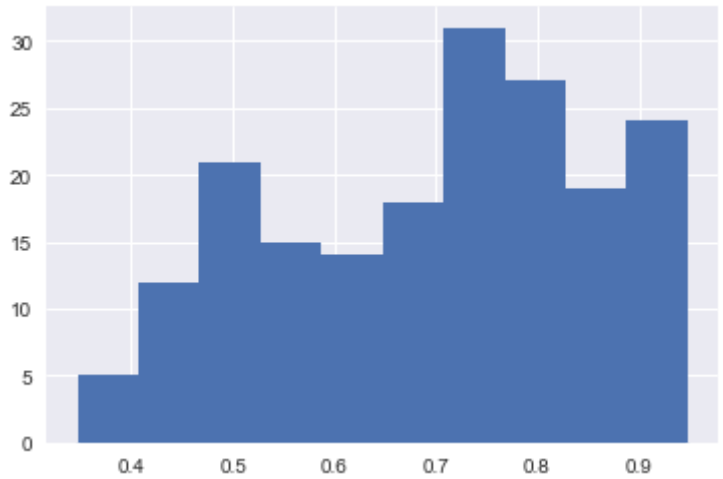

Imagen 1. Resultado histograma *.dropna()*.
Fuente: Desafío Latam.

Resulta que podemos mejorar de forma sustancial la presentación de nuestra figura mediante `seaborn`.
`Seaborn` contiene el método `distplot` que genera el mismo histograma, con una curva de densidad empírica que visualiza cómo se comporta la distribución. Pasaremos una serie de opciones en la función para implementar una serie de mejoras:

- `rug=True` gráfica el posicionamiento específico de cada observación a lo largo del eje X.
- `fit=stats.norm` agrega una curva siguiendo una distribución especificada. En este caso agregamos una curva gaussiana siguiendo $X \sim N(\bar{hdi}, \sigma(hdi))$.
- `axlabel="Indice de Desarrollo Humano` agrega un título en el eje x.
- `color=sienna` cambia el color del histograma y la curva empírica.

```
sns.distplot(df['undp_hdi'].dropna(), rug=True,
             axlabel="Indice de Desarrollo Humano",
             fit=stats.norm, color='sienna').set_title('Distribución del
Índice de Desarrollo Humano');
```

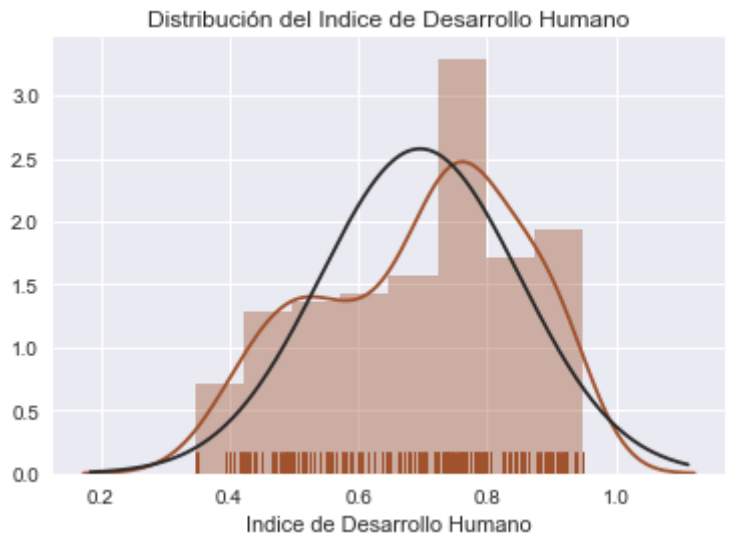

Imagen 2. Distribución del Índice de Desarrollo Humano.
Fuente: Desafío Latam.

Ahora refactorizemos el gráfico de barras que muestra el tamaño de cada región en la muestra. El gráfico original se realizaba con la siguiente línea:

```
# generemos una variable para agregar los nombres asociados a cada
número
df['region_recod'] = df['ht_region'].replace([1, 2, 3, 4, 5, 6, 7, 8, 9,
10],
                                    ['EastEurope', 'LatAm', 'NorthAfrica',
                                     'SubSaharian', 'WesternDem',
'EastAsia',

'SouthEastAsia','SouthAsia','Pacific', 'Caribbean' ])
# Gráficamos en barras el resultado de value_counts.
df['region_recod'].value_counts().plot(kind='bar').set_title('Cantidad
de registros por región');
```

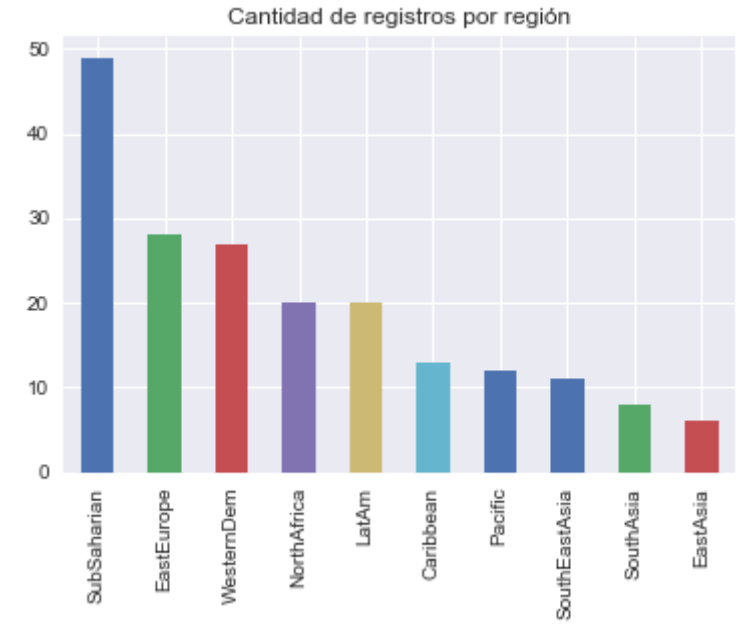

Imagen 3. Cantidad de registros por región.
Fuente: Desafío Latam.

`Seaborn` ofrece el método `countplot` para realizar un gráfico de barras donde se cuenta la cantidad de observaciones en cada valor único. De esta manera, no resulta necesario el pedir `value_counts()` de la columna a analizar.

Incluímos `value_counts()`.index para poder ordenar las barras.

```
sns.countplot(y= df['region_recod'],
              order =
df['region_recod'].value_counts().index).set_title('Cantidad de
registros por región');
```

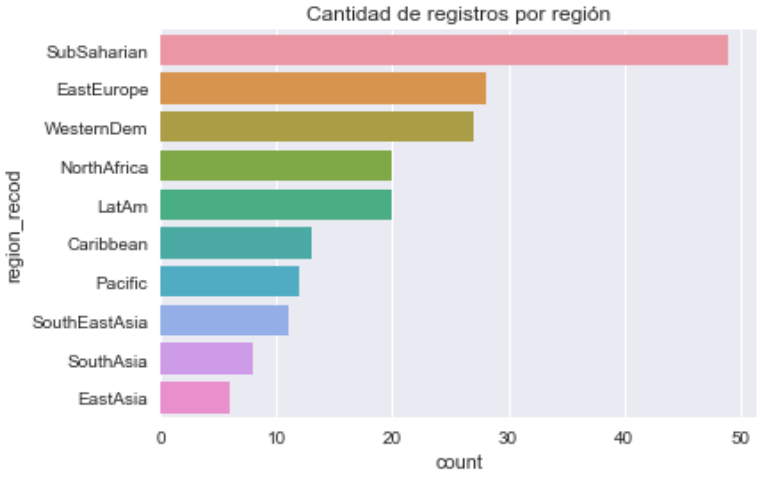

Imagen 4. Cantidad de registros por región.
Fuente: Desafío Latam.

La ventaja de `countplot` es que permite segmentar directa, sin necesidad de estar generando objetos para el preprocesamiento de datos intermedios.

En el siguiente gráfico vamos a graficar la cantidad de democracias o dictaduras **dentro** de cada región.

Para ello incluímos `hue=df['democracies']`, indicando la forma de segmentar.

```
# generamos una recodificación binaria con np.where
df['democracies'] = np.where(df['gol_inst'] <= 2, 'Democracia', 'No
democracia')

sns.countplot(y = df['region_recod'], hue=df['democracies'],
              order =
df['region_recod'].value_counts().index).set_title('Cantidad de
gobiernos en cada región, según tipo de gobierno');
```

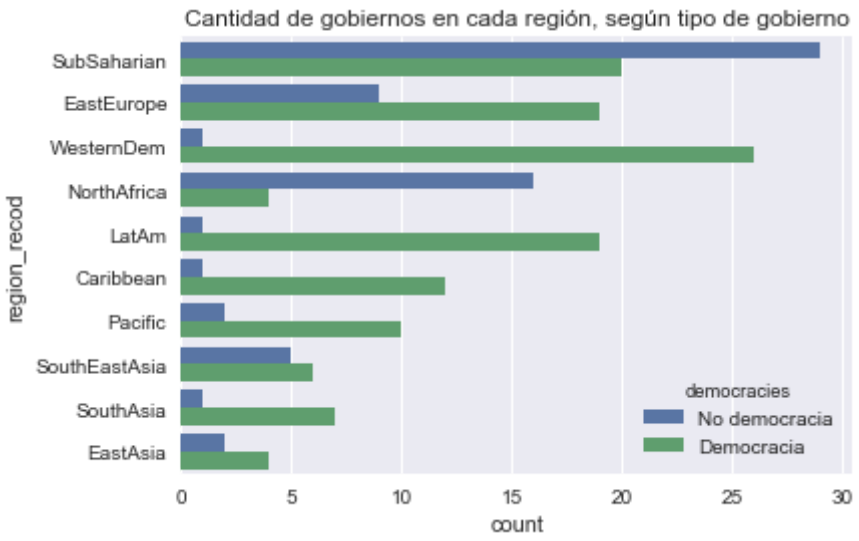

Imagen 5. Cantidad de gobiernos en cada región, según tipo de gobierno.
Fuente: Desafío Latam.

Ahora visualizamos la distribución de los puntajes del IDH. A diferencia de nuestro gráfico de puntos donde visualizamos las medias, el gráfico generado con `swarmplot` muestra cada observación de forma separada.

El método requiere de dos argumentos obligatorios:

- `y=df['region_recod']`: En este caso nuestro eje Y (vertical) va a representar cada uno de los grupos.
- `x=df['undp_hdi']`: El eje X representa los valores del índice de desarrollo humano para cada observación.

```
sns.swarmplot(y=df['region_recod'],
x=df['undp_hdi']).set_title('Distribución del IDH por zona geográfica');
```

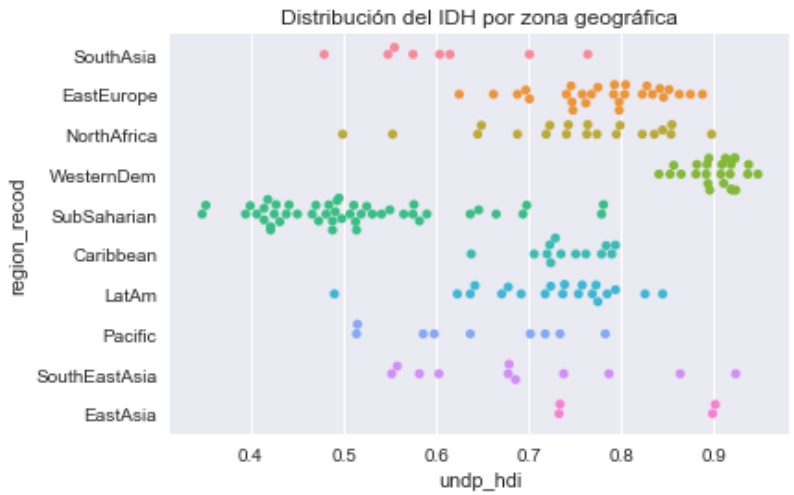

Imagen 6. Distribución de IDH por zona geográfica.
Fuente: Desafío Latam.

Resulta que para `swarmplot` y otros métodos de `seaborn`, el orden entre `x` e `y` depende de cómo deseamos presentar la información. Intercambiamos el orden y agregamos `hue=df['democracies']` para identificar si cada observación es democracia o dictadura.

```
plt.xticks(rotation = 45)
sns.swarmplot(x=df['region_recod'], y=df['undp_hdi'], hue =
df['democracies']).set_title(
'Distribución del IDH por región y tipo de gobierno');
```

Imagen 7. Distribución del IDH por región y tipo de gobierno.
Fuente: Desafío Latam.

FacetGrid

`FacetGrid` se utiliza para graficar múltiples figuras que comparten ejes, y cada figura está condicionada por un valor en específico. Técnicamente, `FacetGrid` es una clase que genera un objeto que permite asociar un DataFrame de `pandas` con una estructura particular de una figura de `matplotlib`.

La idea general de `FacetGrid` es generar una forma canónica de gráficos que se repita en múltiples espacios reducidos.

El flujo de trabajo básico con `FacetGrid` es:

1. Iniciar un objeto FacetGrid declarando el DataFrame y las variables para estructurar la grilla.
2. Aplicar una o más funciones de gráficos mediante los métodos .map() o .map_dataframe() del objeto FacetGrid.

Iniciar un objeto FacetGrid

`seaborn.FacetGrid` necesita de 3 argumentos como mínimo: un objeto `DataFrame`, una columna del `DataFrame` que informe al objeto sobre la cantidad de cuadros a desarrollar y un argumento `col_wrap` que define la cantidad de columnas.

Tomemos el siguiente ejemplo: Deseamos dividir nuestro espacio a lo largo de todas las categorías de la variable `df['gol_inst']`. El resultado que nos indica es 4 valores únicos.

```
df['gol_inst'].value_counts()
```

```
0.0    54
2.0    41
1.0    32
4.0     2
Name: gol_inst, dtype: int64
```

Si deseamos generar una serie de gráficos de dimensiones 2 x 2, podemos hacer lo siguiente:

1. El primer parámetro es el nombre de la tabla.
2. El segundo parámetro es el nombre de la columna para utilizar como referencia.
3. El tercer parámetro es el dimensionado.

```
grid = sns.FacetGrid(df, col="gol_inst", col_wrap = 2)
```

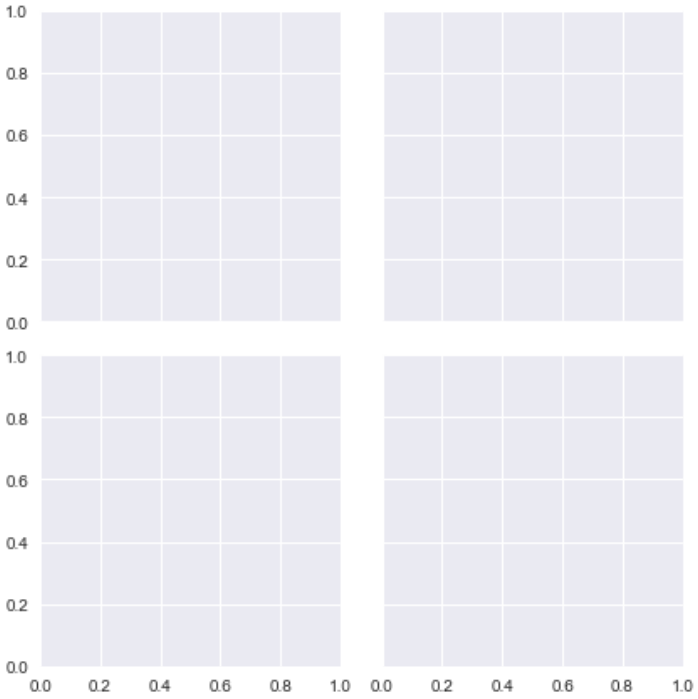

Imagen 8. Gráficos de dimensión 2x2.
Fuente: Desafío Latam.

Aplicar gráficos a nuestro objeto

Una vez iniciado, nuestro objeto tendrá la opción `.map` para aplicar una función a cada cuadrante del área del gráfico.

- El primer argumento de `map` debe ser la función. Ésta puede ser de `matplotlib`, `seaborn` u otra librería. La llamada sólo debe incluir el nombre de la función. No hay que pasar los parámetros.

- El segundo argumento de `map` es la variable a graficar. En este ejemplo vamos a utilizar `distplot`, por lo que sólo necesitamos una columna. En este caso será `undp_hdi`.

- Posterior a los dos argumentos obligatorios, se pueden incorporar argumentos que modifican:

```
sns.set(font_scale=0.8) # Escalamiento de los títulos para que no sean
tan grandes

grid = sns.FacetGrid(df, col="gol_inst", col_wrap=2)

axes = grid.axes.flatten() # Obtener los ejes de ploteo para poder darle
```

```
título a cada grafico
axes[0].set_title('gol_inst = 0.0')
axes[1].set_title('gol_inst = 1.0')
axes[2].set_title('gol_inst = 2.0')
axes[3].set_title('gol_inst = 4.0')

grid = grid.map(sns.distplot, "undp_hdi")
```

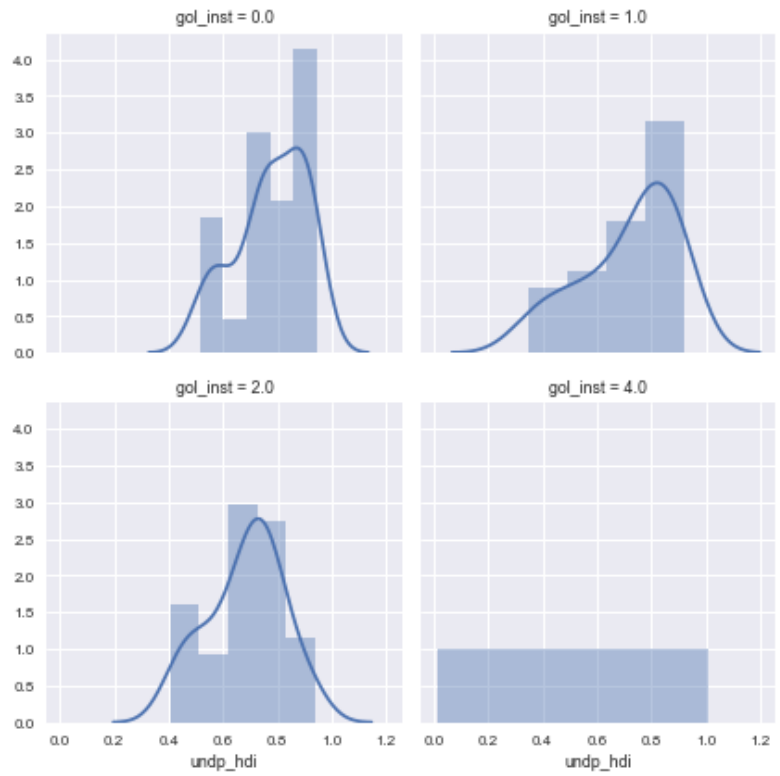

Imagen 9. Resultado de objetos en gráficos.
Fuente: Desafío Latam.

Al visualizar los histogramas, observamos que las distribuciones del índice de desarrollo humano tienden a ser similares entre los regímenes parlamentarios (`gol_inst=0.0`), semi-presidenciales (`gol_inst=1.0`) y presidenciales (`gol_inst=2.0`).
Se observa que los regímenes presidenciales tienden a presentar niveles de desarrollo más bajos, en comparación a los regímenes parlamentarios y semi parlamentarios.

Con respecto al error, si investigamos un poco, encontraremos que el error con el que nos topamos se produce cuando un método se encuentra con un argumento de largo nulo. El error se produce específicamente al graficar la distribución del Índice de Desarrollo Humano para las dictaduras militares (`gol_inst=4.0`).
Analicemos los datos que debían ser graficados:

```
print('Cantidad de dictaduras militares registradas: %d'
%len(df.loc[df['gol_inst'] == 4.0]))
print('Cantidad de NaN en la columna \'undp_hdi\' para las dictaduras
militares: %d'
      %df.loc[df['gol_inst'] == 4.0]['undp_hdi'].isnull().sum())
```

```
Cantidad de dictaduras militares registradas: 2
Cantidad de NaN en la columna 'undp_hdi' para las dictaduras militares:
1
```

Existen tan solo dos registros de dictaduras militares en todo el dataset de los cuales uno tiene `NaN` en la columna `undp_hdi`, por lo tanto, era de esperarse que el método `sns.distplot()` no pudiese graficarlo.

(**Nota:** La razón no es porque haya un `NaN` en los registros, sino porque al final le estábamos pidiendo que graficase un solo valor en lugar de una serie de valores, que es lo que el método espera recibir como input. El método `sns.distplot()` por defecto ignora los valores NaN)

La baja cantidad de dictaduras militares (`gol_inst=4.0`) nos impide sacar cualquier conclusión general válida con respecto al Índice de Desarrollo Humano de este tipo de gobiernos.

¡Reforcemos lo aprendido!

A partir de lo que has revisado respecto al análisis de datos ¿De qué manera hemos podido mejorar nuestros gráficos mediante el uso de seaborn? Construye un pequeño mapa conceptual que te permita visualizar de qué manera has podido ir complejizando las figuras a partir de lo que has aprendido hasta ahora.

Scatterplots

Un scatterplot (o diagrama de dispersión) es una visualización que presenta observaciones de una base de datos mediante coordenadas cartesianas para dos variables.

Cada punto en la colección de puntos visualizada en el plano cartesiano.

Algunas formalidades:

- Eje X, se posiciona en la línea horizontal. Se le conoce como Eje de Abscisas y corresponde a la variable independiente en el contexto de regresión.
- Eje Y, se posiciona en la línea vertical. Se le conoce como Eje de Ordenadas y corresponde a la variable dependiente en el contexto de regresión.
- El siguiente código es la implementación de un diagrama de dispersión utilizando `matplotlib`. El método `plt.scatter` necesita de dos argumentos obligatorios, nuestro eje X e Y.
- En el eje X vamos a utilizar nuestra variable *Índice de Desarrollo Humano*, y en el eje Y utilizaremos el indicador de la Calidad de Gobierno. El objetivo es evaluar la asociación existente entre ambas mediciones. Valores mayores en el Indicador de la Calidad de Gobierno indican mejor calidad.

```
# generamos el gráfico
plt.scatter(x=df['undp_hdi'], y=df['icrg_qog'])
plt.title('Calidad de Gobierno en función del IDH')
plt.xlabel("Indice de Desarrollo Humano")
plt.ylabel("Calidad del Gobierno");
```

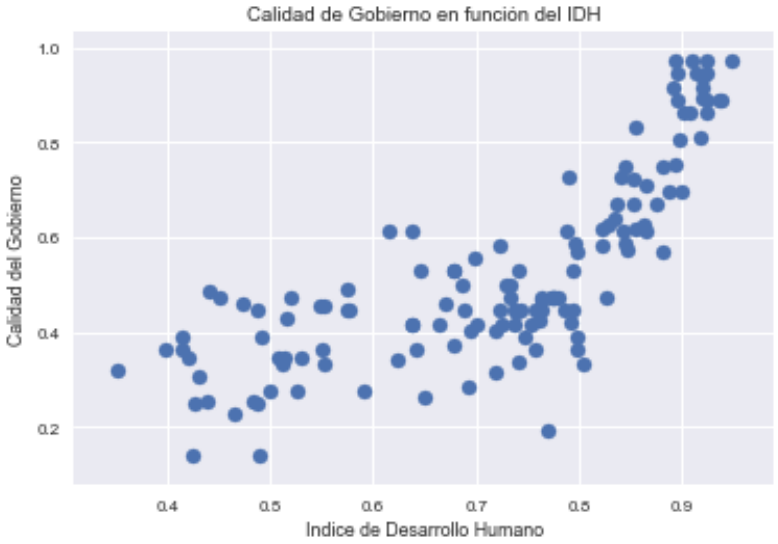

Imagen 10. Calidad de Gobierno en función del IDH.
Fuente: Desafío Latam.

A simple vista, el gráfico enseña que en la medida que el IDH va aumentando en la escala, la calidad del gobierno mejora de igual manera. Esto nos servirá para hablar de medidas de asociación.

Refactorizando el gráfico

Por consiguiente, el gráfico se puede mejorar de forma sustancial al incluir una recta que resuma la tendencia e identifique los países.

Para generar una recta utilizando sólo `matplotlib` necesitamos estimar un intercepto y pendiente. Ésto se puede lograr de variadas formas, pero para evitar hablar de regresión, utilizaremos el método `np.polyfit` de `numpy`. El método requiere especificar los ejes X e Y de la relación bivariada y la cantidad de términos a estimar. Dado que buscamos una recta lineal, sólo necesitamos un término.

```
# Separemos las columnas a trabajar y eliminemos los datos perdidos.
scatter_data = df.loc[:, ['undp_hdi', 'icrg_qog', 'ccodealp']].dropna()
# calculamos los valores de la recta
pendiente, intercepto = np.polyfit(scatter_data['undp_hdi'],
scatter_data['icrg_qog'], 1)

# pidamos los valores
print("La pendiente es de: ", pendiente.round(3))
print("El intercepto es de: ", intercepto.round(3))
```

```
La pendiente es de:  1.008
El intercepto es de:  -0.195
```

Para graficar la recta, necesitamos calcular los valores para cada nivel del índice de desarrollo humano. Esto lo podemos hacer mediante una list comprehension.

Comprensiones de Lista

- Las *list comprehension* forman parte del python idiomático. Son un híbrido entre un loop `for` y una `lista`. El fin es generar nuevas listas de una forma concisa y elegante.
- Una list comprehension general tiene una estructura canónica:

```
[<expresion_a_evaluar> <for <variable> in <secuencia_de_valores>>]
```

- Esto nos ahorra tiempo y líneas en estar generando objetos para guardar los resultados de un loop.

```
# para separar los elementos del gráfico, generamos dos objetos a partir de subplots
fig, ax = plt.subplots()

# generamos el gráfico, declaramos que los puntos sean lo más pequeños posible con marker="," y s=.1
ax.scatter(x=scatter_data['undp_hdi'], y=scatter_data['icrg_qog'], marker=",", s=.1)

# graficamos la recta a lo largo de undp_hdi
ax.plot(scatter_data['undp_hdi'],
         # generamos un list comprehension que calcule el valor de la recta a lo largo de undp_hdi
        [pendiente * j + intercepto for j in scatter_data['undp_hdi']],
        color='tomato')
ax.set_title('Tendencia de la Calidad de gobierno dependiendo del IDH')
ax.set_xlabel("Indice de Desarrollo Humano")
ax.set_ylabel("Calidad del Gobierno");
```

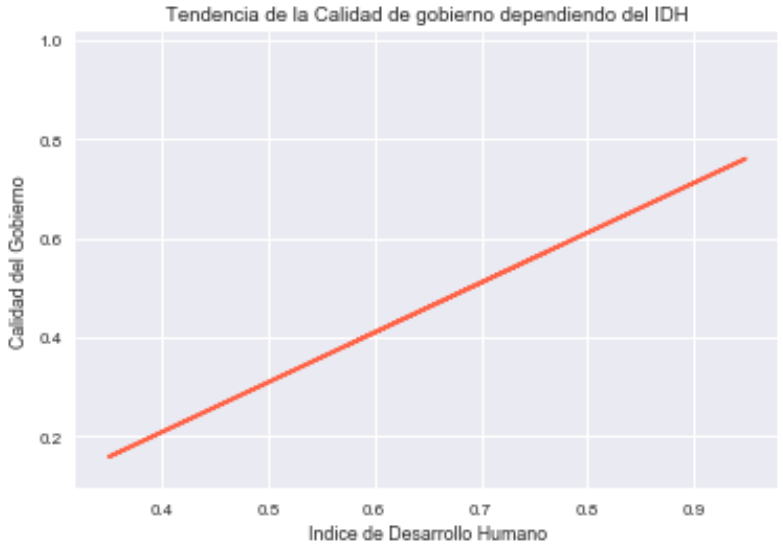

Imagen 11. Tendencia de la calidad de gobierno dependiendo del IDH.
Fuente: Desafío Latam.

Hasta ahora nuestro gráfico tiene la recta, pero no tenemos las observaciones etiquetadas. Vamos a generar las anotaciones mediante el método `text`, el cual dependerá del objeto `ax` que creamos con `subplots`.

Para este caso utilizamos un loop `for` clásico porque no necesitamos guardar los valores en una lista, si no el repetir la instrucción `text` por cada observación en nuestro dataframe.

La sintaxis es la siguiente:

```
for i in scatter_data.index:
    ax.text(
        scatter_data.loc[i, 'undp_hdi'],
        scatter_data.loc[i, 'icrg_qog'],
        str(scatter_data.loc[i, 'ccodealp'])
        )
```

Utilizamos `.loc` para acceder a las observaciones específicas, y solicitamos al loop que recorra por el índice del DataFrame. Al juntar todo el código, queda lo siguiente:

```
fig, ax = plt.subplots()
ax.scatter(x=scatter_data['undp_hdi'], y=scatter_data['icrg_qog'],
marker=",", s=.1)

ax.plot(scatter_data['undp_hdi'],
        [pendiente * j + intercepto for j in scatter_data['undp_hdi']],
```

```
        color='tomato')


for i in scatter_data.index:
    ax.text(scatter_data.loc[i, 'undp_hdi'],
            scatter_data.loc[i, 'icrg_qog'],
            str(scatter_data.loc[i, 'ccodealp']),
           fontsize = 5.5)

ax.set_title('Calidad del Gobierno e IDH por país')
ax.set_xlabel("Indice de Desarrollo Humano")
ax.set_ylabel("Calidad del Gobierno");
```

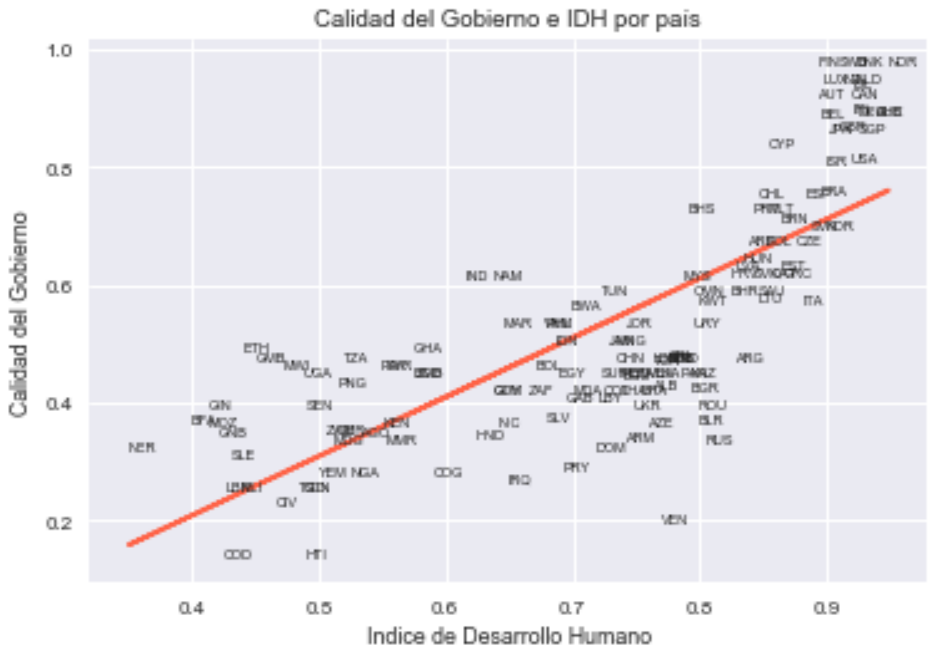

Imagen 12. Calidad del Gobierno e IDH por país.
Fuente: Desafío Latam.

El gráfico nos informa de la posición de países como Haití, con un nivel bajo de desarrollo humano y de calidad de gobierno, así como de un cluster importante de países con niveles de desarrollo humano sobre el .90 y calidad de gobierno sustancialmente alto.

Refactorización con seaborn

Volviendo a la librería `seaborn`, resulta que presenta un método llamado `jointplot` que permite realizar una diagrama de dispersión con más información.

De manera similar, al ejemplo con `matplotlib`, necesitamos declarar nuestros ejes X e Y.

```
sns.jointplot(df['undp_hdi'], df['icrg_qog']);
```

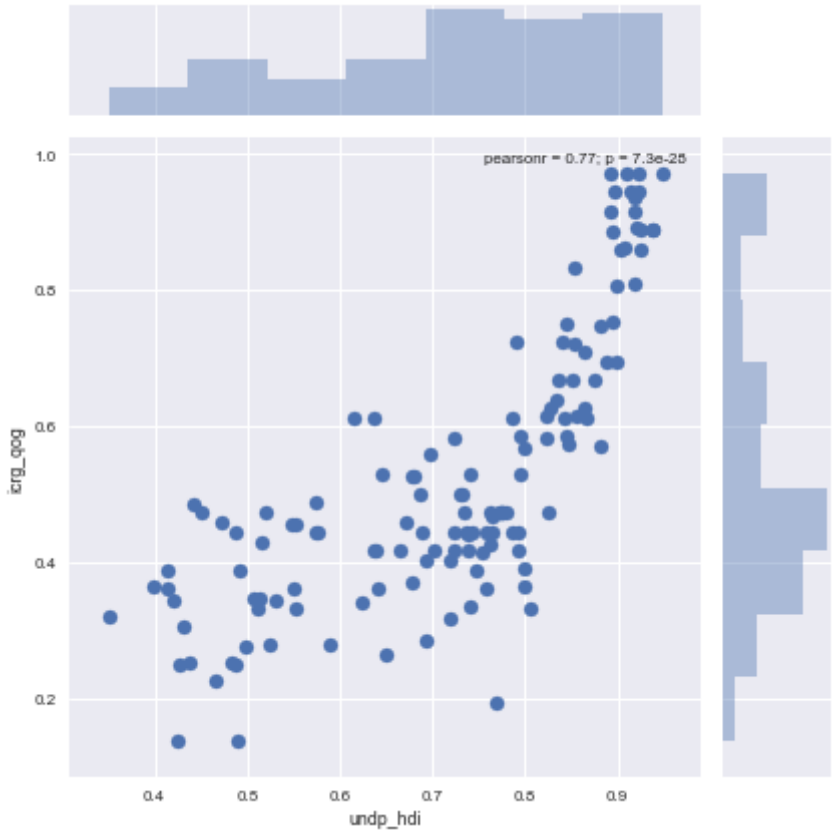

Imagen 13. Refactorización con *seaborn*.
Fuente: Desafío Latam.

Aparte de la nube de puntos visualizada, la figura incluye información sobre la distribución de cada variable resumida en forma de histogramas.

Para agregar una recta de ajuste, simplemente agregamos el argumento `kind = 'reg'` a `jointplot`.

```
sns.jointplot(scatter_data['undp_hdi'], scatter_data['icrg_qog'],
```

```
kind='reg');
```

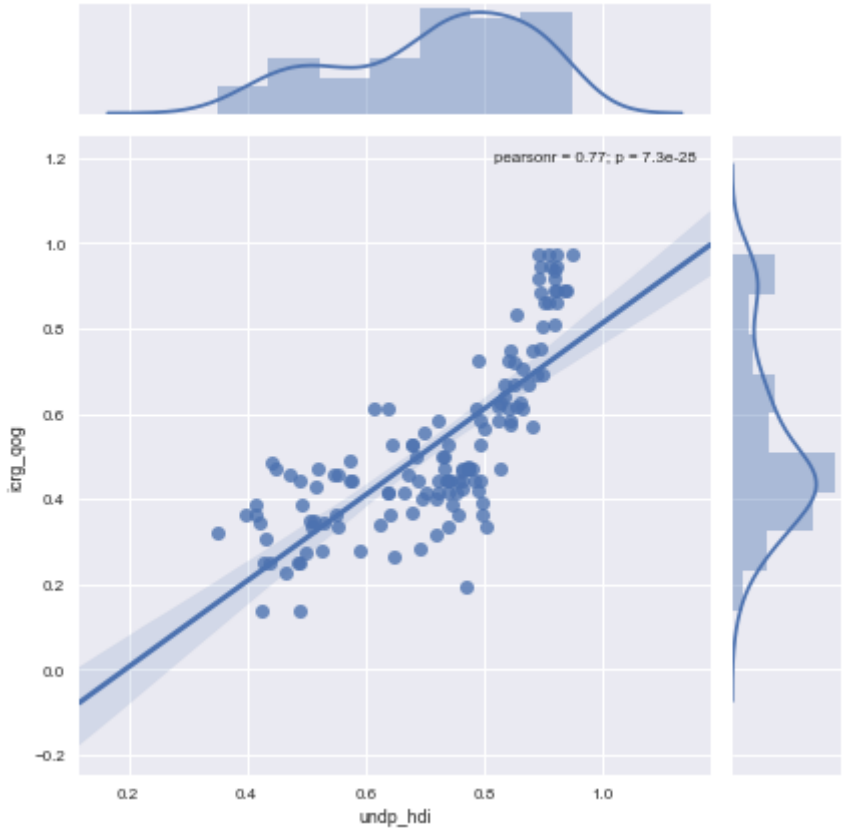

Imagen 14. Resultado de la recta.
Fuente: Desafío Latam.

Se reporta un estadístico `pearsonr = 0.77; p = 1.8e-25`. La primera cifra resume la intensidad y dirección de la asociación, mientras que la segunda reporta su *plausibilidad* bajo condiciones similares.

El primer elemento se conoce como *Correlación de Pearson*, y el segundo como *p Value*. Dedicaremos el resto de la lectura a estudiarlos.

Para terminar, veremos un tipo de gráfico que nos permitirá identificar características asociadas a la dispersión de los valores de una variable y valores atípicos en esa variable. Estamos hablando de **Boxplots** (gráficos de caja). Antes de aprender a construir boxplots, sin embargo, debemos aprender a obtener ciertos estadísticos que son representados en este tipo de gráficos.

Consideremos una muestra de tamaño 9. de valores medidos para las alturas de personas:

$$X = \{1.76,\ 1.63,\ 1.82,\ 1.69,\ 1.97,\ 1.77,\ 1.65,\ 1.92,\ 1.85\}$$

Consideremos ahora esta muestra ordenada de menor a mayor:

$$X = \{1.63,\ 1.65,\ 1.69,\ 1.76,\ 1.77,\ 1.82,\ 1.85,\ 1.92,\ 1.97\}$$

Si tomamos el elemento "central" de esta serie, es decir, aquél que ocupa la 5ta posición (el 1.77), obtendremos dos conjuntos de igual tamaño a ambos extremos. A este elemento central (luego de haber ordenado la muestra en orden creciente) se le conoce bajo el nombre de **Mediana** y es un estadístico de tendencia, al igual que la media. Sin embargo, la mediana tiene propiedades interesantes, por ejemplo, notar que en nuestra muestra el valor de la mediana no cambia si reemplazamos el valor más alto (1.97) por uno incluso más alto cómo 2.6. Si nos fijamos en la media de esta muestra veremos, no obstante, que esta pasa de tener un valor de 1.78 en los datos originales, a tener un valor de 1.85. A esta propiedad de "resistir" la magnitud de ciertos datos en la muestra que puedan ser extremos, se le conoce como "***robustez***".

La mediana es, entonces, un estadístico de tendencia robusto porque nos entrega información sobre la tendencia de una variable hacia ciertos valores del rango en el que está definida y al mismo tiempo es capaz de resistir la influencia de la magnitud de algunos elementos extremos en la muestra.

Consideremos ahora los dos conjuntos extremos que resultaron

$(X_{inferior} = \{1.63,\ 1.65,\ 1.69,\ 1.77\}$ y $X_{superior} = \{1.82,\ 1.85,\ 1.92,\ 1.97\})$

Si repetimos el proceso, nos encontraremos con que no existe un valor central que divida las submuestras en partes iguales ya que la cantidad de elementos en cada submuestra es par, en estos casos se considera la media de los dos elementos centrales de la muestra, estos serían 1.67 y 1.88 respectivamente para cada submuestra. Con lo anterior, tendríamos dividida nuestra muestra en cuatro secciones de igual cantidad de elementos mediante tres valores (1.67, 1.77 y 1.88), a estos tres valores se les denominan **cuartiles** 1, 2 (o mediana) y 3 respectivamente.

Los cuartiles, al igual que la mediana, nos entregan información sobre la distribución de los datos y pueden ser fácilmente interpretados si recordamos que, al estar haciendo las divisiones con respecto a la cantidad de datos y no a sus magnitudes, entonces la mediana representa el límite entre el 50% inferior de la muestra y el 50% superior.

En nuestro ejemplo, ya que la mediana es 1.77 podemos decir lo siguiente: "El 50% de las personas de la muestra miden 1.77 o menos", de forma análoga podemos decir lo mismo para el 50% superior.

Esta interpretación mencionada no es exclusiva de la mediana y puede ser fácilmente generalizada a los cuartiles 1 y 3:

- **Cuartil 1:** El 25% de la muestra mide 1.57 o menos y de forma análoga, el 75% de la muestra (todo el restante superior) toma valores de 1.57 o más.
- **Cuartil 3:** El 75% de la muestra mide 1.88 o menos, mientras que el 25% de la muestra mide 1.88 o más.

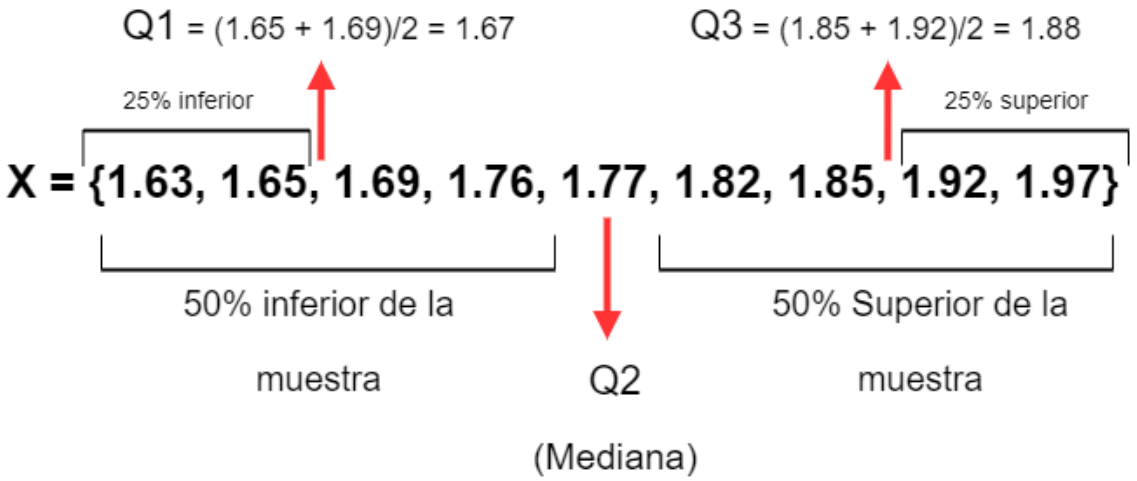

Imagen 15. Cuartiles 1 y 3.
Fuente: Desafío Latam.

Podemos generalizar aún más esta idea recursiva de división de subsecciones de la muestras ordenada para llegar a dividir la muestras en 100 partes equidistantes en cantidad de elementos. En este caso, a esas divisiones (lo que antes llamábamos cuartiles) se les conoce con el nombre de **percentiles** y, como probablemente ya hayas intuido, su interpretación es análoga: Si la altura de esa persona se encuentra entre el percentil 95 y el 96, por ejemplo, esa persona mide más que, al menos, el 95% de las personas más bajas de la muestra.

Una forma en la que suele ser referenciado este concepto general de división de la muestra en *n* partes, es mediante el nombre **"cuantil"**. Hay que notar también que no hay una sola forma establecida de los cuantiles de una variable, aunque la mayoría de los métodos dan resultados similares.

Boxplot

Un boxplot es un gráfico que muestra los cuartiles de una variable por medio de una estructura de caja de la siguiente forma:

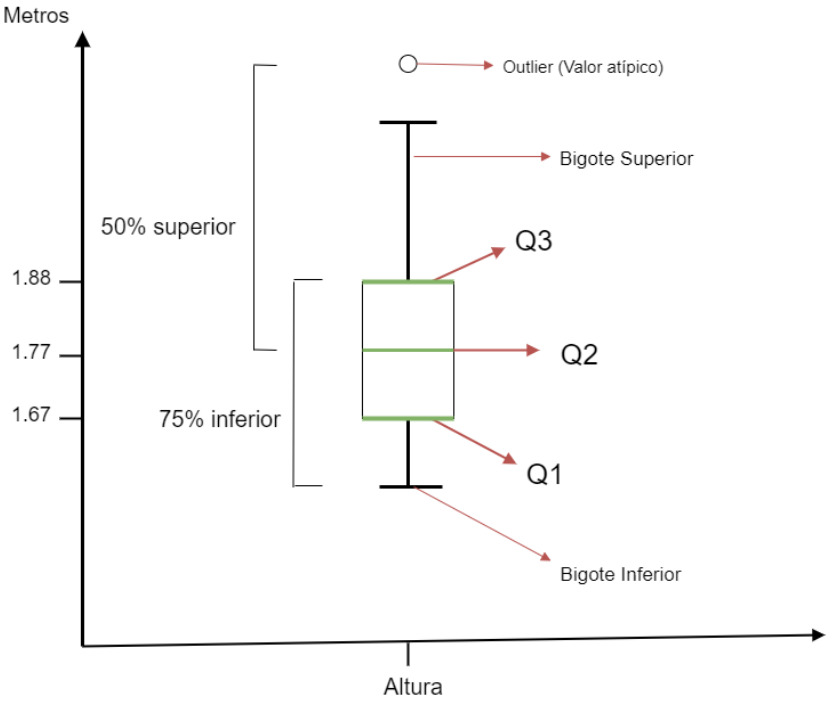

Imagen 16. Boxplot.
Fuente: Desafío Latam.

Un boxplot no solo grafica los cuartiles, sino que también muestra un "*rango de tolerancia*" para los valores que pueden tomar los elementos de la muestras antes de que se consideran valores atípicos (ya sea porque son demasiado altos o demasiado bajos), a estos rangos de tolerancia también se les llama "*bigotes*" del boxplot y cómo podrá el lector ver, no son necesariamente simétricos con respecto a la caja.
En la figura anterior se observa un solo valor atípico (en inglés denominados ***outliers***) en el extremo superior, esto corresponde a una persona con una altura considerablemente alta con respecto al resto de personas de la muestra.

Los bigotes siempre deben extenderse hasta un valor de la muestras y la forma de calcularlos es, primero, mediante el cálculo de la extensión máxima a la que estos pueden llegar:

$$Lim_{sup} = Q3 + 1.5 * IQR$$
$$Lim_{inf} = Q1 - 1.5 * IQR$$

Donde IQR es el denominado Rango Intercuartílico, una medida de dispersión calculada de la siguiente forma:

$$IQR = Q3 - Q1$$

Una vez calculados los límites, se debe observar el valor de la muestra más cercano al límite que no se extiende más allá del mismo. Una analogía que permite visualizar esto es imaginando que en primera instancia los bigotes son extendidos hasta sus límites y, después, estos son recogidos hasta quedar "*anclados*" en el primer valor de la muestra encontrado.

Para terminar con boxplot, es interesante mencionar el contraste que existe entre la varianza/desviación estándar con el IQR, mientras que en todos estos casos las medidas nos dan nociones sobre qué tan dispersos están los datos, el IQR al ser calculado a partir de los cuantiles es un poco más robusto que la varianza/desviación estándar que son calculados a partir de la media.

En matplotlib se puede hacer un boxplot utilizando la función `boxplot` para hacer este tipo de gráficos:

```
plt.boxplot(df.school.dropna())

plt.xticks([1], ['Años medios de escolaridad (school)'])

plt.title('Años medios de escolaridad en la población adulta', size =
16)

plt.ylabel('Años');
```

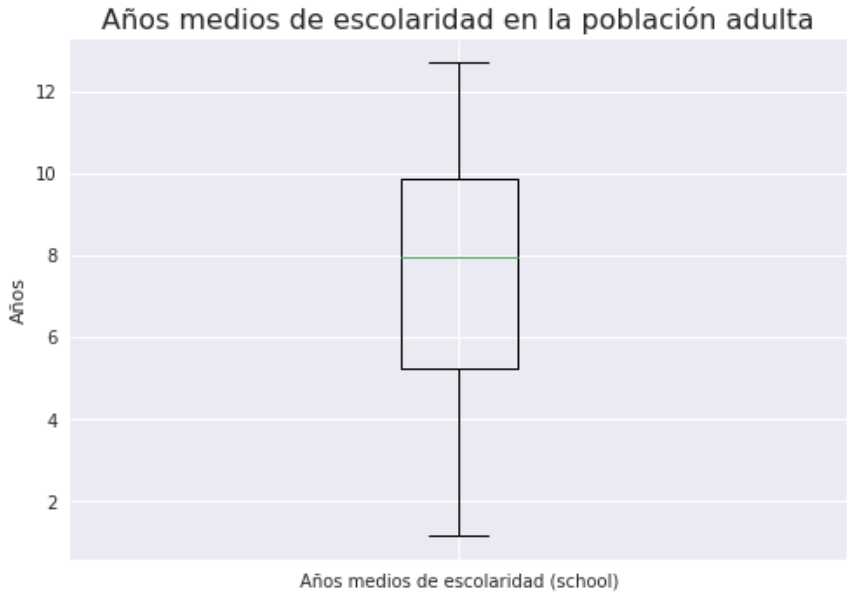

Imagen 17. Resultado boxplot..
Fuente: Desafío Latam.

En la librería seaborn, podemos hacer boxplot de la siguiente forma:

```
sn.boxplot(y = df.school.dropna());
plt.xticks([1], ['Años medios de escolaridad (school)']);
plt.title('Años medios de escolaridad en la población adulta', size =
16)
plt.ylabel('Años');
```

Imagen 18. Resultado boxplot 2.
Fuente: Desafío Latam.

Preguntas de cierre

- ¿Por qué seaborn resulta una herramienta importante de trabajo?
- ¿Para qué se utiliza facetgrid? Construye un ejemplo que explique su utilidad.
- Explique el flujo de trabajo de facetgrid
- ¿Qué es un scatterplots? Enumera sus características o de qué se compone.
- ¿Por qué debemos aprender respecto a Boxplot? Argumenta.