

# ✓ Estadística Descriptiva y Visualización de Datos

Python en el Ámbito Científico

## Sesión 2

Alan Badillo Salas ([alan@nomadacode.com](mailto:alan@nomadacode.com))

## ✓ Repaso - Problemas del ámbito científico y programación en Python

En la [Sesión 1](#) dimos la bienvenida al curso y los problemas del ámbito científico, detectando **tres puntos** importantes a considerar dentro de cualquier proyecto científico:

- **El fenómeno real** - No podemos hablar de un fenómeno real, si no logramos observar los datos que produce
- **El planteamiento del problema** - Si tuviéramos los datos, debemos decir cuál sería la pregunta a resolver a través de los datos
- **El modelo matemático/datos** - Si quisiéramos resolver una pregunta a través de los datos, debemos decir cuál sería el modelo matemático o modelo de datos a utilizar y cómo se interpretará

Estos **tres puntos** los iremos desarrollando durante el curso y serán fundamentales para hacer **Ciencia de Datos**, mediante técnicas de Machine Learning / Aprendizaje Automático (**ML**) o modelos de Deep Learning / Aprendizaje Profundo / Redes Neuronales (**AI**).

Pero primero comenzaremos por dominar algunos conceptos de programación en Python.

# Diferencias de entre la programación general y específica

Existen dos modos de programar en los lenguajes de programación:

- **Una programación general** - Permite desarrollar sistemas y aplicaciones de propósito general, como administrar negocios o transacciones diarias. Esto requiere habilidades de Diseño, Gestión y Metodologías de Proyectos Software.
- **Una programación específica** - Permite implementar algoritmos para resolver problemas específicos, por ejemplo, problemas de ingeniería o matemática. Esto requiere el uso del lenguaje, pero principalmente de librerías científicas como las librerías estadísticas.

Nosotros a partir de ahora, nos asumiremos como *Científicos de Datos* (aprendices o entusiastas).

## Tipos de Científicos de Datos

Dentro de la Ciencia de Datos hay muchas ramas y actividades a desarrollar, pero podemos encontrar *tres categorías* principales reconocidas por empresas:

- **Científico de Datos** - Crea análisis avanzados y domina los modelos de **ML/AI** teóricos, es capaz de determinar cuál es el modelo más adecuado a un problema y cómo ajustar sus valores.
- **Analista de Datos** - Crea análisis estandarizados y domina técnicas de **ML/AI** prácticas, es capaz de determinar el modelo más adecuado a un problema y experimentar con distintas configuraciones para lograr un ajuste ideal del modelo a los datos.
- **Ingeniero de Datos** - Implementa arquitecturas de bases de datos y programas de automatización de **ML/AI** principalmente en soluciones en la nube, es capaz de armar un lago de datos, dashboards estadísticos automatizados e integrar APIs y rutinas de *ML/AI* dentro de las aplicaciones operativas de una empresa.

Generalmente uno comienza por el análisis de datos y luego toma la decisión de su vida: Programar o Modelar.

En este curso, nos asumiremos como un poco de los *tres perfiles*, capaces de ser integrales en todos los ámbitos de la ciencia de datos.

## ¿Qué necesitamos saber sobre la programación en Python?

Aunque no lo crean, saber crear variables, importar librerías, usar funciones y generar reportes es casi suficiente, al menos que aspiremos a ser buenos ingenieros de software (digo de datos).

Si no nacimos en la cuna de los desarrolladores de software y queremos ver la ciencia de datos de un modo más matemático que computacional, la respuesta es simple, usa lo que hay, pero úsalo bien.

Así como todos aprenden a resolver un sistema de ecuaciones lineales para una o dos variables incógnitas, uno debería aprender un par de modelos matemáticos que te resolverán la vida en la ciencia de datos.

Poder implementar estos modelos en Python, no será tan difícil usando las librerías correctas.

Así que repasemos la programación que hemos aprendido y extendamos solo algunas cosas extra.

### ✓ El Tour de la Ciencia de Datos Paso a Paso

Veamos un modelo de ciencia de datos que será fundamental para entrenar nuestra programación en Python (uso de variables y generación de reportes).

El modelo es sencillo, busquemos la relación entre los datos de las estaturas de personas y su peso.

¿Qué relación hay?

Primero definamos una lista de valores en Python.

## ✓ Lista de valores

Para definir una lista de valores en Python usamos la sintaxis:

```
<mi lista> = [<elemento 1>, <elemento 2>, ..., <elemento k> ]
```

Donde `<mi lista>` es el nombre de la lista (piensa en un vector o un eje de datos), y `<elemento 1>, ..., <elemento k>` son los valores de cada elemento en la lista.

**Pregunta: ¿Cómo se relacionan los elementos de la lista con las muestras de un experimento?**

Veamos un ejemplo, definamos las listas de estaturas y pesos de una persona, piensa en los valores más comunes o haz una encuesta ficticia:

```
estaturas = [1.71, 1.64, 1.73, 1.62, 1.81, 1.70, 1.54, 1.56, 1.67, 1.68]  
pesos = [76.8, 67.8, 84.5, 69.1, 98.5, 85.6, 65.4, 59.3, 67.1, 74.5]
```

Al definir las listas de valores, hemos almacenados un montón de valores en una sola variable (cada lista en una variable).

¿Cómo podemos imprimir los valores como tabla cuando haga falta?

Python tiene la capacidad de unir y separar listas de valores en una sola lista si sus dimensiones son iguales.

Esto nos crea un flujo de elementos agrupados, por ejemplo:

```
<x> = [<x1>, <x2>, ..., <xk> ]  
<y> = [<y1>, <y2>, ..., <yk> ]  
  
<flujo> = zip(<x>, <y>)
```

donde `zip(<lista 1>, <lista 2>, ...)`, agrupa en un mismo flujo los valores de cada lista secuencialmente, dentro de algo llamado tupla (o paquete).

Esto significa que `<flujo>` tendrá los elementos agrupados de la forma:

```
<flujo> ~ [ (<x1>, <y1>), (<x2>, <y2>), ..., (<xk>, <yk>) ]
```

Si queremos imprimir al mismo tiempo cuánto vale  $(x_i, y_i)$  bastará con recorrer cada elemento del flujo, esto lo podemos hacer con un **for**

```
for <xi>, <yi> in <flujo>:  
    # Aquí podemos usar el valor de <xi>, <yi>  
    print(f"{<xi>} {<yi>}")
```

Veamos un ejemplo

```
flujo = zip(estaturas, pesos)
```

```
for estatura, peso in flujo:  
    print(f"{estatura} {peso}")
```

```
➡ 1.71 76.8  
   1.64 67.8  
   1.73 84.5  
   1.62 69.1  
   1.81 98.5  
   1.7 85.6  
   1.54 65.4  
   1.56 59.3  
   1.67 67.1  
   1.68 74.5
```

Con nuestro conocimientos sobre impresión y reportes podemos ajustar el reporte para que se vea más estructurado

```
flujo = zip(estaturas, pesos)
```

```
print("Estatura   |   Peso")  
print("-" * 20)  
for estatura, peso in flujo:  
    print(f"{estatura:^10.2f} | {peso:^8.1f}")
```

```
➡ Estatura   |   Peso  
-----  
    1.71     |   76.8  
    1.64     |   67.8  
    1.73     |   84.5  
    1.62     |   69.1  
    1.81     |   98.5  
    1.70     |   85.6  
    1.54     |   65.4  
    1.56     |   59.3  
    1.67     |   67.1  
    1.68     |   74.5
```

Recapitemos lo que ocurre, las listas y tuplas son el núcleo para empaquetar y mover datos naturales en Python, esto evita usar matrices (arreglos o tablas) más complejas.

Por lo que entender las listas, tuplas y diccionarios a marchas forzadas será sumamente necesario para dominar operaciones estadísticas básicas con Python.

## Ejercicio de clase: Reportar tres ejes de datos

1. Define una lista de géneros (hombre y mujer) con al menos 6 datos
2. Define una lista de edades con el mismo tamaño que la lista de géneros
3. Define una lista de probabilidad de tener una tableta con valores entre  $[0, 1]$  para cada edad y género.
4. Crea un flujo para unir y recorrer las tres listas al mismo tiempo
5. Recorre los elementos del flujo con un **for**, desacoplando los tres elementos unidos como `genero`, `edad`, `probTableta` e imprime los tres valores en cada iteración
6. Mejora el reporte para mostrar cabeceras (*headers*) y alineaciones, considera que la probabilidad esté en formato de porcentaje (multiplicado por 100) y con el símbolo %.

# Tu código aquí

### ▼ Tuplas de datos

En Python, los valores de una variable pueden ser coleccionados en una lista, lo que significa ir agregando o quitando elementos con el tiempo, o acoplarlos en un tupla, que significa crear un vector o paquete con los datos congelados.

Veamos la diferencia:

En una lista vacía podemos ir agregando elementos para después utilizarlos mediante el método `<mi lista>.append(<valor>)`. Esto agregará el nuevo valor al final de la lista (inicialmente vacía)

```
calificaciones = [] # []

calificaciones.append(8.5) # [8.5]
calificaciones.append(9.2) # [8.5, 9.2]
calificaciones.append(7.8) # [8.5, 9.2, 7.8]
```

Así podríamos en algún momento reportar las calificaciones o sacar algunos estadísticos

```
for calif in calificaciones:
    print(f"La calificación es: {calif:.1f}")

print(f"La suma de calificaciones es: {suma(calificaciones):.2f}")
```

Esto es útil cuando los datos evolucionan en el tiempo, o se tienen secuencias de datos repetidos en un eje, pero qué pasa si queremos retener en una misma variable los valores de una muestra, por ejemplo, el género, edad y probabilidad de tener una tableta para una persona, entonces usaremos una *tupla* para acoplar dichos valores, como son tres valores los que queremos acoplar, la llamaremos *3-tupla*.

```
muestra1 = "Hombre", 23, 0.85
```

también podemos usar los paréntesis para ser más explícitos `muestra1 = ("Hombre", 23, 0.85)`.

A este proceso de unir valores por *comas* se le denomina acoplamiento o construcción de una tupla.

La tupla puede ser manipulada como una lista, pero está pensada para organizar datos de distinto tipo o naturaleza en un mismo elemento, por ejemplo, para determinar puntos, muestras o experimentos.

Su principal utilidad se da cuando desacoplamos los valores y recuperamos los datos. Por ejemplo, si generamos una lista con tuplas, podemos acceder a algún elemento mediante su índice y recuperar la tupla, luego esa tupla la podemos desacoplar y extraer sus valores. Para una *k-tupla* podemos desacoplar exactamente (no más ni menos) *k-variables*.

```
muestras = [  
    ("Hombre", 23, 0.85),  
    ("Mujer", 18, 0.51),  
    ("Hombre", 15, 0.40),  
    ("Mujer", 65, 0.94),  
    ("Mujer", 34, 0.67),  
    ("Hombre", 36, 0.93),  
]  
  
muestra1 = muestras[0]  
  
genero1, edad1, probTableta1 = muestra1  
  
print(f"{genero1} con {edad1} años tiene una \  
prob. de tener tableta del {100 * probTableta1:.1f}%")
```

Observa que los índices en Python siempre empiezan en 0, lo cual es un dolor de cabeza, pero



nos tendremos que acostumbrar.

Lo mejor es no pensar en índices, sino más bien intentar procesar todos los datos a la vez.

Existe una fuerte relación entre las tuplas y listas, una lista puede ser convertida a tupla mediante `tuple(<mi lista>)` y al revés, una tupla puede ser transformada a lista mediante `list(<mi tupla>)`. Además la función `<flujo> = zip(<lista 1>, <lista 2>, ...)` combinará listas en una sola lista de tuplas (el flujo), o un flujo (una lista de tuplas) puede ser separado en listas individuales (proceso inverso) mediante `<lista 1>, <lista 2>, ... = zip(*<flujo>)`.

Todo esto significa que cuando construyamos una lista de tuplas, le llamaremos el flujo de datos y podremos recorrerlo mediante un **for**.

Veamos un ejemplo más avanzado, para reportar el flujo de datos sobre la probabilidad de tener una tableta

```

muestras = [
    ("Hombre", 23, 0.85),
    ("Mujer", 18, 0.51),
    ("Hombre", 15, 0.40),
    ("Mujer", 65, 0.94),
    ("Mujer", 34, 0.67),
    ("Hombre", 36, 0.93),
]

print(" Género | Edad | Prob. Tableta (%) ")
print("-" * 39)
for genero, edad, probTableta in muestras:
    print(f"{genero:^8} | {edad:^6} | {100 * probTableta:^17.1f}")
print("-" * 39)

generos, edades, probsTableta = zip(*muestras)

print(f"{'-':^8} | {sum(edades) / len(edades):^6.1f} | \
{100 * sum(probsTableta) / len(probsTableta):^17.1f}")

```

```

⇌ Género | Edad | Prob. Tableta (%)
-----
Hombre | 23 | 85.0
Mujer | 18 | 51.0
Hombre | 15 | 40.0
Mujer | 65 | 94.0
Mujer | 34 | 67.0
Hombre | 36 | 93.0
-----
- | 31.8 | 71.7

```

### Ejercicio de clase: Reporte de puntos

1. Define una lista de tuplas (flujo de datos) que contenga valores  $(x, y)$
2. Reporta cada punto  $(x_i, y_i)$  del flujo de datos
3. Separa la lista de puntos  $X$  y puntos  $Y$
4. Reporta el punto mínimo de cada lista
5. Reporta el punto promedio de cada lista
6. Reporta el punto máximo de cada lista

# Tu código aquí

Hasta ahora en este **Tour de la Ciencia de Datos Paso a Paso** hemos visto cómo construir un flujo de datos, a través de listas separadas y mediante una lista de tuplas, también hemos visto como separar el flujo de datos en listas independientes y generar reportes de interés.

Ahora es momento que empecemos con los temas de la sesión sobre la estadística descriptiva y la visualización de datos.

## ✓ Estadística descriptiva aplicada a problemas del ámbito científico

Ya hemos hablado sobre que los fenómenos reales y naturales producen datos, los datos resuelven preguntas acerca del fenómeno (planteamiento del problema) y para extraer esas respuestas construimos modelos (matemáticos).

Una de las ramas matemáticas más potentes para describir a los datos es la estadística, la cual piensa a los datos como variables aleatorias que se relacionan con errores de observación y forman un universo de distribuciones de probabilidad.

¿Qué es una variable aleatoria?

Desde un punto de vista aplicado, la aleatoriedad es todo proceso oculto que tiene un efecto en la realidad y es difícil determinarlo. Entonces, podemos pensar que una variable aleatoria es una variable que observamos a través de los datos, pero que no podemos saber cuál será exactamente el siguiente valor que observaremos de esa variable.

Por ejemplo, si salimos a la calle a preguntarle la edad a las personas que transitan y las registramos, estaremos observando los valores de una variable aleatoria, supongamos que son:

```
45 52 29 32 16 63 9 19 18 21 ...
```

¿Cuál será la siguiente edad observada?

Ni idea, es la respuesta corta. Pero dentro de estas variables aleatorias se esconden patrones y procesos interesantes que la estadística nos ayudará a detectar.

Existen *tres grandes formas* de describir una variable aleatoria que desarrollaremos esta clase:

- **Distribución de los datos** - Cuando observamos una variable aleatoria, podemos pensar que los datos son puramente aleatorios y nunca sabremos qué dato observaremos la próxima vez, pero si observamos la forma en la que se distribuyen los datos, quizás

entendamos donde están más concentrados los valores observados, por ejemplo, podría ser que los valores se aglutinen al rededor de un valor central y sea poco probable encontrar un valor muy lejano a este.

- **Media y Varianza** - El valor medio de una variable aleatoria nos permite formar una estadística interesante de los datos, por ejemplo, si la edad promedio de las personas que transitan fuera del metro politécnico es de 21 años, mientras que la edad promedio de las personas que transitan fuera del metro instituto del petroleo es de 34 años, significaría que en promedio la gente fuera del politécnico es más joven que la que está fuera de instituto del petróleo. Si además observamos la variación de las edades, podemos comenzar a describir el comportamiento oculto (aleatorio) de la variable aleatoria.
- **Cuartiles y Cuantiles** - Es posible que una variable aleatoria tenga miles de datos, uno para cada muestra, esto dificulta visualizar espacialmente donde se ubica, si podemos construir una caja con los estadísticos principales de la variable aleatoria, podemos determinar de forma rápida dónde se ubica su valor máximo y mínimo, los cuartiles correspondientes al 25%, 50% y 75% de los datos, además de los puntos atípicos que se encuentren.

Sin embargo, no todas las variables aleatorias se podrán describir de esta forma, todo dependerá del tipo de variable observada:

- **Variable Numérica** - Toma valores continuos o discretos del espacio o tiempo, podríamos encontrar en cualquier momento cualquier valor intermedio, más grande o más pequeño.
- **Variable Categórica** - Toma valores en un conjunto finito de estado, el valor es exactamente un estado, por ejemplo, *A, B, C, . . .*. Cuando solo hay dos estados, se llama variable *binaria*.

El análisis de variables numéricas es diferente al análisis de variables categóricas.

Veamos algunos tratamientos aplicados:

- **Variable numérica** - Determina un valor continuo y podemos determinar su distribución y estadísticos principales.
- **Variable Categórica** - Determina un valor en categorías y podemos determinar cuántos elementos hay de cada categoría y las probabilidades asociadas.
- **Análisis Mixto** - Dos variables continuas explican una correlación entre ambas, una variable categórica y una continua, explica las distribuciones y estadísticos principales de la variable continua, por cada categoría de la variable categórica.

Al análisis de variables continuas lo llamaremos *Análisis cuantitativo* y al análisis de variables categóricas lo llamaremos *Análisis cualitativo*.

Además los valores de una variable aleatoria pueden ser objetivo, si es que fueron extraídos mediante un proceso determinado como la medición de un instrumento calibrado o la respuesta registrada en un evento, pero será subjetivo si los valores de una variable aleatoria fueron dispuestos de forma directa por la creencia de una persona o resultado de una simulación.

Veamos algunos ejemplos:

1. La lectura de la temperatura de un biorreactor es variable aleatoria continua cuantitativa
2. Si el cliente se ve sospechoso por cómo viste es una variable aleatoria binaria cualitativa
3. El número de coca-colas vendidas la última hora es una variable aleatoria continua (de valores discretos) cuantitativa
4. El candidato a gobernador por el que votará una persona es una variable aleatoria categórica cuantitativa
5. El tipo persona entrevistada (alegre, triste, enojada, frustrada, violenta, apacible) es una variable aleatoria categórica cualitativa

Poder describir una variable aleatoria es fundamental para su tratamiento estadístico e información derivada. Además, siempre debemos tener cuidado con los datos cualitativos, ya que explicarán la realidad, basada en las creencias del intérprete que anotó el valor.

## ✓ Ejercicio S201: Describiendo variables aleatorias

Describe las siguientes variables aleatorias:

1. La edad de un pasajero del titanic
2. El género de un pasajero del titanic
3. La supervivencia de un pasajero del titanic
4. El número de hermanos o cónyuges de un pasajero del titanic
5. El número de padres o hijos de un pasajero del titanic
6. La clase en la que viajaba un pasajero del titanic
7. El costo del boleto (tarifa) de viaje de un pasajero del titanic
8. El puerto donde desembarcó un pasajero del titanic
9. El nombre de un pasajero del titanic
10. La cabina de un pasajero del titanic

Con esto, somos más sensibles al describir los datos, aún nos falta saber qué hacemos con datos faltantes, datos mal formados o datos corruptos, pero de momento, avanzaremos en analizar 3 problemas más relacionados al ámbito científico y su tratamiento estadístico.

## ✓ Problema 4 - Rendimiento de estudiantes

<https://hf-mirror.com/datasets/riyadahmadov/StudentPerformance>

Analicemos los datos para el rendimiento de alumnos en dos escuelas, Gabriel Pereira (GP) y Mousinho da Silveira (MS).

Este conjunto de datos contiene diferentes variables o características del estudiante, como su escuela, edad, género, hasta su nota final del grado (G3).

Comencemos por cargar el conjunto de datos (*dataset*) mediante *Pandas*.

## ✓ Adquisición del conjunto de datos

Hay muchas formas de adquirir un conjunto de datos, por ejemplo, de un CSV local o remoto (como en github), pero también de plataformas proveedoras de *datasets* como [HF-Mirror](#) que es una página China o [Kaggle](#) que es una página Estadounidense, ambas capaces de buscar y descargar conjuntos de datos compartidos y populares.

Nuestro trabajo posterior será descargar los conjuntos de datos y subirlos a un repositorio personal, pero mientras ocuparemos los que están disponibles en línea.

```
# Importa la librería de pandas, fundamental en ciencia de datos
import pandas

# Recupera el dataset de HF-Mirror
# https://hf-mirror.co/datasets/riyadahmadov/StudentPerformance/resolve/main/data
dataset = pandas.read_parquet("hf://datasets/riyadahmadov/StudentPerformance/data

# Muestra el dataset
dataset
```



|     | school | sex | age | address | famsize | Pstatus | Medu | Fedu | Mjob     | Fjob     | .. |
|-----|--------|-----|-----|---------|---------|---------|------|------|----------|----------|----|
| 0   | GP     | F   | 18  | U       | GT3     | A       | 4    | 4    | at_home  | teacher  | .  |
| 1   | GP     | F   | 17  | U       | GT3     | T       | 1    | 1    | at_home  | other    | .  |
| 2   | GP     | F   | 15  | U       | LE3     | T       | 1    | 1    | at_home  | other    | .  |
| 3   | GP     | F   | 15  | U       | GT3     | T       | 4    | 2    | health   | services | .  |
| 4   | GP     | F   | 16  | U       | GT3     | T       | 3    | 3    | other    | other    | .  |
| ... | ...    | ... | ... | ...     | ...     | ...     | ...  | ...  | ...      | ...      | .  |
| 644 | MS     | F   | 19  | R       | GT3     | T       | 2    | 3    | services | other    | .  |
| 645 | MS     | F   | 18  | U       | LE3     | T       | 3    | 1    | teacher  | services | .  |
| 646 | MS     | F   | 18  | U       | GT3     | T       | 1    | 1    | other    | other    | .  |
| 647 | MS     | M   | 17  | U       | LE3     | T       | 3    | 1    | services | services | .  |
| 648 | MS     | M   | 18  | R       | LE3     | T       | 3    | 2    | services | other    | .  |

649 rows x 33 columns

## ✓ Inspección del conjunto de datos

En lugar de mostrar todas las filas de un *dataset*, podemos mostrar solo las primeras `<n>` filas o por defecto 5 si no especificamos cuantas.

```
dataset.head(10)
```




|   | school | sex | age | address | famsize | Pstatus | Medu | Fedu | Mjob     | Fjob     | ... |
|---|--------|-----|-----|---------|---------|---------|------|------|----------|----------|-----|
| 0 | GP     | F   | 18  | U       | GT3     | A       | 4    | 4    | at_home  | teacher  | ... |
| 1 | GP     | F   | 17  | U       | GT3     | T       | 1    | 1    | at_home  | other    | ... |
| 2 | GP     | F   | 15  | U       | LE3     | T       | 1    | 1    | at_home  | other    | ... |
| 3 | GP     | F   | 15  | U       | GT3     | T       | 4    | 2    | health   | services | ... |
| 4 | GP     | F   | 16  | U       | GT3     | T       | 3    | 3    | other    | other    | ... |
| 5 | GP     | M   | 16  | U       | LE3     | T       | 4    | 3    | services | other    | ... |
| 6 | GP     | M   | 16  | U       | LE3     | T       | 2    | 2    | other    | other    | ... |
| 7 | GP     | F   | 17  | U       | GT3     | A       | 4    | 4    | other    | teacher  | ... |
| 8 | GP     | M   | 15  | U       | LE3     | A       | 3    | 2    | services | other    | ... |
| 9 | GP     | M   | 15  | U       | GT3     | T       | 3    | 4    | other    | other    | ... |

10 rows x 33 columns

Las primeras filas muestran la estructura principal de los datos, pero si estos están ordenados, difícilmente entenderemos que clase de valores tiene el *dataset*, por lo que podemos generar una muestra aleatoria con `k` ejemplo, por defecto 1 si no lo especificamos



```
dataset.sample(8)
```



|     | school | sex | age | address | famsize | Pstatus | Medu | Fedu | Mjob     | Fjob     | .. |
|-----|--------|-----|-----|---------|---------|---------|------|------|----------|----------|----|
| 464 | MS     | M   | 16  | R       | GT3     | T       | 1    | 1    | at_home  | other    |    |
| 643 | MS     | F   | 18  | R       | GT3     | T       | 4    | 4    | teacher  | at_home  |    |
| 122 | GP     | F   | 16  | U       | LE3     | T       | 2    | 4    | other    | health   |    |
| 258 | GP     | F   | 17  | U       | GT3     | T       | 3    | 2    | other    | other    |    |
| 45  | GP     | F   | 15  | U       | LE3     | A       | 4    | 3    | other    | other    |    |
| 502 | MS     | M   | 17  | R       | LE3     | T       | 2    | 2    | services | services |    |
| 632 | MS     | F   | 19  | R       | GT3     | T       | 1    | 1    | at_home  | other    |    |
| 499 | MS     | F   | 16  | U       | GT3     | T       | 2    | 2    | services | other    |    |

8 rows x 33 columns

Una muestra aleatoria generalmente nos da una idea de los posibles valores contenidos en el *dataset*, así podemos imaginar el rango de los datos y si son numéricos o categóricos.

## ✓ Información del *dataset*

Podemos obtener la información del *dataset* para ver cuántos registros tiene (*entries*), sus índices, el total de columnas y el tipo de dato de cada columna, generalmente **object** es categórico y **float64** o **int64** son numéricos.

También podemos ver el nombre de la columna, si tiene valores no nulos y cuánta memoria ocupan los datos cargados.

```
dataset.info()
```

```
↔ <class 'pandas.core.frame.DataFrame'>
RangeIndex: 649 entries, 0 to 648
Data columns (total 33 columns):
#   Column              Non-Null Count  Dtype
---  -
0   school              649 non-null    object
1   sex                 649 non-null    object
2   age                 649 non-null    int64
3   address             649 non-null    object
4   famsize             649 non-null    object
5   Pstatus             649 non-null    object
6   Medu                649 non-null    int64
7   Fedu                649 non-null    int64
8   Mjob                649 non-null    object
9   Fjob                649 non-null    object
10  reason              649 non-null    object
11  guardian            649 non-null    object
12  traveltime          649 non-null    int64
13  studytime           649 non-null    int64
14  failures            649 non-null    int64
15  schoolsup            649 non-null    object
16  famsup              649 non-null    object
17  paid                649 non-null    object
18  activities          649 non-null    object
19  nursery             649 non-null    object
20  higher              649 non-null    object
21  internet            649 non-null    object
22  romantic            649 non-null    object
23  famrel              649 non-null    int64
24  freetime           649 non-null    int64
25  goout               649 non-null    int64
26  Dalc                649 non-null    int64
27  Walc                649 non-null    int64
28  health              649 non-null    int64
29  absences            649 non-null    int64
30  G1                  649 non-null    int64
31  G2                  649 non-null    int64
32  G3                  649 non-null    int64
dtypes: int64(16), object(17)
memory usage: 167.4+ KB
```

## ✓ Inspeccionar columnas

Los *datasets* cargados por Pandas son formalmente **DataFrames**, estos representan tablas de datos divididos en filas y columnas, para recuperar una columna del *dataframe*, debemos acceder por su nombre y los corchetes.

Las columnas son **Series** de pandas y contienen todos los valores de la columna asociada.

```
schools = dataset["school"]
```

```
schools
```



|     | school |
|-----|--------|
| 0   | GP     |
| 1   | GP     |
| 2   | GP     |
| 3   | GP     |
| 4   | GP     |
| ... | ...    |
| 644 | MS     |
| 645 | MS     |
| 646 | MS     |
| 647 | MS     |
| 648 | MS     |

649 rows × 1 columns

**dtype:** object

## ✓ Analizar una columna categórica

Generalmente las columnas son numéricas o categóricas, en el caso de ser categórica, podemos ver cuáles son los valores únicos que contienen

```
schools.unique()
```

```
↵ array(['GP', 'MS'], dtype=object)
```

## ✓ Contar los elementos de cada categoría

Una forma primitiva de contar los elementos de cada categoría, es filtrar la serie para los valores que sean iguales a una categoría, por ejemplo, `<serie>[<serie> == <valor>]` y luego aplicar el conteo.

Aunque después veremos formas más estilizadas.

```
schools[schools == "GP"].count()
```

```
↵ np.int64(423)
```

```
schools[schools == "MS"].count()
```

```
↵ np.int64(226)
```

## ✓ Reportar los totales para cada categoría

Esta forma primitiva nos permitiría crear un reporte manual, aunque no es lo mejor que podemos hacer.

```
gp = schools[schools == "GP"].count()
ms = schools[schools == "MS"].count()

print("Gabriel Pereira (GP) | Mousinho da Silveira (MS)")
print("-" * 48)
print(f"{gp:^20} | {ms:^25}")
```

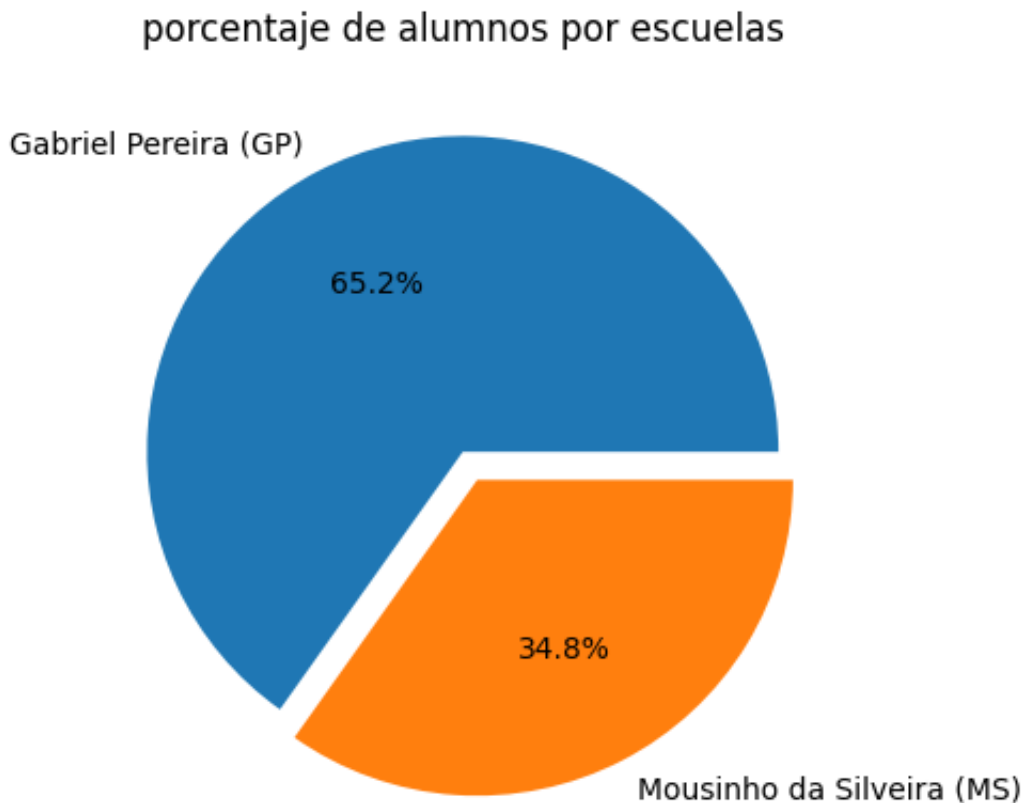
```
↵ Gabriel Pereira (GP) | Mousinho da Silveira (MS)
    423                |                226
```

## ✓ Visualizar la proporción entre las categorías (pastel)

Aunque el proceso es muy manual hasta este momento, podemos si hay pocas categorías, podemos reportar rápidamente con una gráfica de pastel, cuál es la proporción para cada categoría.

```
import matplotlib.pyplot as plt

plt.pie([gp, ms],
        labels=["Gabriel Pereira (GP)", "Mousinho da Silveira (MS)"],
        autopct="%.1f%%", explode = [0, 0.1])
plt.title("porcentaje de alumnos por escuelas")
plt.show()
```



## ✓ Reportar los conteos de una columna categórica

Otra forma de generar un reporte manual es mediante un **DataFrame** manual, con los valores del conteo.

```
import pandas

reportSchool = pandas.DataFrame({
    "Gabriel Pereira (GP)": [gp],
    "Mousinho da Silveira (MS)": [ms]
}, index=["Total"])

reportSchool
```



|  | Gabriel Pereira (GP) | Mousinho da Silveira (MS) |
|--|----------------------|---------------------------|
|--|----------------------|---------------------------|

|       |     |     |
|-------|-----|-----|
| Total | 423 | 226 |
|-------|-----|-----|

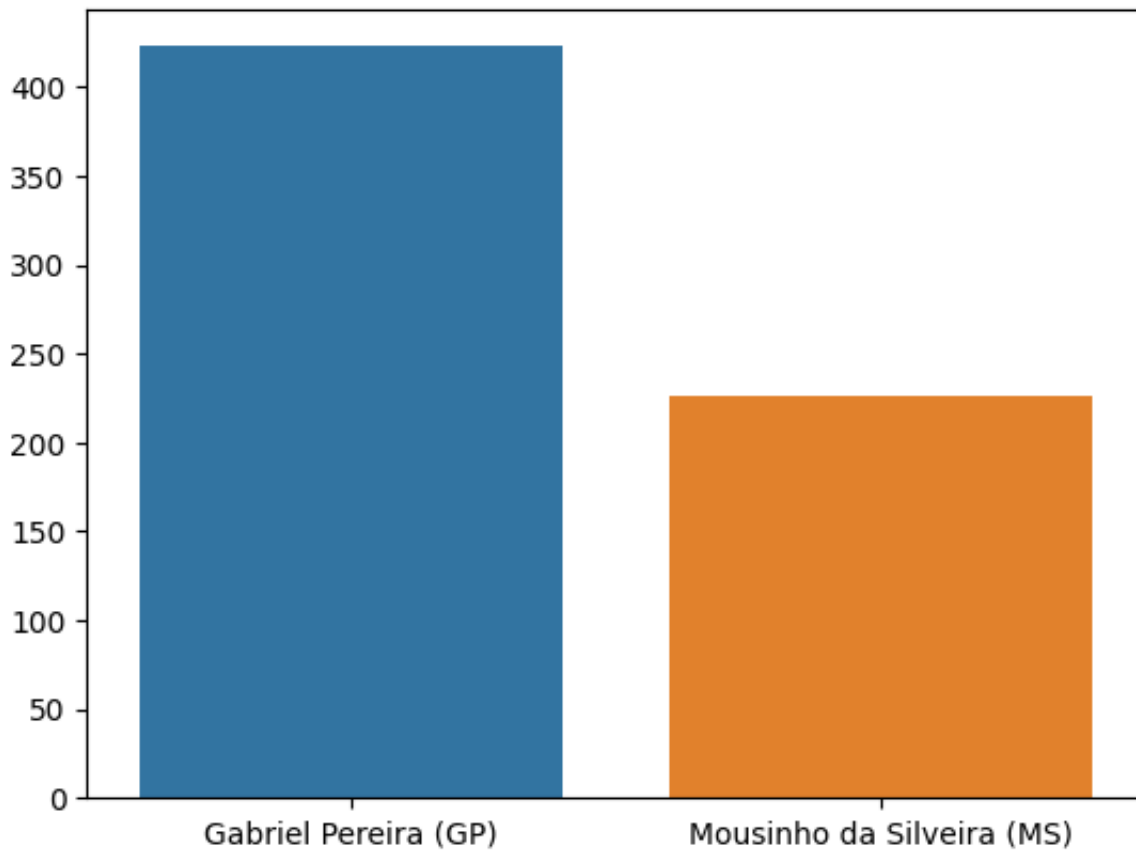


## ✓ Visualizar el total en cada categoría (barras)

Para muchas categorías a veces es mejor mostrar las barras de conteos (alturas) para cada categoría, ya que una gráfica de pastel podría quedar muy saturada.

```
import seaborn
```


```
axis = seaborn.barplot(reportSchool)
```

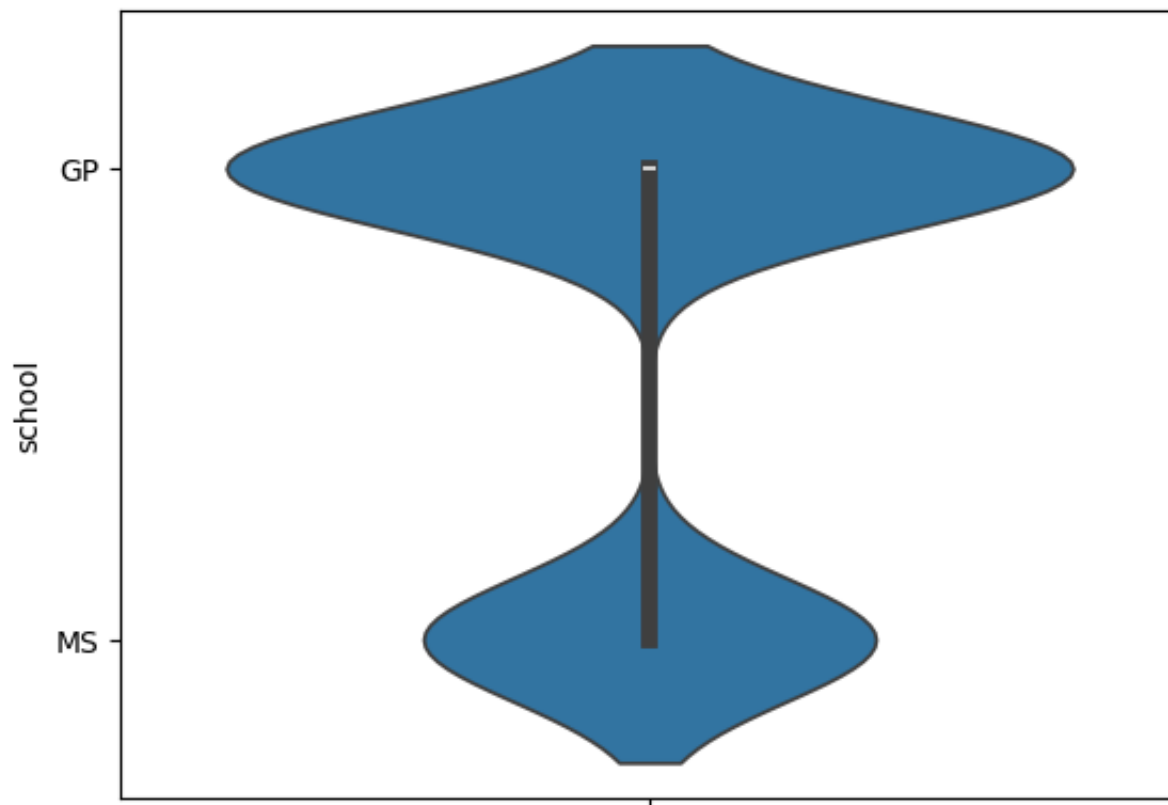


### ✓ Visualizar la distribución de cada categoría (violín)

Para no estar haciendo conteos, una visualización rápida y potente es mostrar la proporción en cada categoría usando una gráfica de violín, esta mostrará qué tantos datos tiene cada categoría en forma de un ancho. Entre más ancha sea una categoría, mayor cantidad de elementos tendrá.

```
seaborn.violinplot(schools)
```

 <Axes: ylabel='school'>



## ✓ Análisis de una columna numérica

Para cuando la columna no es categórica sino numérica, podemos hacer otro tipo de tratamiento de datos.

Por ejemplo, en el conjunto de datos tenemos la columna edad, con la edad de los alumnos de cada escuela.



```
ages = dataset["age"]
```

```
ages
```



| age |     |
|-----|-----|
| 0   | 18  |
| 1   | 17  |
| 2   | 15  |
| 3   | 15  |
| 4   | 16  |
| ... | ... |
| 644 | 19  |
| 645 | 18  |
| 646 | 18  |
| 647 | 17  |
| 648 | 18  |

649 rows × 1 columns

**dtype:** int64

Las variables discretas como números enteros, pueden inspeccionarse para ver sus valores únicos, si consideramos que serán pocos.

Aunque más de 10 valores únicos ya no se consideran tan únicos. Por ejemplo, si las edades fueran de 1 a 100.

```
ages.unique()
```



```
array([18, 17, 15, 16, 19, 22, 20, 21])
```

## ✓ Describir estadísticamente una columna numérica

Las columnas pueden describirse estadísticamente, usando los estadísticos fundamentales:

- **Total** - Número de elementos en la columna
- **Media** - Valor medio de los valores
- **Desviación** - Desviación estándar de los valores, (la raíz cuadrada de la varianza de los datos). Indica qué tanto se alejan los datos de la media en la misma magnitud
- **Mínimo** - El valor mínimo de la columna
- $Q_1$  - Cuartil 1, representa el valor de los datos ordenados al 25% del total de datos (mínimo inferior al 25%)
- $Q_2$  - Cuartil 2, representa el valor de los datos ordenados al 50% del total de datos (mínimo inferior al 50%), este valor coincide con la mediana. Si la mediana es menor a la media, hay un sesgo positivo, es decir, los datos se cargan hacia arriba, si la mediana es mayor a la media, hay un sesgo negativo, es decir, los datos se cargan hacia abajo.
- $Q_3$  - Cuartil 3, representa el valor de los datos ordenados al 75% del total de datos (mínimo inferior al 75%)
- **Máximo** - El máximo de la columna.

Este reporte estadístico nos ayudará a entender los valores del eje, saber si están cargados hacia arriba o hacia abajo.

```
ages.describe()
```



age

|       |            |
|-------|------------|
| count | 649.000000 |
| mean  | 16.744222  |
| std   | 1.218138   |
| min   | 15.000000  |
| 25%   | 16.000000  |
| 50%   | 17.000000  |
| 75%   | 18.000000  |
| max   | 22.000000  |

**dtype:** float64

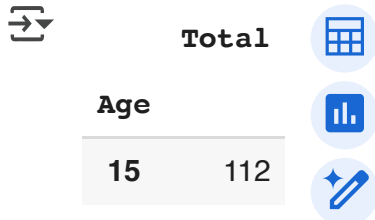
## ✓ Contar todos los elementos en cada valor único

Una forma primitiva de contar los valores para cada valor único, es mediante una lista generada y el uso de tuplas, la lista retiene una tupla para cada valor extraído de la lista de valores únicos, además procesa cuántos elementos tiene la serie para el valor único.

Luego esto se convierte en un **DataFrame** para generar un reporte.

Esto es útil para reportar cada categoría o valor único en columnas numéricas discretas o columnas categóricas.

```
counts = [(age, ages[ages == age].count()) for age in ages.sort_values().unique()]
reportAge = pandas.DataFrame(counts, columns=["Age", "Total"]).set_index("Age")
reportAge
```



|     | Total |
|-----|-------|
| Age |       |
| 15  | 112   |
| 16  | 177   |
| 17  | 179   |
| 18  | 140   |
| 19  | 32    |
| 20  | 6     |
| 21  | 2     |
| 22  | 1     |

Próximos pasos:

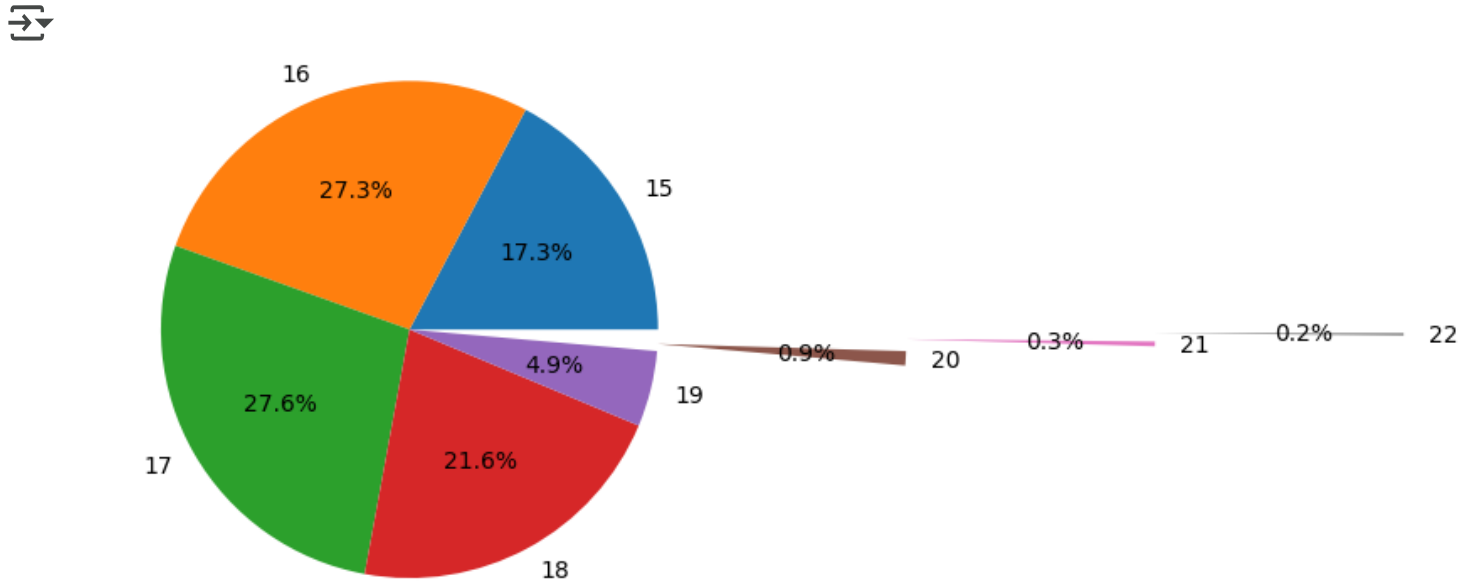
[Ver gráficos recomendados](#)

[New interactive sheet](#)

## ✓ Visualizar el conteo de todos los elementos de cada valor único


Mediante el reporte, podemos automatizar una gráfica de pastel o barras para reportar todos los conteos al mismo tiempo

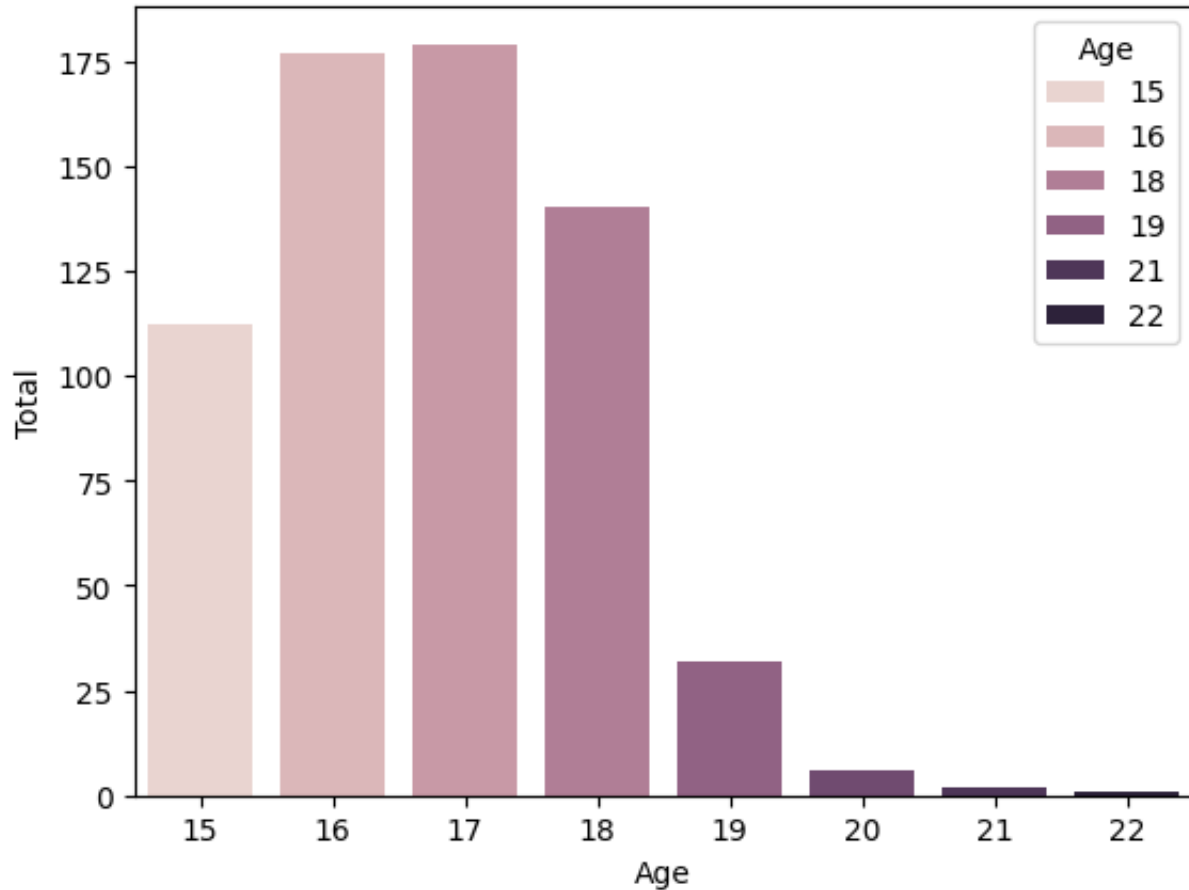
```
pyplot.pie(reportAge["Total"], labels=reportAge.index,  
            explode=[0, 0, 0, 0, 0, 1.0, 2, 3],  
            autopct="%.1f%%")  
pyplot.show()
```



Generalmente las barras funcionan mejor en muchas categorías

```
seaborn.barplot(reportAge, x=reportAge.index, y="Total", hue=reportAge.index)
```

 <Axes: xlabel='Age', ylabel='Total'>




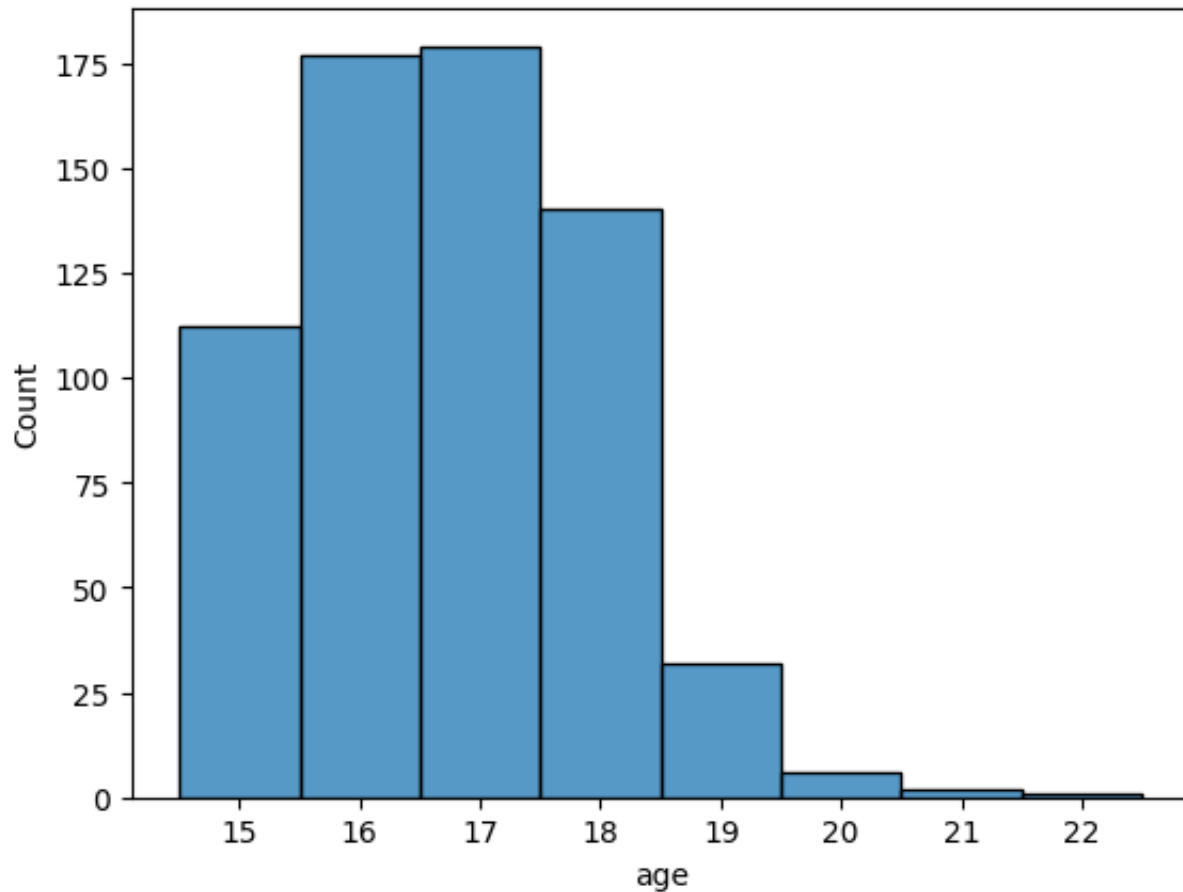
## ✓ Visualizar la distribución de una columna numérica

Dado que las columnas numéricas tienen valores distribuidos entre el mínimo y máximo, generalmente queremos ver en qué intervalos se están concentrando, a esto le llamamos su distribución.

Una forma de visualizar la distribución de un eje numérico es mediante el histograma, que explica las veces en las que un valor cae dentro de un intervalo (*bin*).

```
seaborn.histplot(ages, discrete=True)
```


 <Axes: xlabel='age', ylabel='Count'>

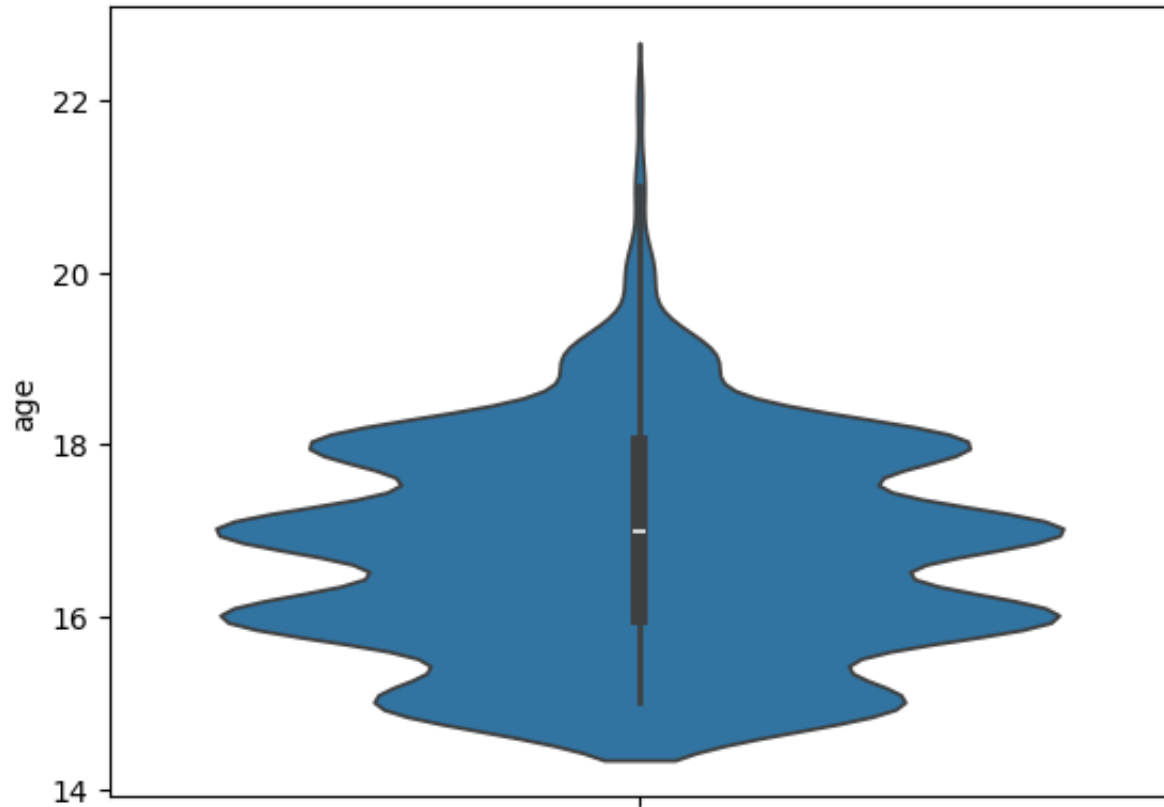


La distribución de una columna numérica también se puede analizar con la gráfica de violín, si la gráfica presenta picos, significa que es evidente que la columna puede ser partida en diferentes categorías o regiones.

Es decir, cada pico demuestra una densidad específica de la población, para nuestro caso, una población de 15, 16, ..., 21 años.

```
seaborn.violinplot(ages)
```

 <Axes: ylabel='age'>



## ✓ Análisis común a una columna numérica

1. Recuperación de la columna

```
final_grades = dataset["G3"]
```

```
final_grades
```



**G3**

|            |     |
|------------|-----|
| <b>0</b>   | 11  |
| <b>1</b>   | 11  |
| <b>2</b>   | 12  |
| <b>3</b>   | 14  |
| <b>4</b>   | 13  |
| ...        | ... |
| <b>644</b> | 10  |
| <b>645</b> | 16  |
| <b>646</b> | 9   |
| <b>647</b> | 10  |
| <b>648</b> | 11  |


649 rows × 1 columns

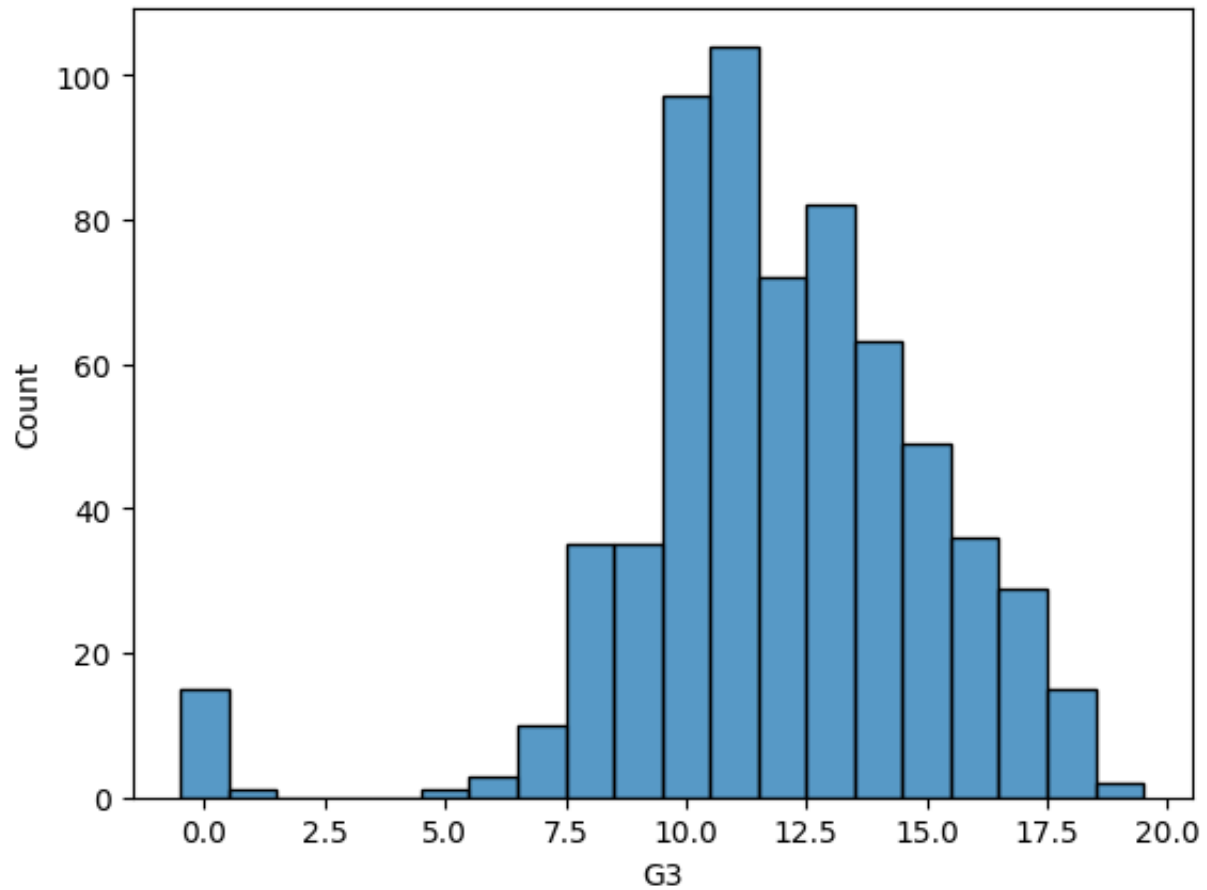
**dtype:** int64

## 2. Inspeccionar la distribución de los datos




```
seaborn.histplot(final_grades, discrete=True)
```

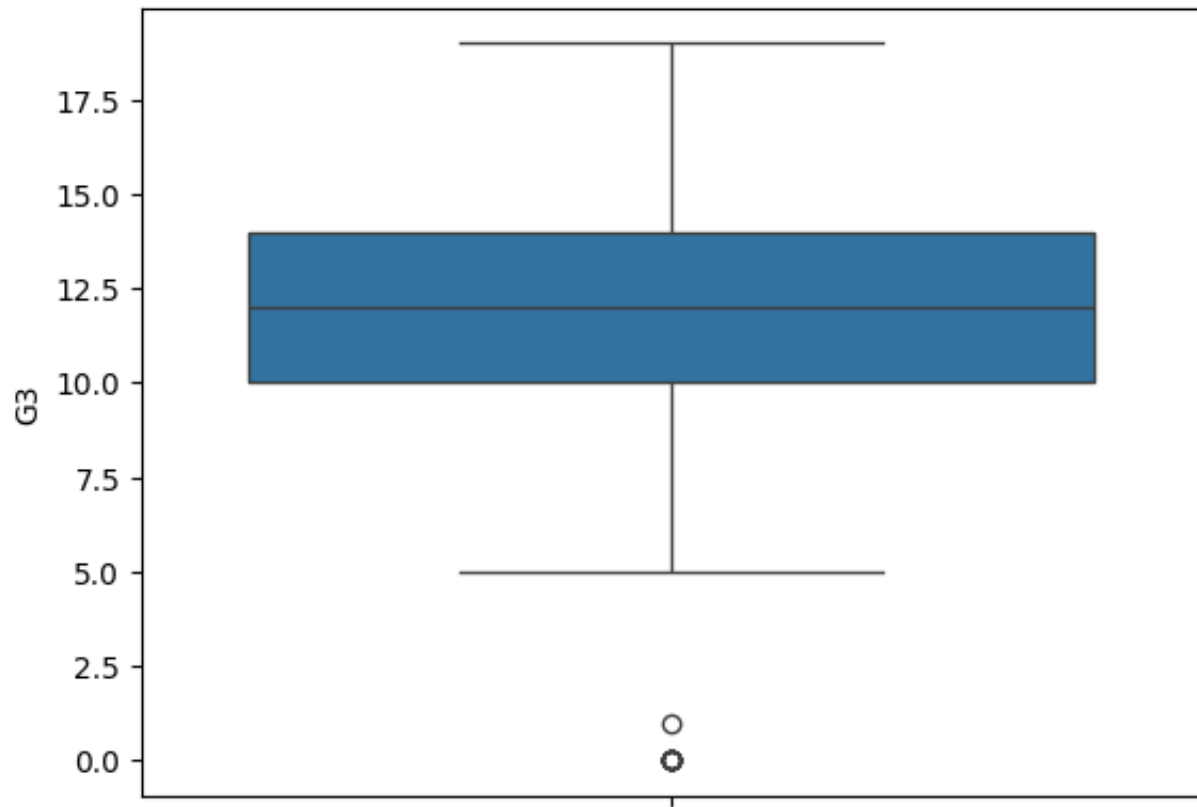
 <Axes: xlabel='G3', ylabel='Count'>



### 3. Inspeccionar los estadísticos principales

```
seaborn.boxplot(final_grades)
```

 <Axes: ylabel='G3'>



## ✓ Análisis conjunto de columnas

También podemos generar otro **DataFrame** seleccionando múltiples columnas, para manejar un conjunto de datos más objetivo.

```
school_grades = dataset[["school", "age", "G3"]]
```

school\_grades



|     | school | age | G3  |
|-----|--------|-----|-----|
| 0   | GP     | 18  | 11  |
| 1   | GP     | 17  | 11  |
| 2   | GP     | 15  | 12  |
| 3   | GP     | 15  | 14  |
| 4   | GP     | 16  | 13  |
| ... | ...    | ... | ... |
| 644 | MS     | 19  | 10  |
| 645 | MS     | 18  | 16  |
| 646 | MS     | 18  | 9   |
| 647 | MS     | 17  | 10  |
| 648 | MS     | 18  | 11  |



649 rows x 3 columns

Próximos pasos:


[Ver gráficos recomendados](#)

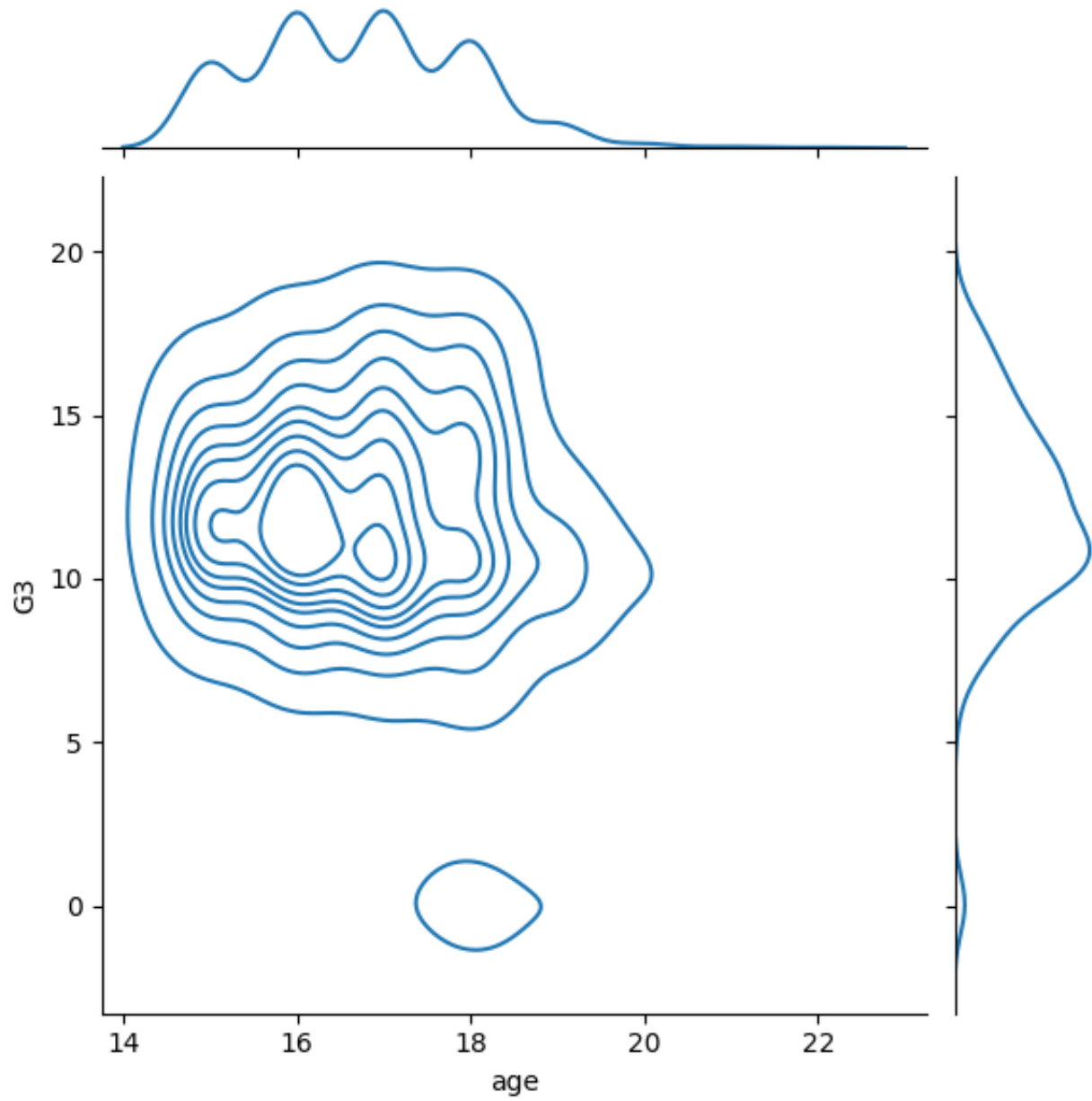
[New interactive sheet](#)

## ✓ Visualizar la distribución entre múltiples columnas

Una gráfica potente es *joinplot* (con modalidad *KDE*) que permite entender la distribución de dos columnas al mismo tiempo, en sus marginales se muestra la distribución de cada columna, y en la gráfica se muestra la acumulación de forma espacial en curvas de nivel.

```
seaborn.jointplot(school_grades, x="age", y="G3", kind="kde")
```

 <seaborn.axisgrid.JointGrid at 0x7a2b0db91310>




Pero la gráfica es aún más potente si separamos los datos por un tercer eje, en este caso la escuela.

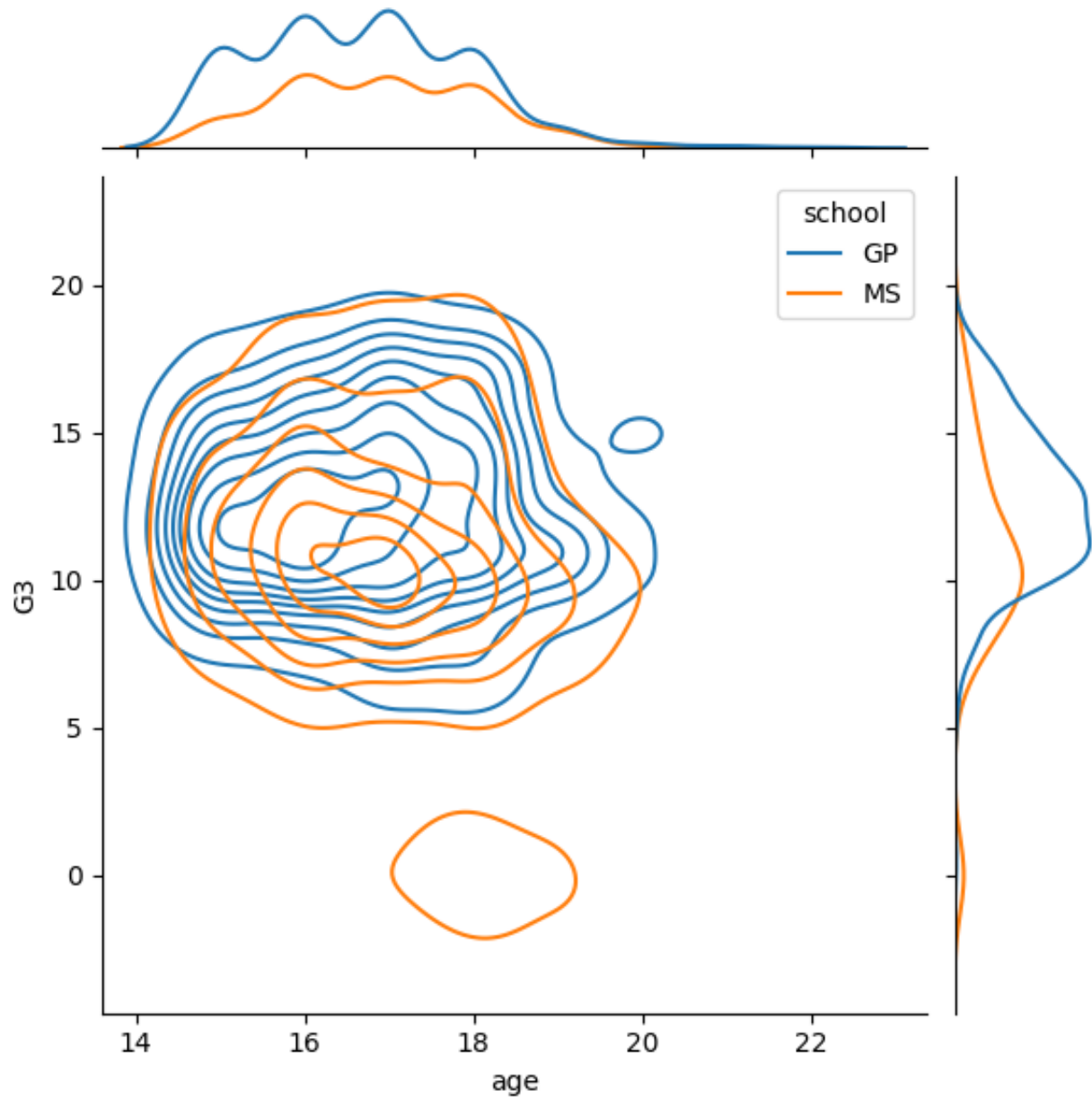
Esta gráfica ahora explica para cada **HUE** o clase cómo se comportan las distribuciones de cada columna.

Por ejemplo, ahora observamos que las notas finales para la escuela *MS* son más bajas que para la escuela *GP*, pero las edades se distribuyen igual, solo que con menos masa de datos.

Así podemos entender que hay menos masa de datos para la escuela *MS* y que sus notas en general son más bajas, aunque el rango es similar.


```
seaborn.jointplot(school_grades, x="age", y="G3", hue="school", kind="kde")
```

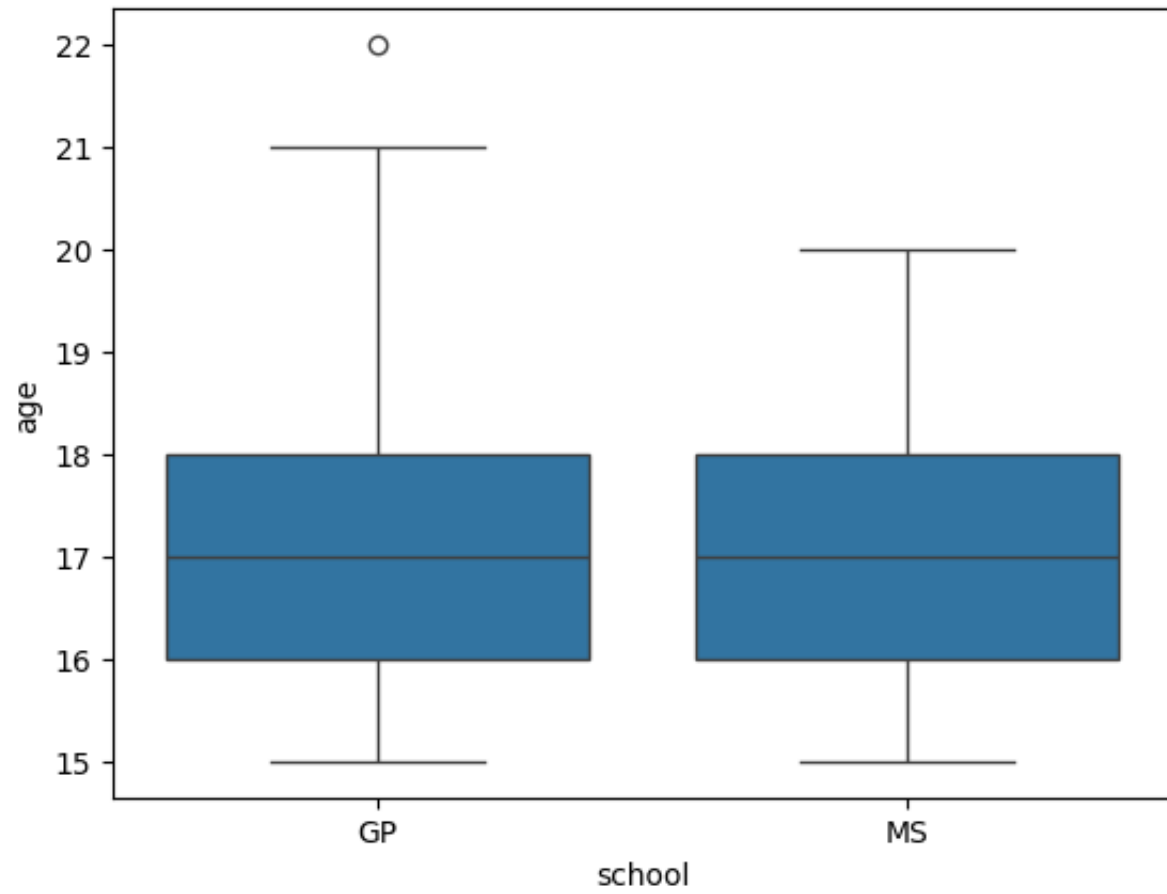
 <seaborn.axisgrid.JointGrid at 0x7a2b0e3d9010>




Esto también se podría observar en las cajas estadísticas.

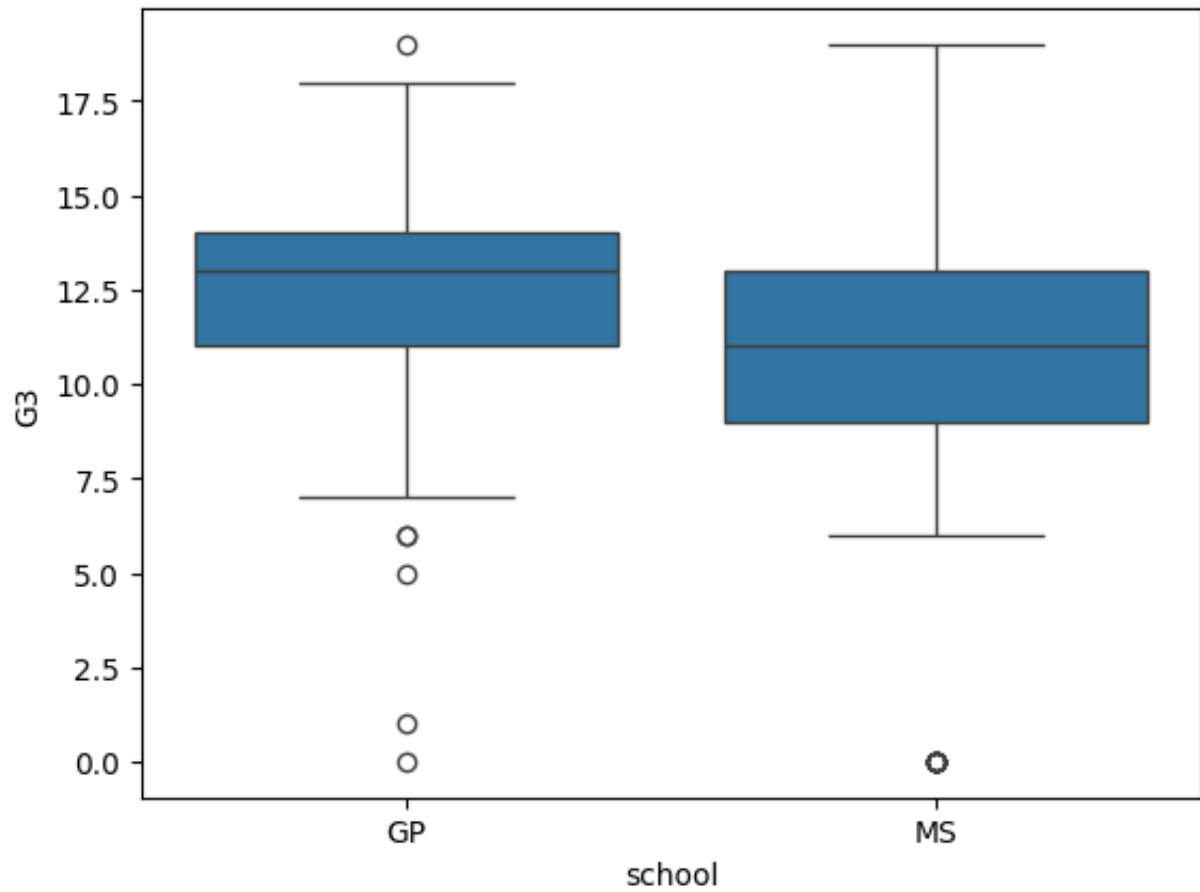
```
seaborn.boxplot(school_grades, x="school", y="age")
```

 <Axes: xlabel='school', ylabel='age'>




```
seaborn.boxplot(school_grades, x="school", y="G3")
```

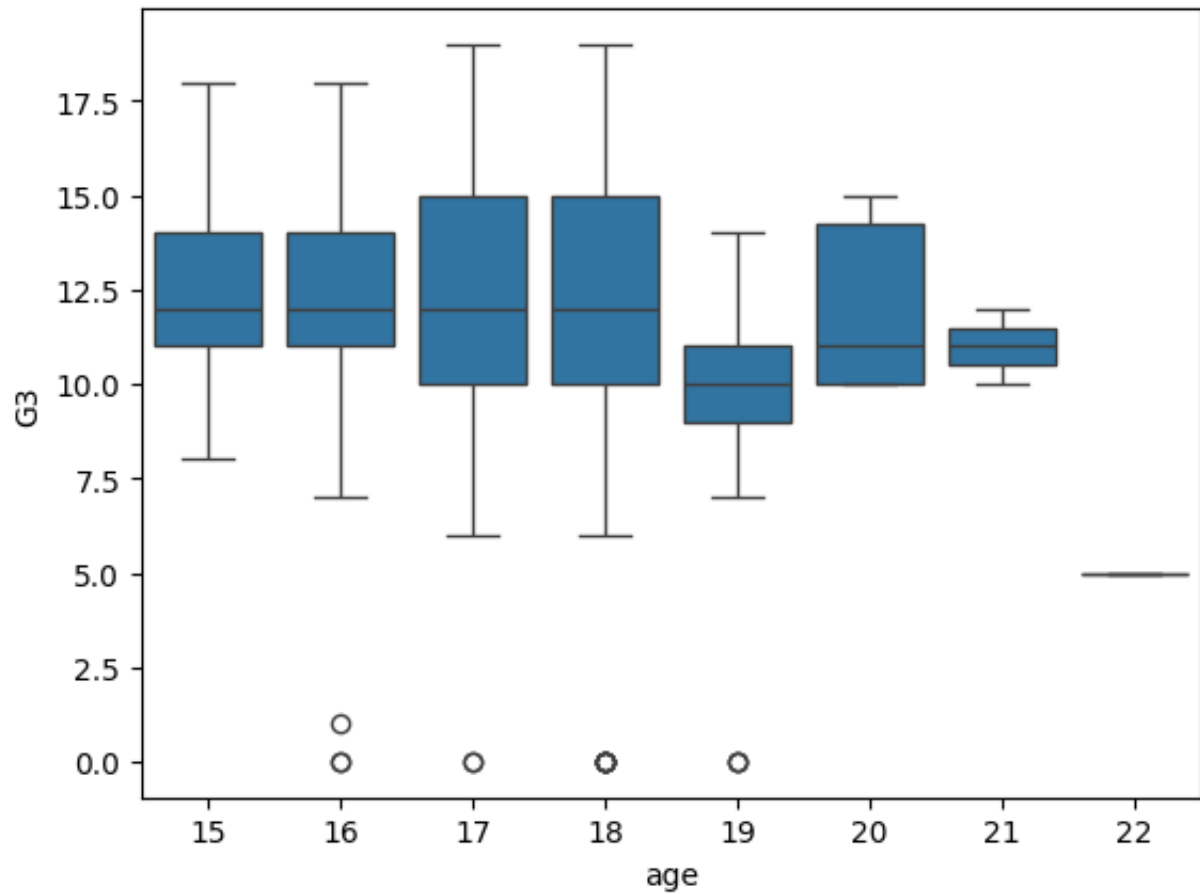
 <Axes: xlabel='school', ylabel='G3'>





```
seaborn.boxplot(school_grades, x="age", y="G3")
```

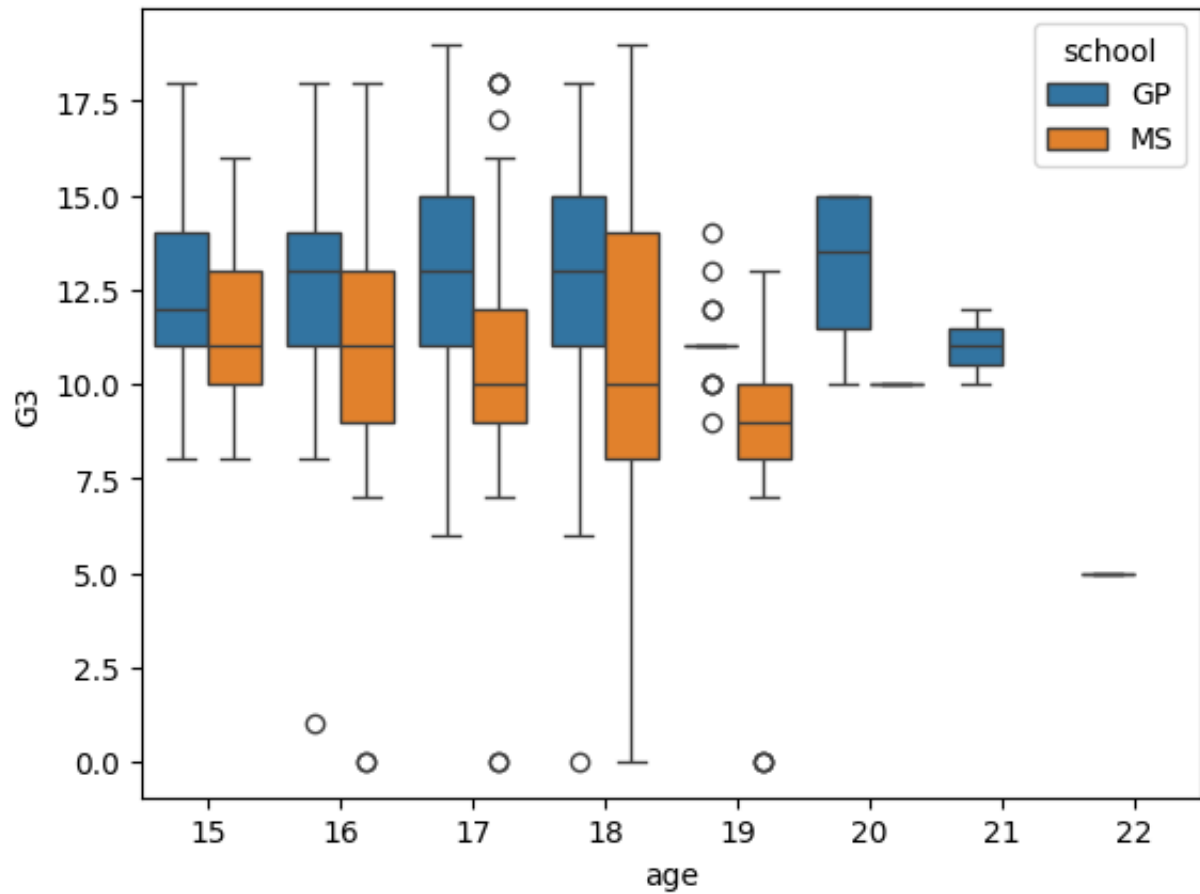
 <Axes: xlabel='age', ylabel='G3'>




La caja estadística es más potente aún si consideramos el **HUE** o clase de segmentación.

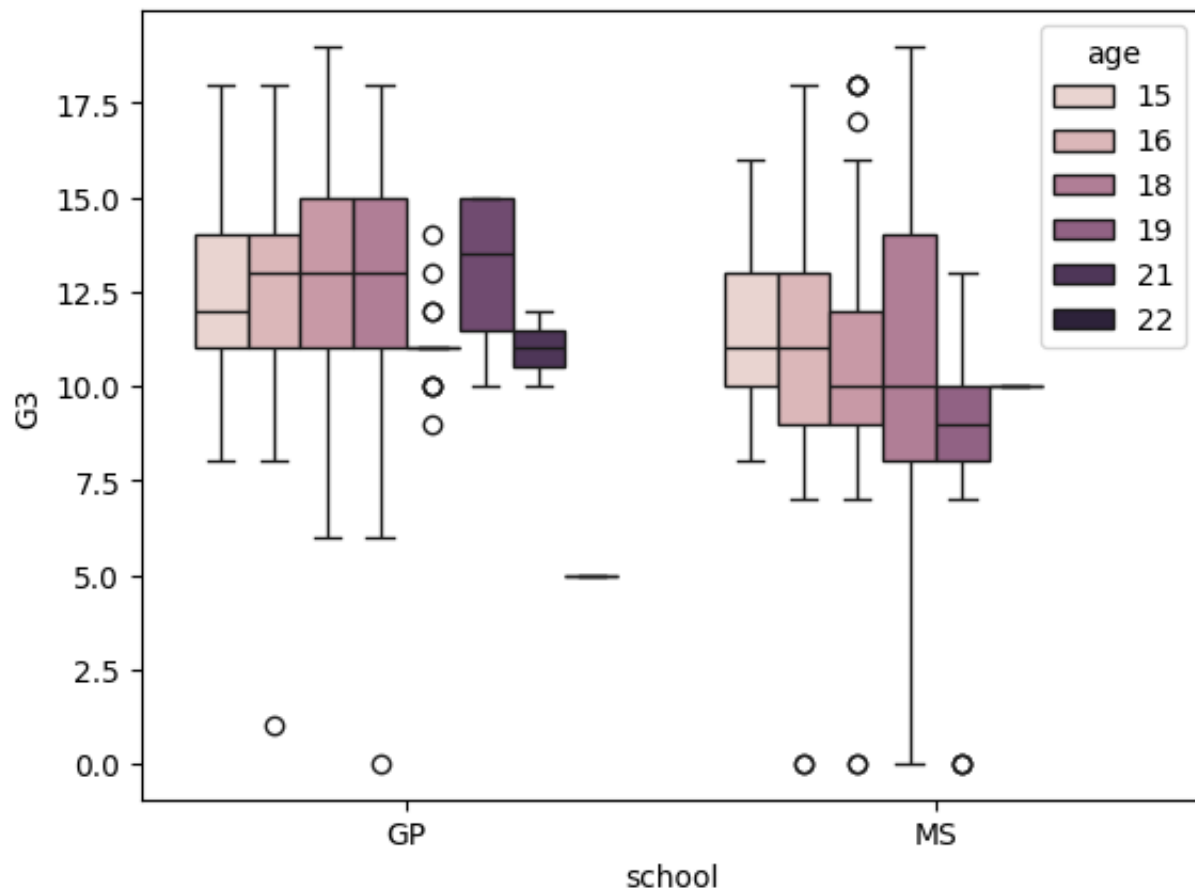
```
seaborn.boxplot(school_grades, x="age", y="G3", hue="school")
```

<Axes: xlabel='age', ylabel='G3'>



```
seaborn.boxplot(school_grades, x="school", y="G3", hue="age")
```

 <Axes: xlabel='school', ylabel='G3'>



## ✓ Reporte estadístico

Además, cuando tenemos un *dataset* objetivo (en el sentido de que todas sus columnas son numéricas), podemos describir el conjunto de datos en general para cada columna numérica.

```
school_grades.describe()
```



|       | age        | G3         |
|-------|------------|------------|
| count | 649.000000 | 649.000000 |
| mean  | 16.744222  | 11.906009  |
| std   | 1.218138   | 3.230656   |
| min   | 15.000000  | 0.000000   |
| 25%   | 16.000000  | 10.000000  |
| 50%   | 17.000000  | 12.000000  |
| 75%   | 18.000000  | 14.000000  |
| max   | 22.000000  | 19.000000  |



## ✓ Agrupar datos de columnas y calcular una agregación


Podemos agrupar los datos para un **DataFrame** y luego sumar, promediar o agregar en general cada grupo (se explicará en el pizarrón).

Por ejemplo, si separamos dos **DataFrames** con los elementos para cada escuela, y luego queremos promediar las notas finales agrupadas por edad, tenemos



```
school_grades_gp = school_grades[school_grades["school"] == "GP"]
school_grades_ms = school_grades[school_grades["school"] == "MS"]

reportSchoolGP = school_grades_gp[["age", "G3"]].groupby("age").aggregate("mean")
reportSchoolMS = school_grades_ms[["age", "G3"]].groupby("age").aggregate("mean")
```


```
school_grades_gp.describe()
```






|       | age        | G3         |
|-------|------------|------------|
| count | 423.000000 | 423.000000 |
| mean  | 16.666667  | 12.576832  |
| std   | 1.244895   | 2.625636   |
| min   | 15.000000  | 0.000000   |
| 25%   | 16.000000  | 11.000000  |
| 50%   | 17.000000  | 13.000000  |
| 75%   | 18.000000  | 14.000000  |
| max   | 22.000000  | 19.000000  |



```
reportSchoolGP
```



|     | G3        |
|-----|-----------|
| age |           |
| 15  | 12.325581 |
| 16  | 12.491071 |
| 17  | 12.991453 |
| 18  | 12.773810 |
| 19  | 11.117647 |
| 20  | 13.000000 |
| 21  | 11.000000 |
| 22  | 5.000000  |



Próximos pasos:

-  [Ver gráficos recomendados](#)
- [New interactive sheet](#)

```
school_grades_ms.describe()
```



|       | age        | G3         |
|-------|------------|------------|
| count | 226.000000 | 226.000000 |
| mean  | 16.889381  | 10.650442  |
| std   | 1.155152   | 3.833991   |
| min   | 15.000000  | 0.000000   |
| 25%   | 16.000000  | 9.000000   |
| 50%   | 17.000000  | 11.000000  |
| 75%   | 18.000000  | 13.000000  |
| max   | 20.000000  | 19.000000  |



```
reportSchoolMS
```



|     | G3        |
|-----|-----------|
| age |           |
| 15  | 11.384615 |
| 16  | 11.138462 |
| 17  | 10.903226 |
| 18  | 10.267857 |
| 19  | 7.733333  |
| 20  | 10.000000 |



Próximos pasos:

 [Ver gráficos recomendados](#)

[New interactive sheet](#)

## ✓ Problema 5 - Flor Íris

<https://hf-mirror.com/datasets/skorkmaz88/iris>

```
import pandas
```

```
# https://huggingface.co/datasets/skorkmaz88/iris/resolve/main/data/train-00000-of-00000.parquet
iris = pandas.read_parquet("hf://datasets/skorkmaz88/iris/data/train-00000-of-00000.parquet")
```

```
iris
```



|     | sepal.length | sepal.width | petal.length | petal.width | variety |
|-----|--------------|-------------|--------------|-------------|---------|
| 0   | 5.1          | 3.5         | 1.4          | 0.2         | 0       |
| 1   | 4.9          | 3.0         | 1.4          | 0.2         | 0       |
| 2   | 4.7          | 3.2         | 1.3          | 0.2         | 0       |
| 3   | 4.6          | 3.1         | 1.5          | 0.2         | 0       |
| 4   | 5.0          | 3.6         | 1.4          | 0.2         | 0       |
| ... | ...          | ...         | ...          | ...         | ...     |
| 145 | 6.7          | 3.0         | 5.2          | 2.3         | 2       |
| 146 | 6.3          | 2.5         | 5.0          | 1.9         | 2       |
| 147 | 6.5          | 3.0         | 5.2          | 2.0         | 2       |
| 148 | 6.2          | 3.4         | 5.4          | 2.3         | 2       |
| 149 | 5.9          | 3.0         | 5.1          | 1.8         | 2       |



150 rows x 5 columns

Próximos pasos:

[Ver gráficos recomendados](#)[New interactive sheet](#)

```
iris.head()
```



|   | sepal.length | sepal.width | petal.length | petal.width | variety |
|---|--------------|-------------|--------------|-------------|---------|
| 0 | 5.1          | 3.5         | 1.4          | 0.2         | 0       |
| 1 | 4.9          | 3.0         | 1.4          | 0.2         | 0       |
| 2 | 4.7          | 3.2         | 1.3          | 0.2         | 0       |
| 3 | 4.6          | 3.1         | 1.5          | 0.2         | 0       |
| 4 | 5.0          | 3.6         | 1.4          | 0.2         | 0       |



Próximos pasos:

[Ver gráficos recomendados](#)[New interactive sheet](#)

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  -
0   sepal.length    150 non-null    float32
1   sepal.width     150 non-null    float32
2   petal.length    150 non-null    float32
3   petal.width     150 non-null    float32
4   variety         150 non-null    int64
dtypes: float32(4), int64(1)
memory usage: 3.6 KB
```

```
iris.sample(10)
```

```

    sepal.length  sepal.width  petal.length  petal.width  variety
34              4.9          3.1           1.5          0.2         0
121             5.6          2.8           4.9          2.0         2
134             6.1          2.6           5.6          1.4         2
96              5.7          2.9           4.2          1.3         1
123             6.3          2.7           4.9          1.8         2
144             6.7          3.3           5.7          2.5         2
103             6.3          2.9           5.6          1.8         2
26              5.0          3.4           1.6          0.4         0
67              5.8          2.7           4.1          1.0         1
124             6.7          3.3           5.7          2.1         2
```





```
iris.describe()
```



|       | sepal.length | sepal.width | petal.length | petal.width | variety    |
|-------|--------------|-------------|--------------|-------------|------------|
| count | 150.000000   | 150.000000  | 150.000000   | 150.000000  | 150.000000 |
| mean  | 5.843333     | 3.057333    | 3.758000     | 1.199333    | 1.000000   |
| std   | 0.828066     | 0.435866    | 1.765299     | 0.762238    | 0.819232   |
| min   | 4.300000     | 2.000000    | 1.000000     | 0.100000    | 0.000000   |
| 25%   | 5.100000     | 2.800000    | 1.600000     | 0.300000    | 0.000000   |
| 50%   | 5.800000     | 3.000000    | 4.350000     | 1.300000    | 1.000000   |
| 75%   | 6.400000     | 3.300000    | 5.100000     | 1.800000    | 2.000000   |
| max   | 7.900000     | 4.400000    | 6.900000     | 2.500000    | 2.000000   |



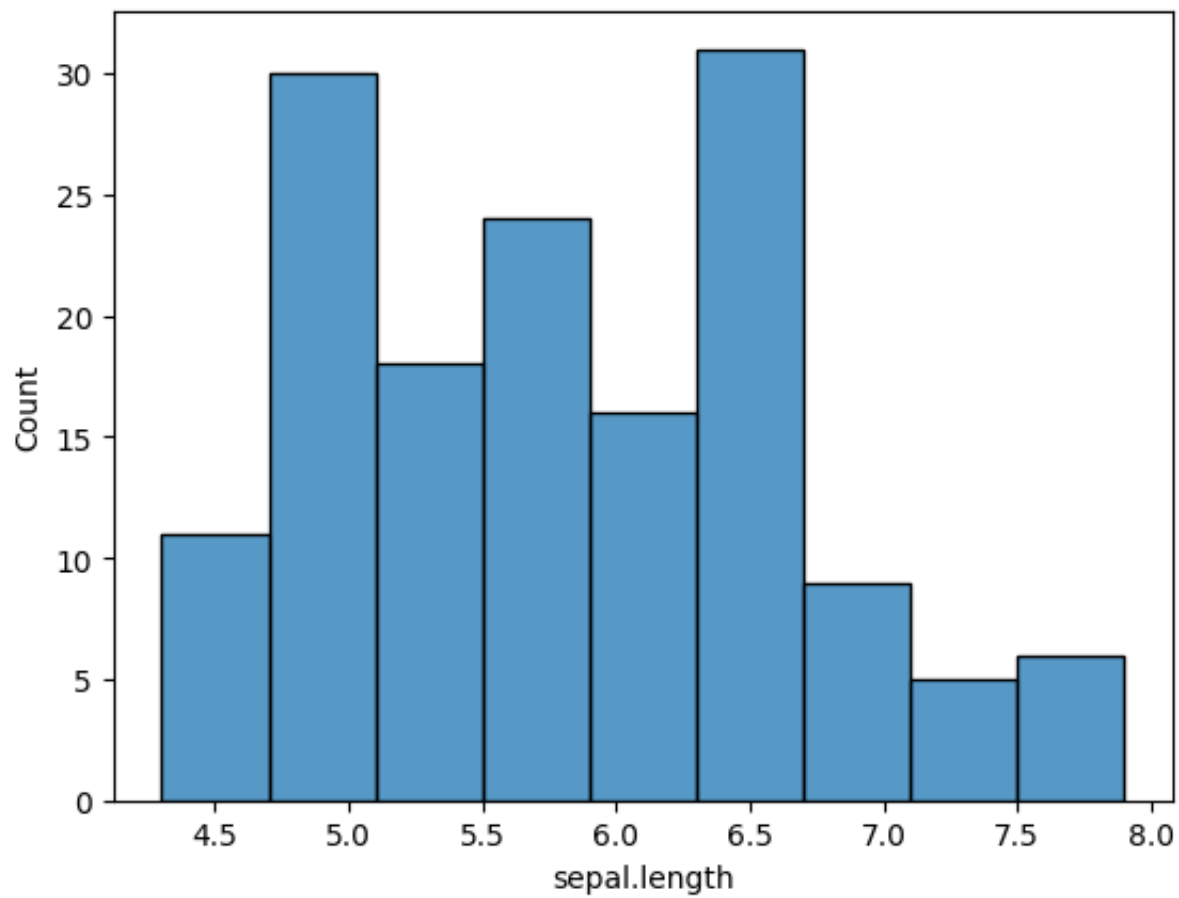
```
sepalLengths = iris["sepal.length"]  
sepalWidths = iris["sepal.length"]
```

```
petalLengths = iris["petal.length"]  
petalWidths = iris["petal.length"]
```


```
import seaborn
```

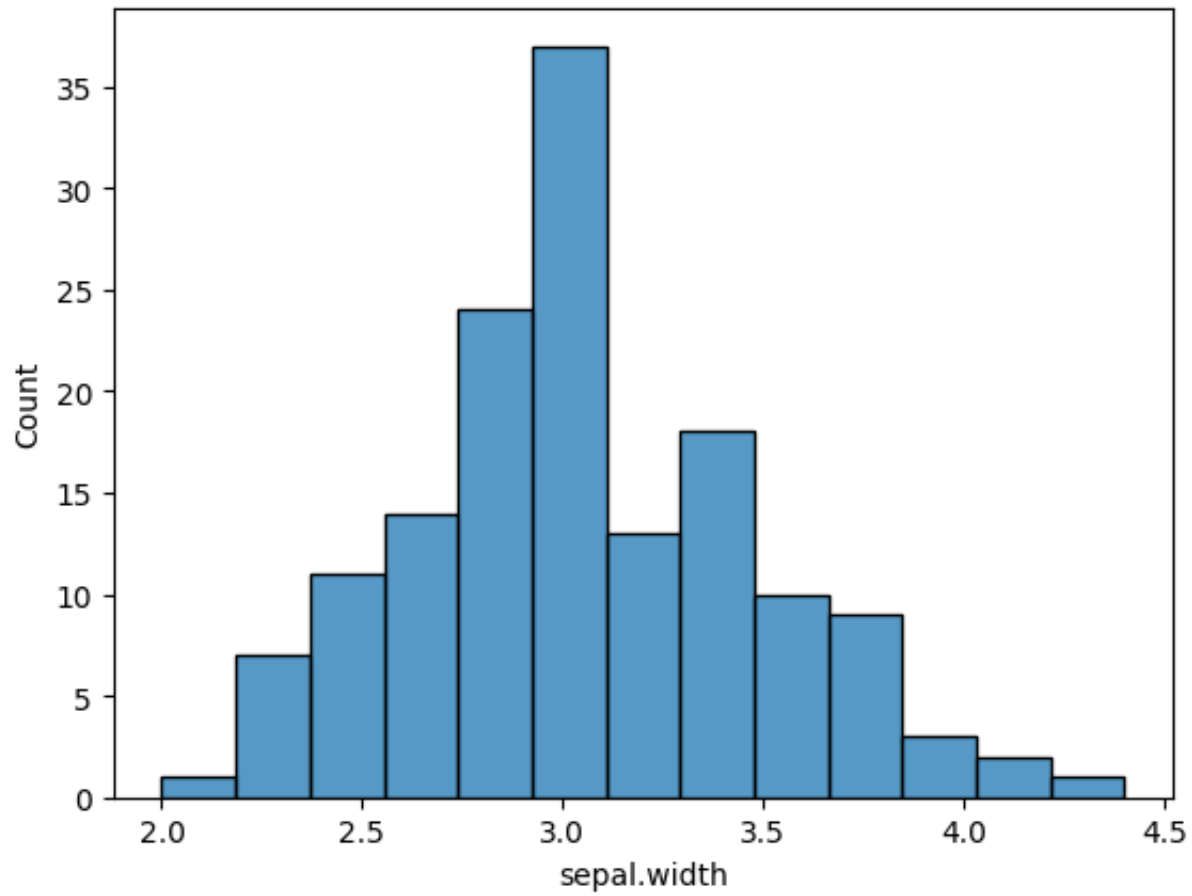
```
seaborn.histplot(iris, x="sepal.length")
```

↔ <Axes: xlabel='sepal.length', ylabel='Count'>




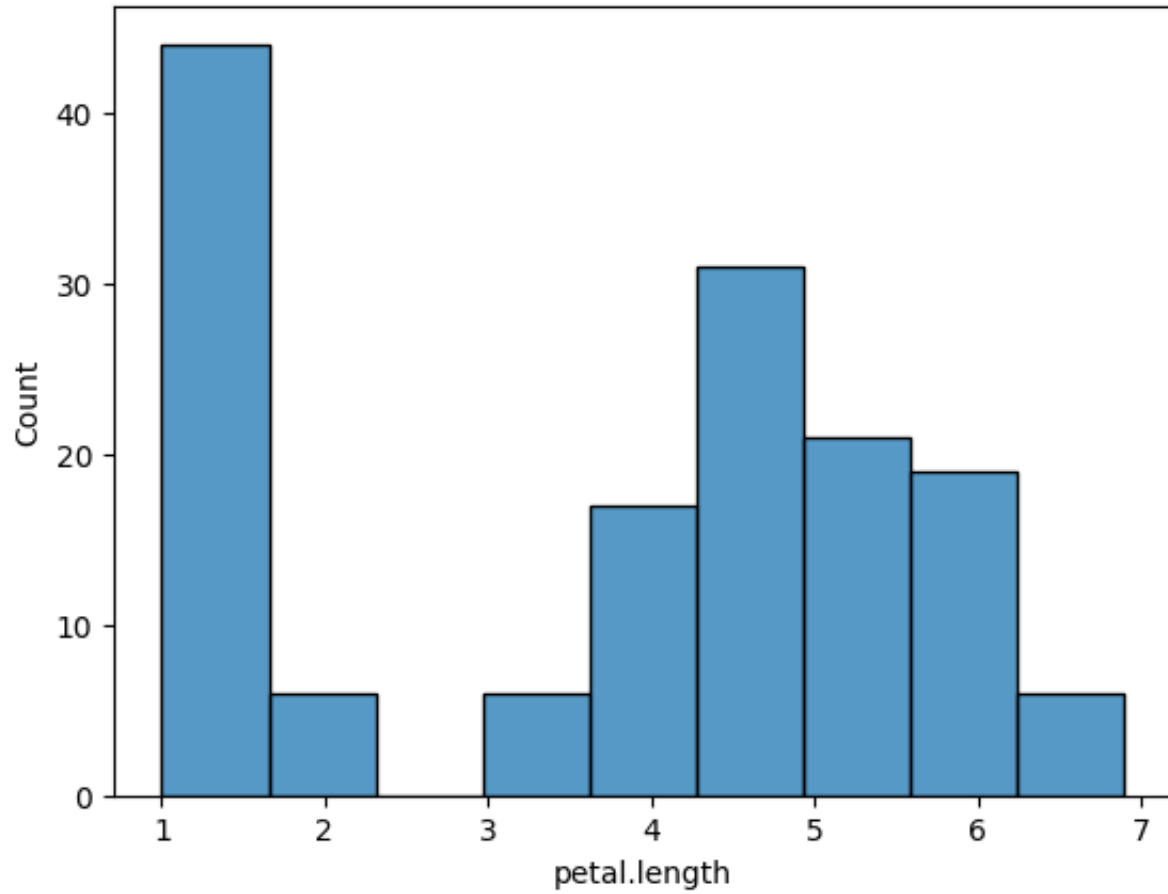
```
seaborn.histplot(iris, x="sepal.width")
```

 <Axes: xlabel='sepal.width', ylabel='Count'>




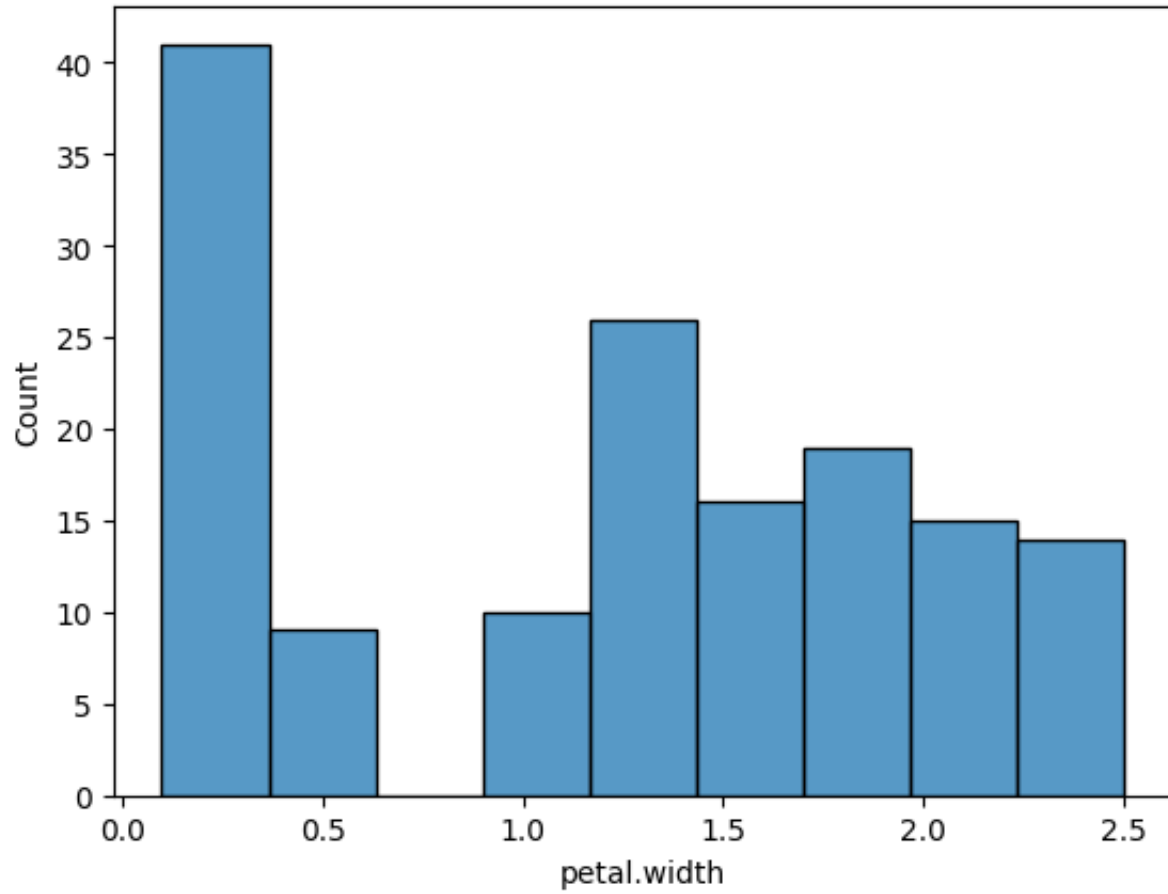
```
seaborn.histplot(iris, x="petal.length")
```

 <Axes: xlabel='petal.length', ylabel='Count'>




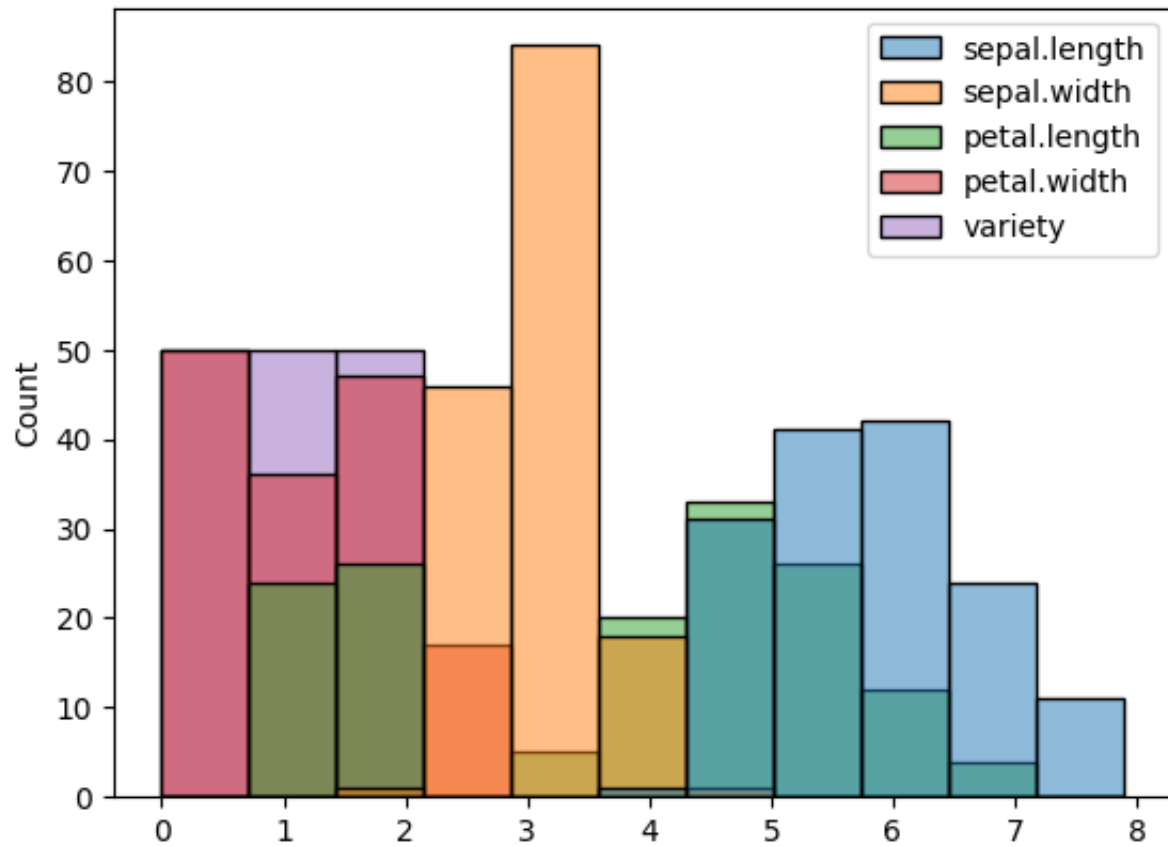
```
seaborn.histplot(iris, x="petal.width")
```

 <Axes: xlabel='petal.width', ylabel='Count'>




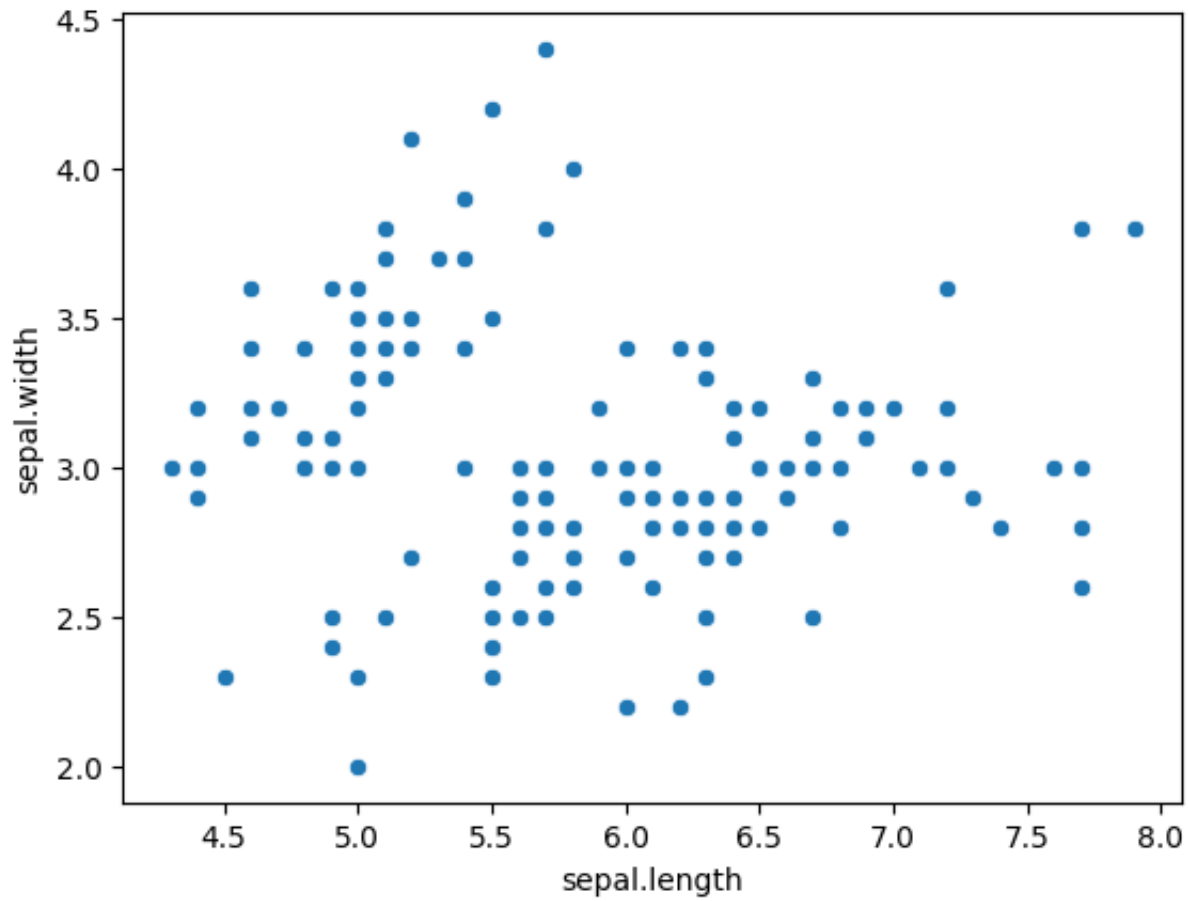
```
seaborn.histplot(iris)
```

 <Axes: ylabel='Count'>




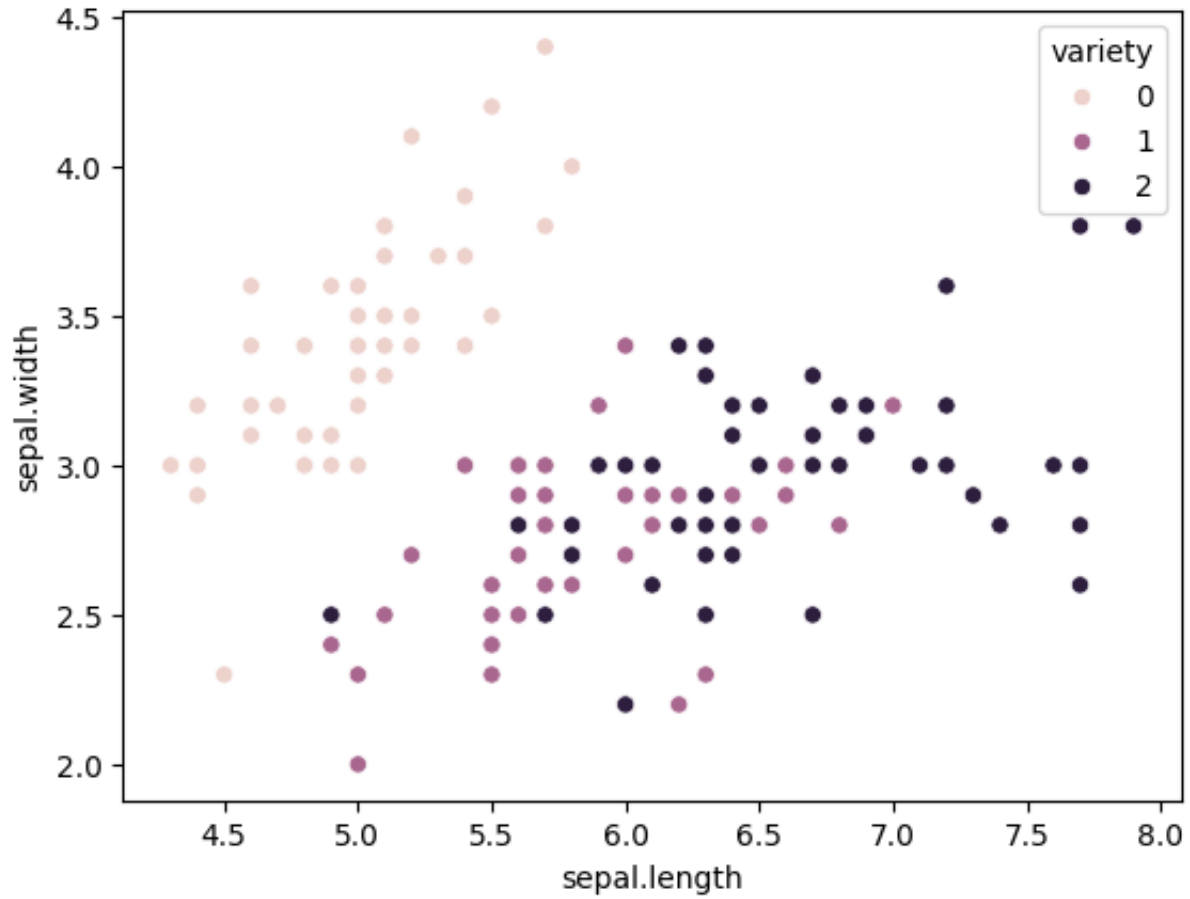
```
seaborn.scatterplot(iris, x="sepal.length", y="sepal.width")
```

 <Axes: xlabel='sepal.length', ylabel='sepal.width'>




```
seaborn.scatterplot(iris, x="sepal.length", y="sepal.width", hue="variety")
```

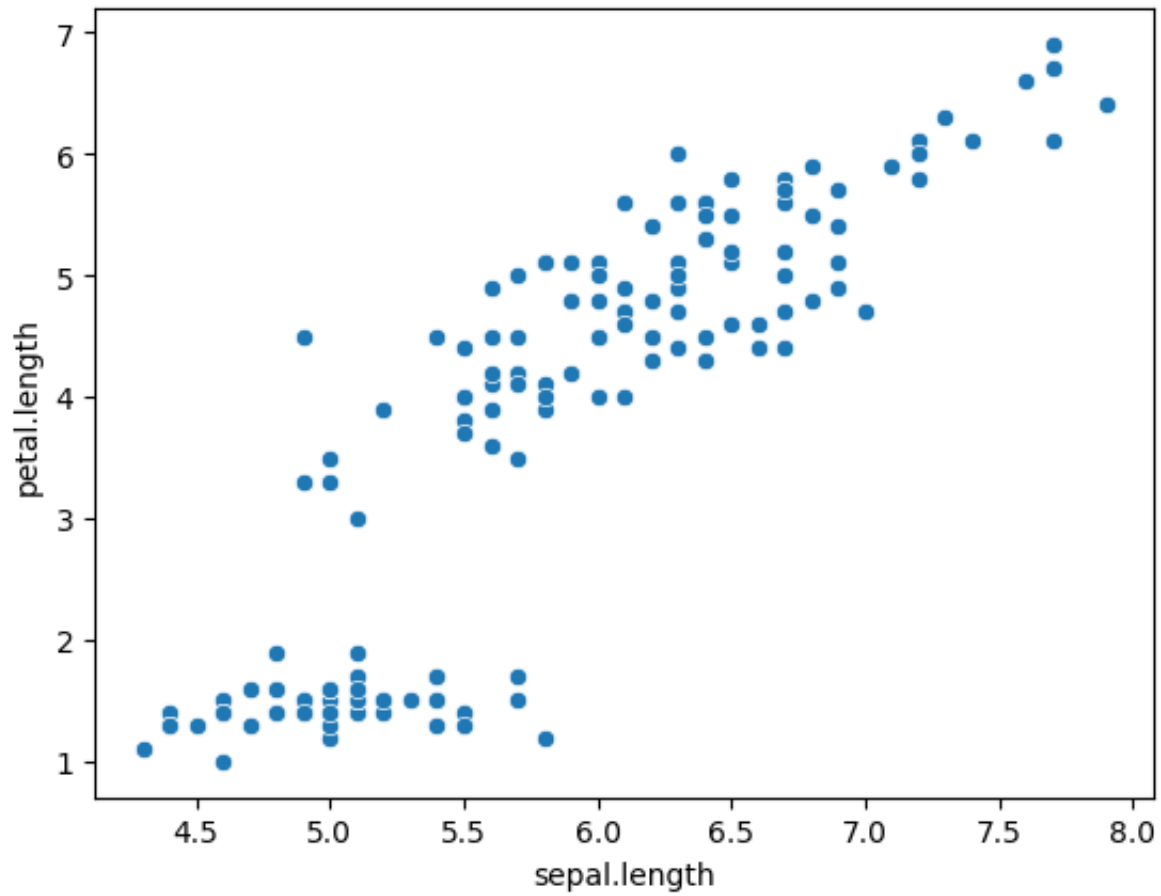
 <Axes: xlabel='sepal.length', ylabel='sepal.width'>





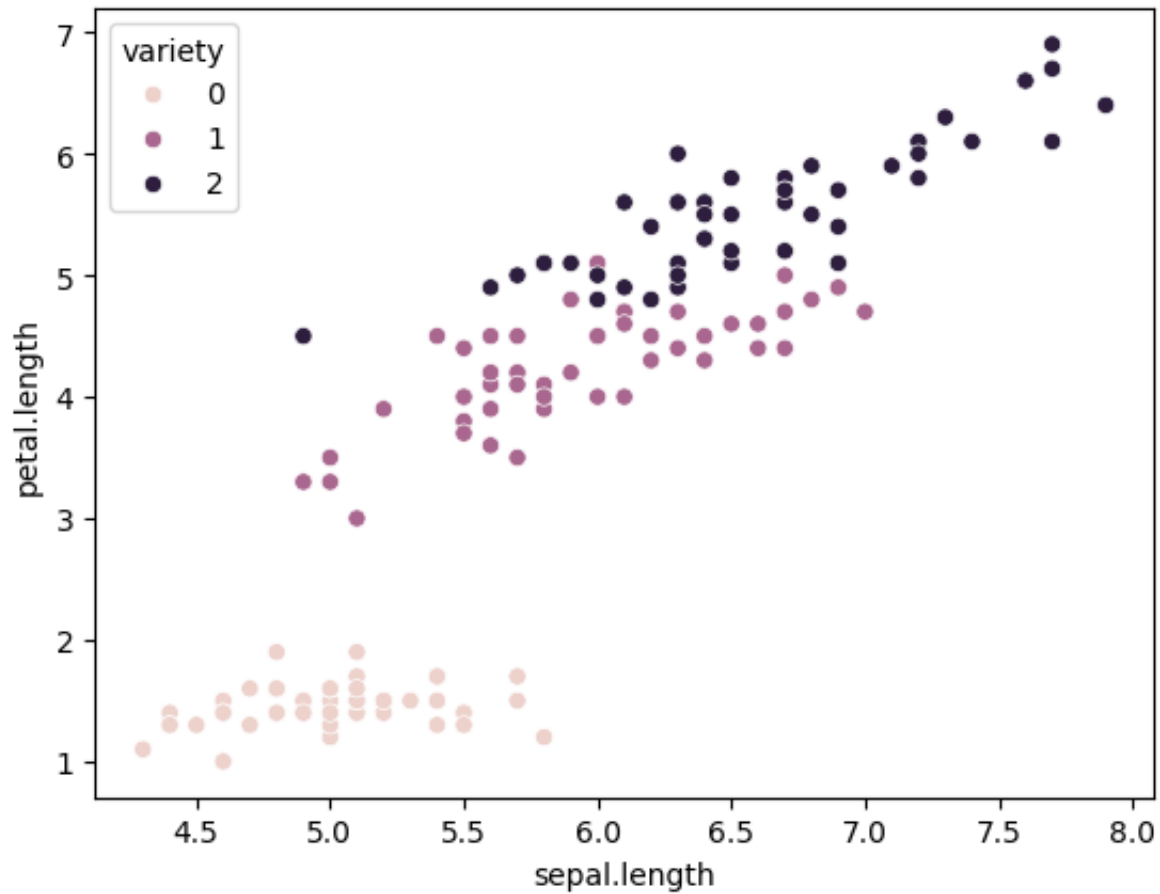
```
seaborn.scatterplot(iris, x="sepal.length", y="petal.length")
```

 <Axes: xlabel='sepal.length', ylabel='petal.length'>



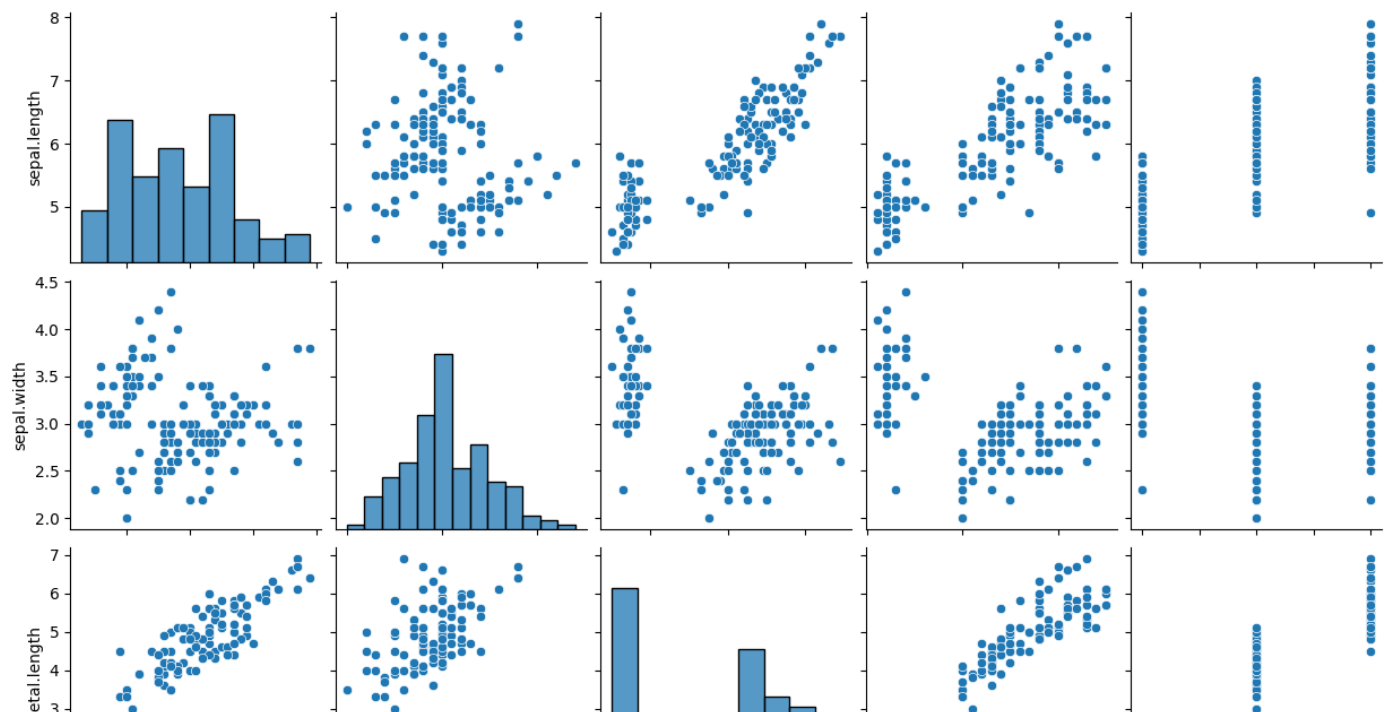
```
seaborn.scatterplot(iris, x="sepal.length", y="petal.length", hue="variety")
```

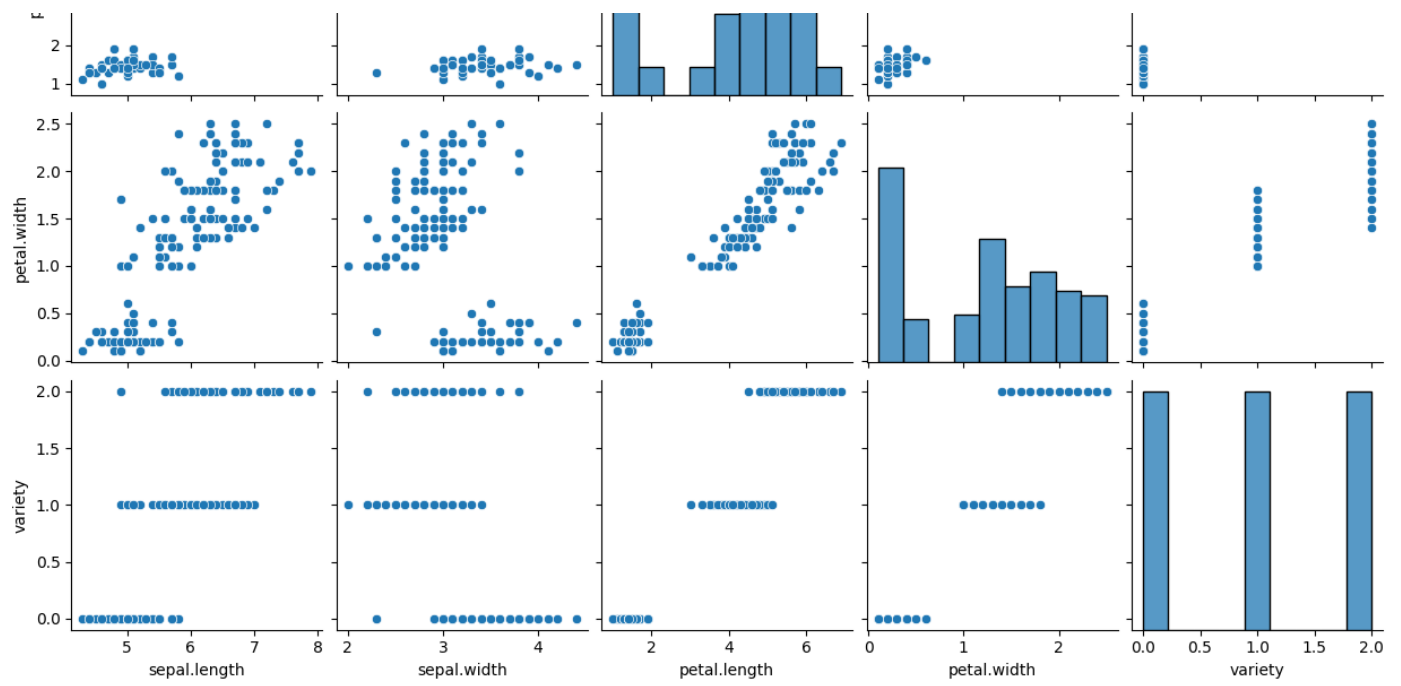
↔ <Axes: xlabel='sepal.length', ylabel='petal.length'>




```
seaborn.pairplot(iris)
```

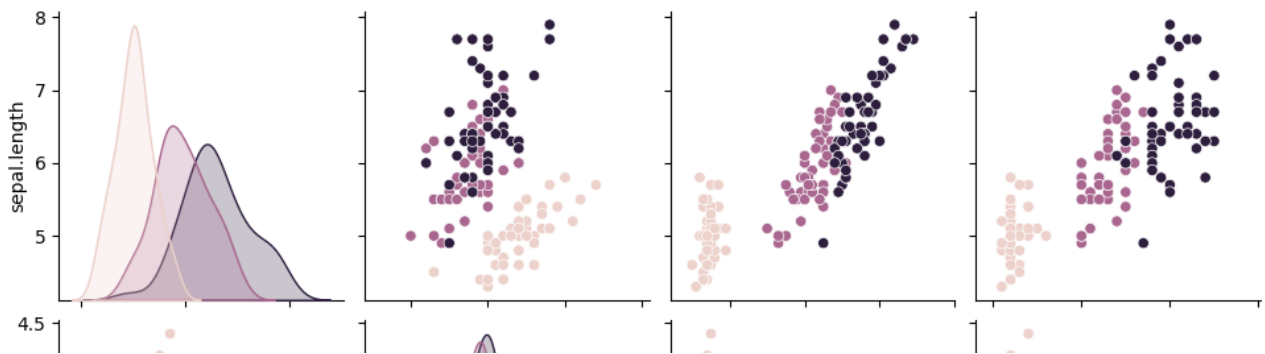
↔ <seaborn.axisgrid.PairGrid at 0x7f9d2b4ffb90>

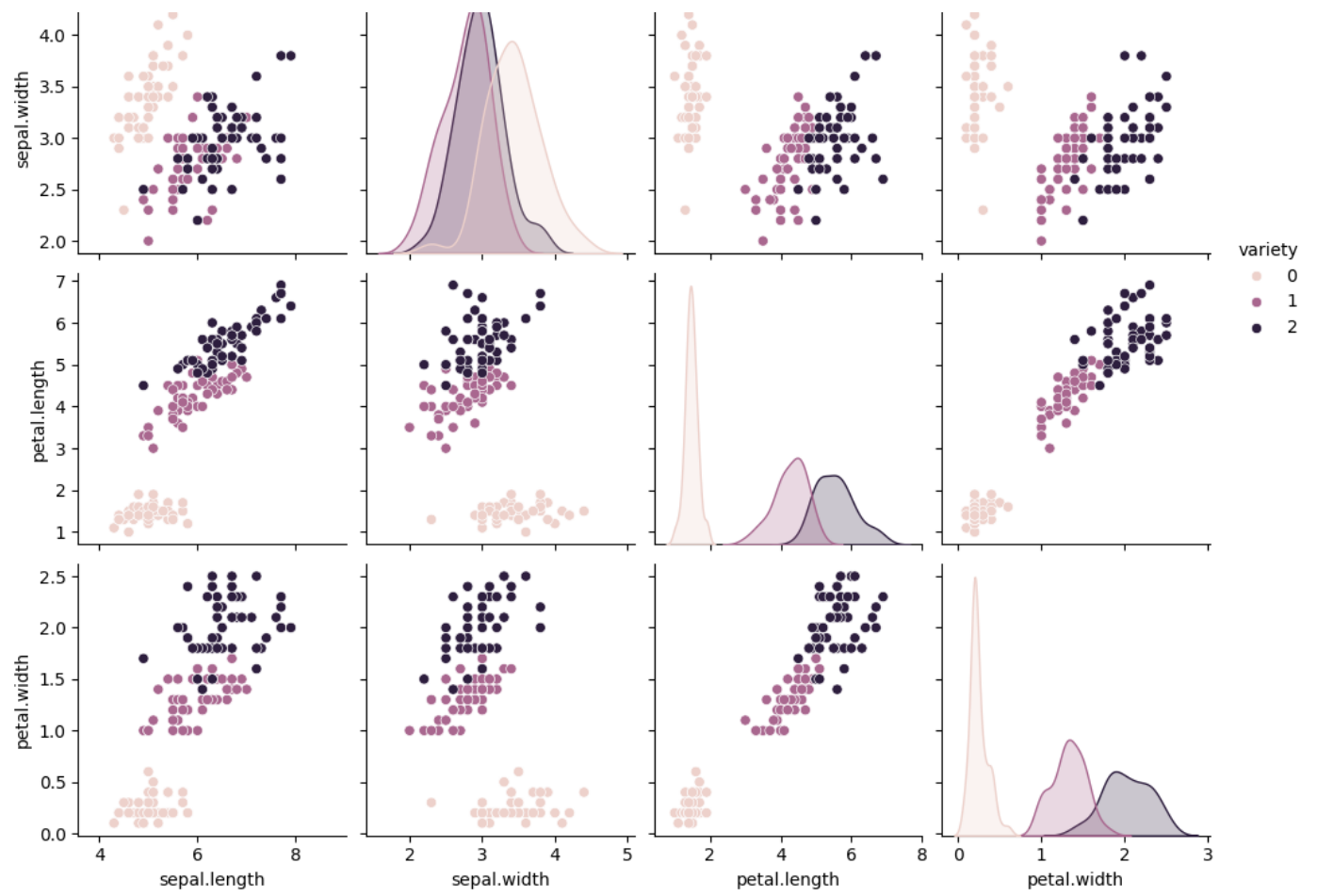





```
seaborn.pairplot(iris, hue="variety")
```

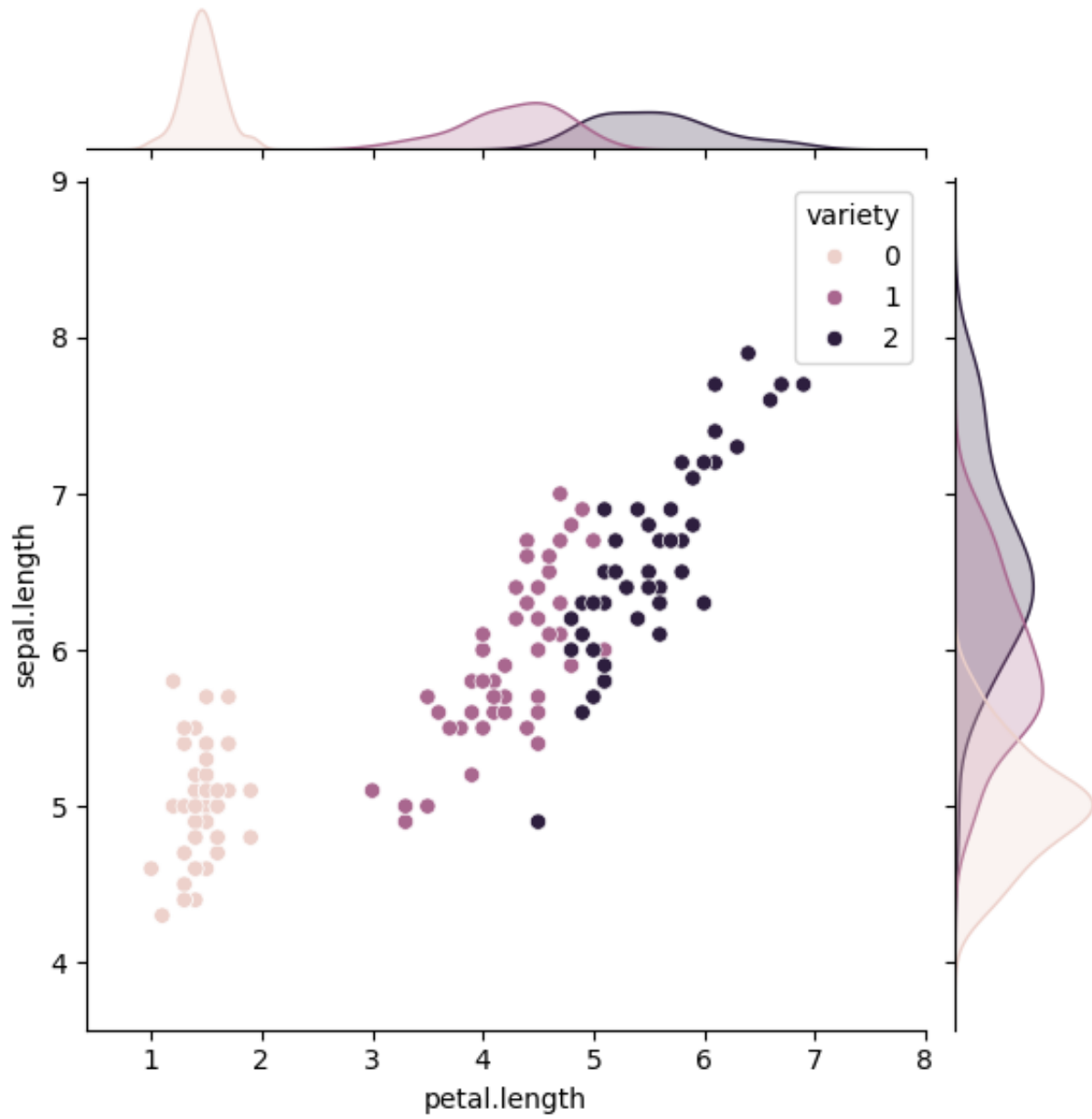
 <seaborn.axisgrid.PairGrid at 0x7f9d2865c710>






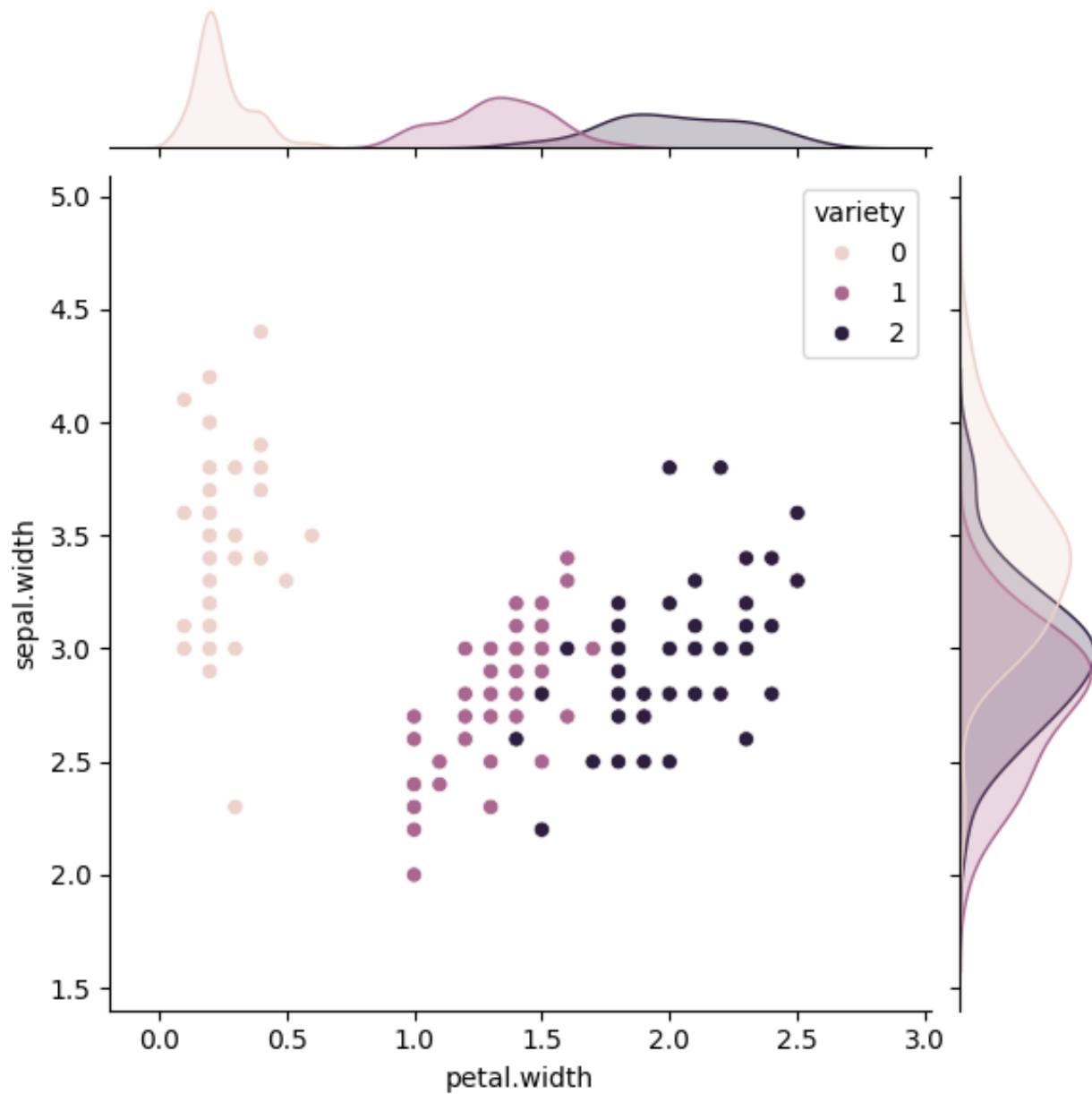
```
seaborn.jointplot(iris, x="petal.length", y="sepal.length", hue="variety")
```

 <seaborn.axisgrid.JointGrid at 0x7f9d250674d0>




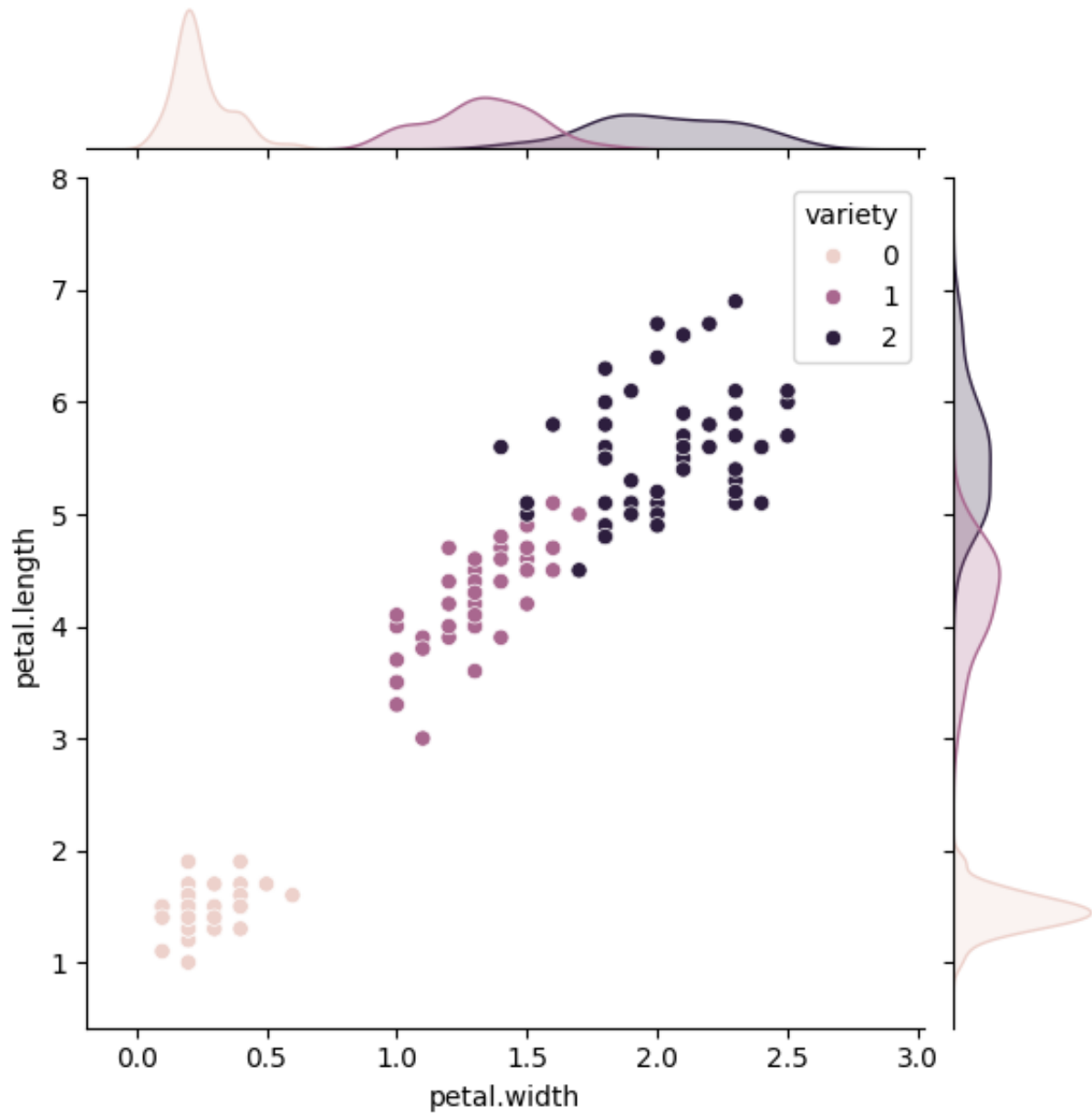
```
seaborn.jointplot(iris, x="petal.width", y="sepal.width", hue="variety")
```

 <seaborn.axisgrid.JointGrid at 0x7f9d25002b10>




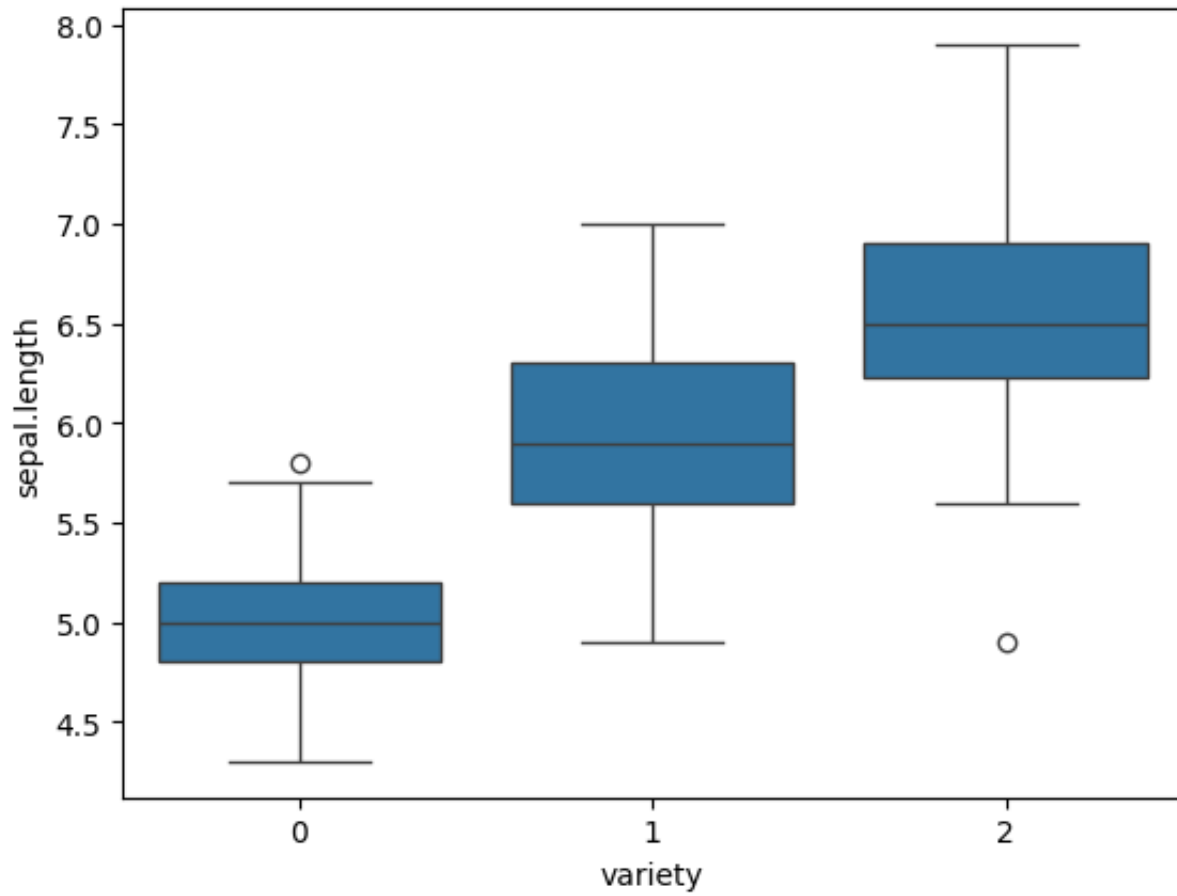
```
seaborn.jointplot(iris, x="petal.width", y="petal.length", hue="variety")
```

 <seaborn.axisgrid.JointGrid at 0x7f9d24f07050>




```
seaborn.boxplot(iris, x="variety", y="sepal.length")
```

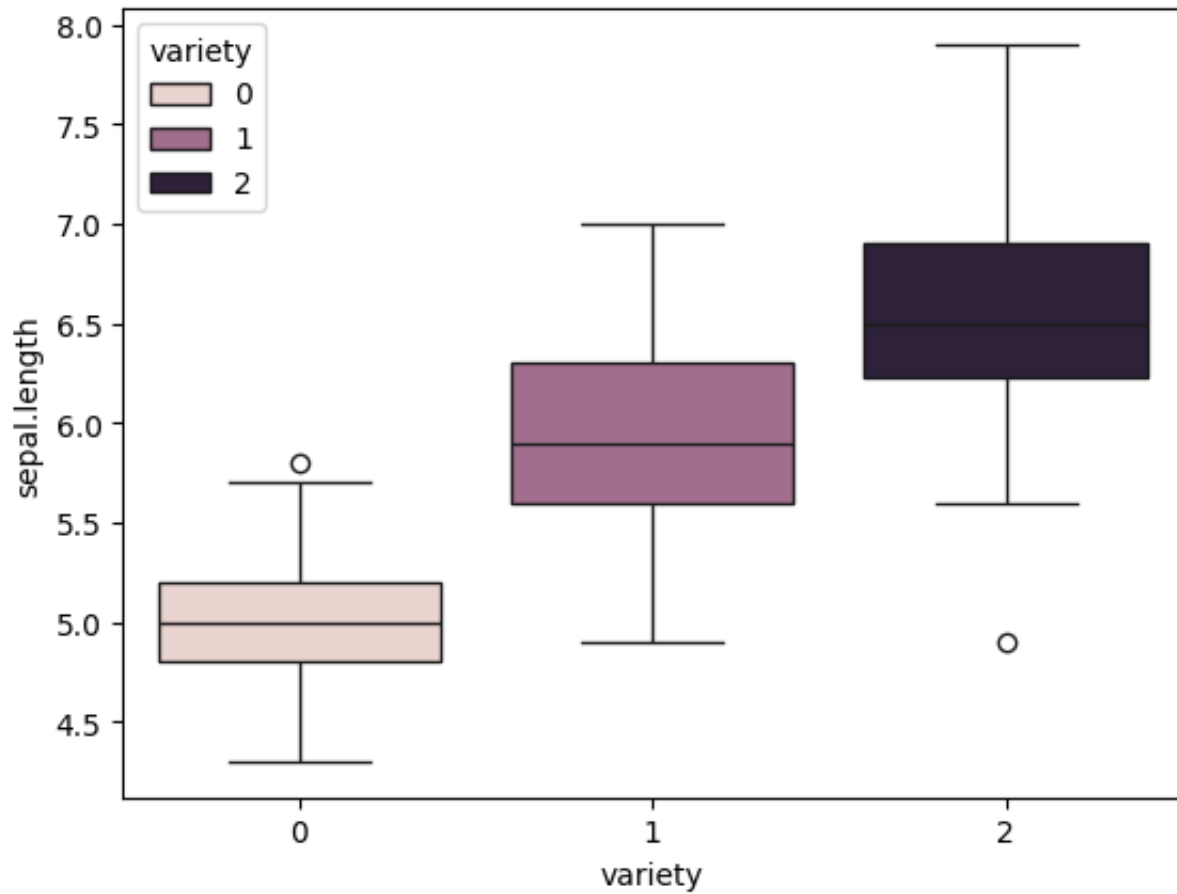
 <Axes: xlabel='variety', ylabel='sepal.length'>






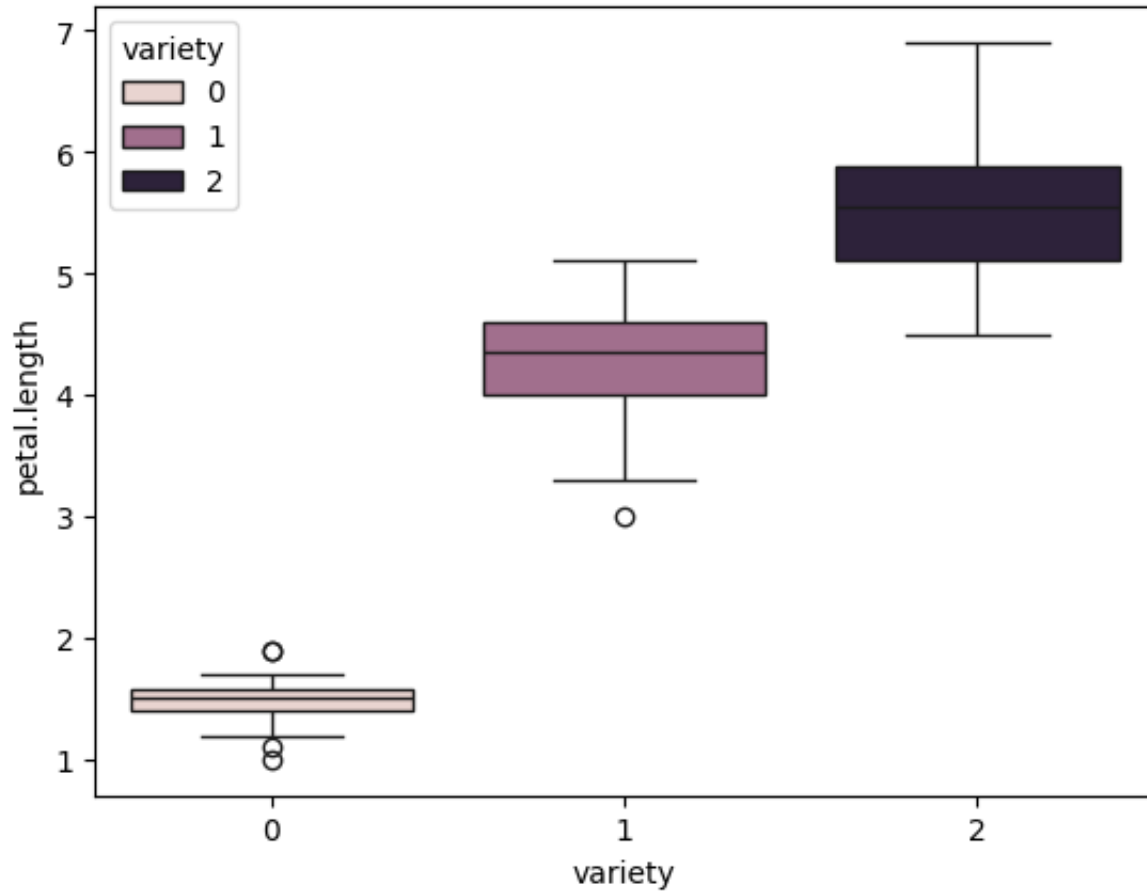
```
seaborn.boxplot(iris, x="variety", y="sepal.length", hue="variety")
```

 <Axes: xlabel='variety', ylabel='sepal.length'>




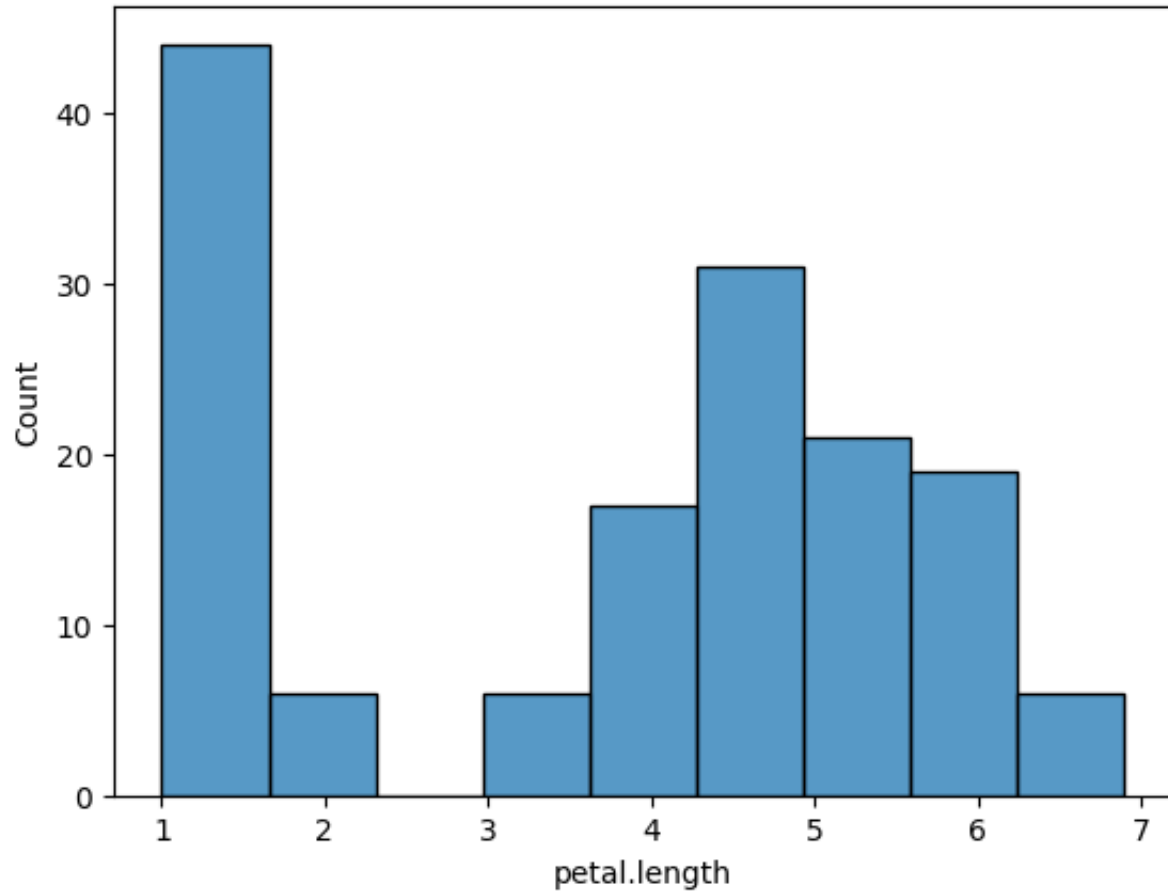
```
seaborn.boxplot(iris, x="variety", y="petal.length", hue="variety")
```

 <Axes: xlabel='variety', ylabel='petal.length'>




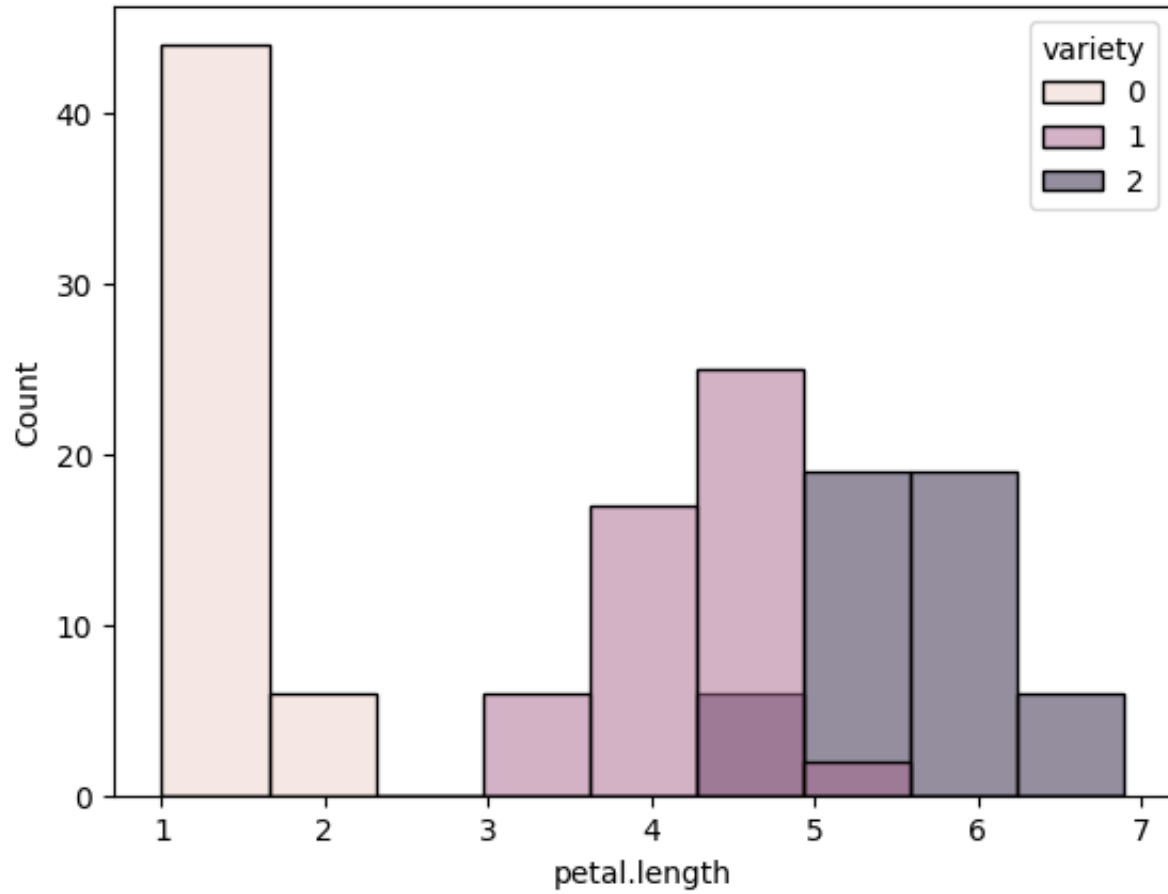
```
seaborn.histplot(iris, x="petal.length")
```

 <Axes: xlabel='petal.length', ylabel='Count'>




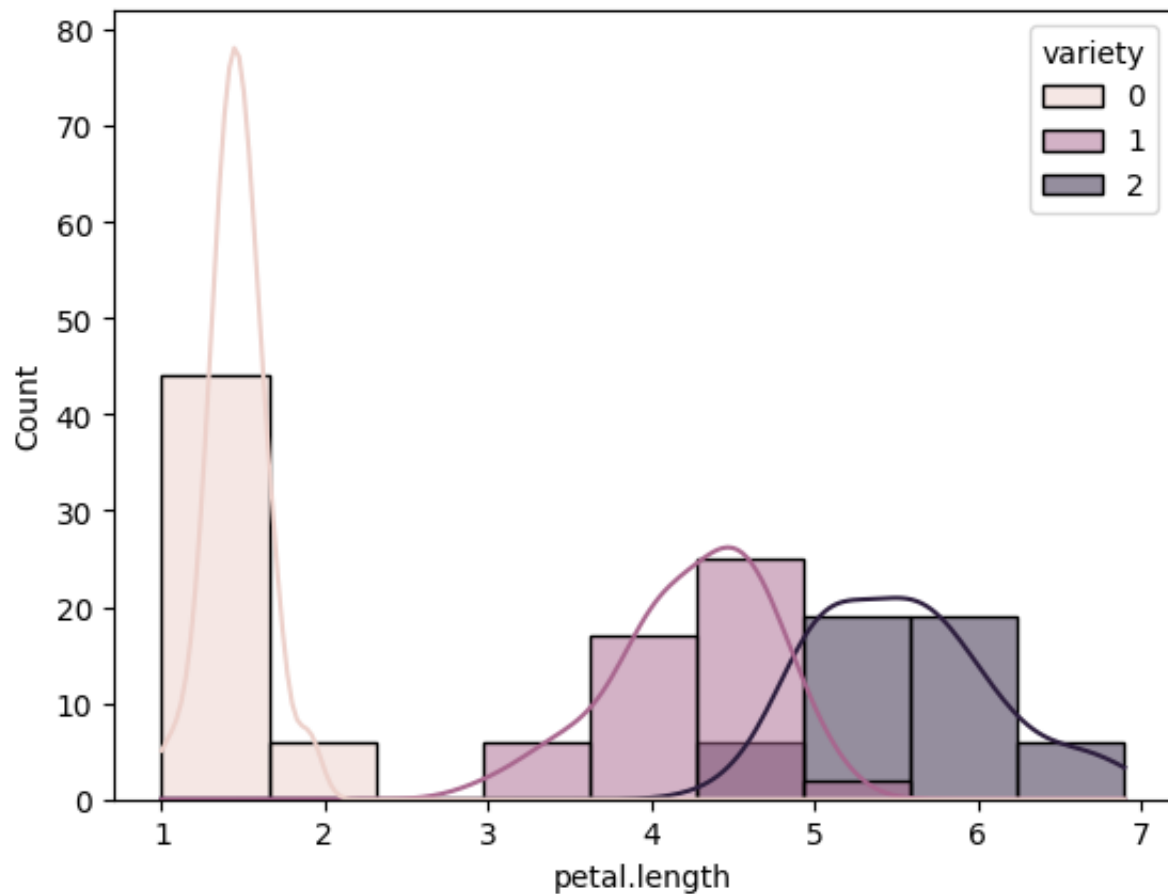
```
seaborn.histplot(iris, x="petal.length", hue="variety")
```

 <Axes: xlabel='petal.length', ylabel='Count'>



```
seaborn.histplot(iris, x="petal.length", hue="variety", kde=True)
```


 <Axes: xlabel='petal.length', ylabel='Count'>

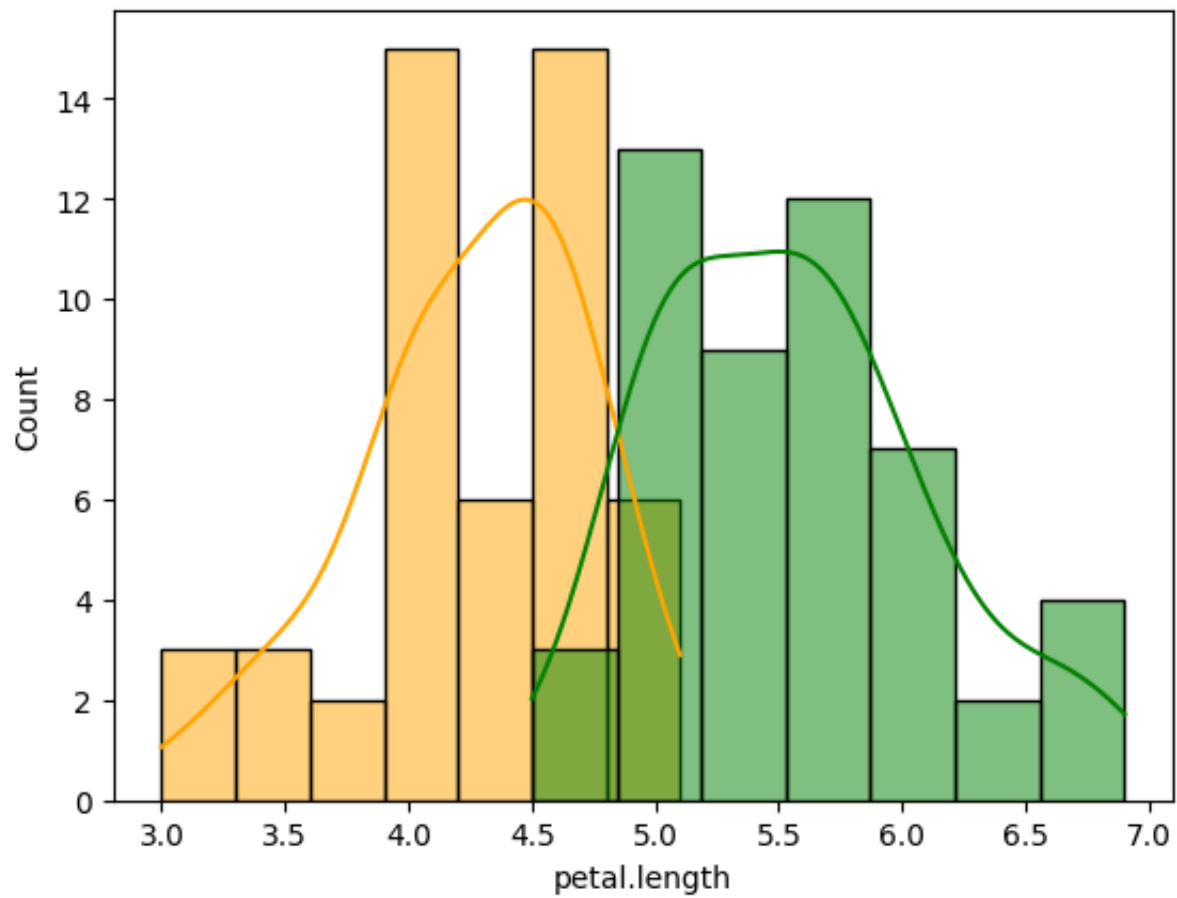


```
petalLengths = iris["petal.length"]  
classes = iris["variety"]
```

```
x1 = petalLengths[classes == 1]  
x2 = petalLengths[classes == 2]
```

```
seaborn.histplot(x1, color="orange", kde=True)  
seaborn.histplot(x2, color="green", kde=True)
```

 <Axes: xlabel='petal.length', ylabel='Count'>



## ✓ Prueba T-Test

Determina si dos muestras independientes tienen medias similares.

Esto permite determinar si al separar dos muestras para dos categorías distintas, su comportamiento del valor medio será similar o distinto.

- **Medias similares** - La categoría con la que se separaron las muestras no explica una diferencia en los datos, por ejemplo, si separamos los pesos de los hombres y los pesos de las mujeres y ambos tienen medias similares, entonces, los hombres pesan igual que las mujeres (caso ficticio)
- **Medias diferentes** - La categoría con la que se separaron las muestras explica que hay una diferencia significativa en los datos, por ejemplo, si separamos las estaturas de los hombres y las estaturas de las mujeres y ambas tienen medias distintas, entonces, los hombres tienen una altura distinta a la de las mujeres (caso real)

---

Prueba de Hipótesis T-Student para dos muestras independientes

### Independent Two-Sample T-Test

$H_0$ : Ambos grupos tienen la misma media  $\mu_1 = \mu_2$

- No hay diferencia significativa entre las medias de ambas muestras

$H_a$ : Los grupos tienen media distinta  $\mu_1 \neq \mu_2$

- Hay diferencia significativa entre las medias de las muestras

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

donde  $\bar{x}_j$  es la media muestral,  $s_j^2$  es la varianza y  $n_j$  el tamaño de la muestra  $j = 1, 2$ .

$$p_{val} = 2 \cdot P(T > |t|)$$

donde  $|t|$  es el estadístico observado y  $T$  sigue una distribución t-student. Es decir, el valor-p se calcula a partir de los datos usando la función de distribución acumulada (CDF) para la distribución t-student (PDF).

Significancia:  $\alpha = 0.05$

```
import scipy.stats as stats

t_stat, p_val = stats.ttest_ind(x1, x2)

print(f"T-Test: {t_stat:.2f}, Valor-P: {p_val:.2f}")
```

➡ T-Test: -12.60, Valor-P: 0.00

Por lo que se rechaza la hipótesis nula de que ambas medias son significativamente iguales, dado que  $p_{val} < \alpha$ . Por lo que las medias son significativamente distintas.

En conclusión, observamos que ambos grupos para la longitud de pétalo tienen medias distintas para las muestras de la clase Versicolor y Virgínica.

## ✓ Prueba Person R

Determina si dos muestras están correlacionadas.

Esto determina si las muestras se comportarán de forma similar:

- **Correlación positiva fuerte** - Cuando el valor es cercano a +1 en la correlación, entonces, cuando el valor de una muestra sube, el otro también lo hace.
- **Correlación negativa fuerte** - Cuando el valor es cercano a -1 en la correlación, entonces, cuando el valor de una muestra baja, el otro sube y viceversa.
- **Correlación nula** - Cuando el valor es cercano a 0 en la correlación, entonces, ambas muestras muestran independencia lineal, lo que significa que podrían provenir de grupos distintos o explicar fenómenos diferentes. Por ejemplo, si se comparan las estaturas de hombres y mujeres, si la correlación es nula, podría ser las estaturas de los hombres se expliquen de una forma (media 1.67) mientras que las estaturas de las mujeres se expliquen de otra forma (media 1.54). En otro ejemplo, si hay correlación nula entre edad y género, podría ser que la edad no explique linealmente al género y viceversa.

Cuando la correlación es fuerte (positiva o negativa), una variable explicará a la otra y viceversa, esto es útil en modelos de regresión lineal.

---

Coeficiente de Correlación de Pearson

### Person R Test

R representa la fuerza y dirección de la relación lineal entre dos variables:

- **+1** - Significa que hay una correlación positiva perfecta (correlación directa)



- **0** - Significa que no hay correlación lineal (correlación nula)
- **-1** - Significa que hay una correlación negativa perfecta (correlación inversa)

El que no hay una correlación lineal (correlación nula), no significa que no haya una correlación no-lineal.

$H_0$ : Ambos grupos no tienen correlación  $R = 0$

- No hay diferencia significativa para determinar que las muestras tienen correlación (lineal)

$H_a$ : Ambos grupos tienen una correlación significativa  $\rho \neq 0$

- Hay diferencia significativa para determinar que las muestras están correlacionadas

$$t = \frac{R\sqrt{n-2}}{\sqrt{1-R^2}}$$

donde  $R$  es el coeficiente de correlación de pearson y  $n = \min(n_1, n_2)$  es el tamaño de ambas muestras (grados de libertad).

Además el coeficiente de correlación se calcula como

$$R = \frac{\sum(x_{1,i}-\bar{x}_1)(x_{2,i}-\bar{x}_2)}{\sqrt{\sum(x_{1,i}-\bar{x}_1)^2 \sum(x_{2,i}-\bar{x}_2)^2}}$$

Y el valor-p está determinado por

$$p_{val} = 2 \cdot P(T > |t|)$$

donde  $|t|$  es el estadístico observado y  $T$  sigue una distribución t-student. Es decir, el valor-p se calcula a partir de los datos usando la función de distribución acumulada (CDF) para la distribución t-student (PDF).

Significancia:  $\alpha = 0.05$

```
r, p_val = stats.pearsonr(x1, x2)
print(f"R: {r:.2f}, Valor-P: {p_val:.2f}")
```

↔ R: -0.09, Valor-P: 0.54

Por lo tanto, el coeficiente de correlación de Pearson  $R$  es cercano a 0, y el valor-p supera el valor crítico de significancia ( $p_{val} > \alpha$ ) por lo que no hay suficiente evidencia para rechazar la hipótesis nula sobre que no hay correlación lineal entre ambas muestras.

En conclusión, observamos que los largos de pétalo no están correlacionados entre las muestras de la clase Versicolor y Virgínica.


## ✓ Problema 6 - Marketing Bancario

<https://hf-mirror.com/datasets/jlh/uci-bank>


```
import pandas

# https://huggingface.co/datasets/jlh/uci-bank/resolve/main/data/train-00000-of-00000.parquet
bank_marketing = pandas.read_parquet("hf://datasets/jlh/uci-bank/data/train-00000-of-00000.parquet")
```

bank\_marketing[["job", "marital", "education", "balance", "housing", "loan"]]



|      | job           | marital | education | balance | housing | loan |
|------|---------------|---------|-----------|---------|---------|------|
| 0    | unemployed    | married | primary   | 1787    | no      | no   |
| 1    | services      | married | secondary | 4789    | yes     | yes  |
| 2    | management    | single  | tertiary  | 1350    | yes     | no   |
| 3    | management    | married | tertiary  | 1476    | yes     | yes  |
| 4    | blue-collar   | married | secondary | 0       | yes     | no   |
| ...  | ...           | ...     | ...       | ...     | ...     | ...  |
| 4516 | services      | married | secondary | -333    | yes     | no   |
| 4517 | self-employed | married | tertiary  | -3313   | yes     | yes  |
| 4518 | technician    | married | secondary | 295     | no      | no   |
| 4519 | blue-collar   | married | secondary | 1137    | no      | no   |
| 4520 | entrepreneur  | single  | tertiary  | 1136    | yes     | yes  |



4521 rows x 6 columns

```
data = bank_marketing[["job", "marital", "education", "balance",\
                      "housing", "loan"]].dropna()
```

```
data.sample(10)
```




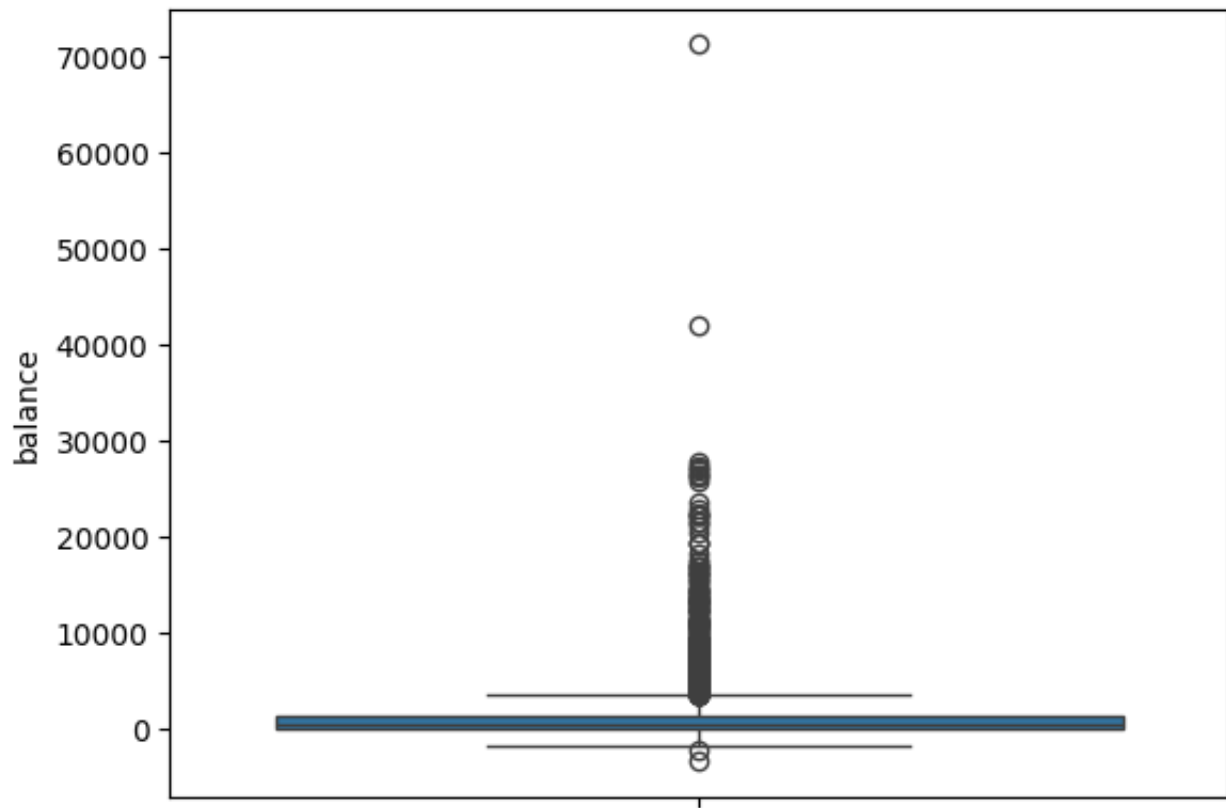
|      | job         | marital  | education | balance | housing | loan |
|------|-------------|----------|-----------|---------|---------|------|
| 2778 | blue-collar | married  | secondary | 602     | yes     | no   |
| 585  | management  | divorced | tertiary  | 38      | yes     | no   |
| 510  | blue-collar | married  | secondary | 121     | yes     | yes  |
| 2272 | technician  | divorced | tertiary  | -1148   | yes     | yes  |
| 4055 | management  | single   | tertiary  | 1673    | no      | no   |
| 3851 | admin.      | single   | secondary | 802     | no      | no   |
| 1468 | management  | married  | tertiary  | 0       | yes     | no   |
| 40   | management  | single   | tertiary  | 11971   | yes     | no   |
| 1805 | retired     | married  | tertiary  | 15834   | no      | no   |
| 3174 | blue-collar | single   | secondary | 168     | yes     | no   |



```
import seaborn
```

```
seaborn.boxplot(data["balance"])
```

 <Axes: ylabel='balance'>



```

import numpy

a = data["balance"].min()
a_ = numpy.quantile(data["balance"], 0.05)
q1 = numpy.quantile(data["balance"], 0.25)
q2 = numpy.quantile(data["balance"], 0.50)
q3 = numpy.quantile(data["balance"], 0.75)
b_ = numpy.quantile(data["balance"], 0.95)
b = data["balance"].max()

print(f"Min:           ${a:10.2f}")
print(f"Min 5%:        ${a_:10.2f}")
print("-" * 22)
print(f"Q1:              ${q1:10.2f}")
print(f"Q2:              ${q2:10.2f}")
print(f"Q3:              ${q3:10.2f}")
print("-" * 22)
print(f"Max 95%:         ${b_:10.2f}")
print(f"Max:             ${b:10.2f}")

```

```

↔
Min:           $ -3313.00
Min 5%:        $ -162.00
-----
Q1:            $    69.00
Q2:            $   444.00
Q3:            $  1480.00
-----
Max 95%:       $   6102.00
Max:           $  71188.00

```

```
data2 = data[
  (data["balance"] >= -175) &\
  (data["balance"] <= 1412)
]
```


```
data2.sample(10)
```

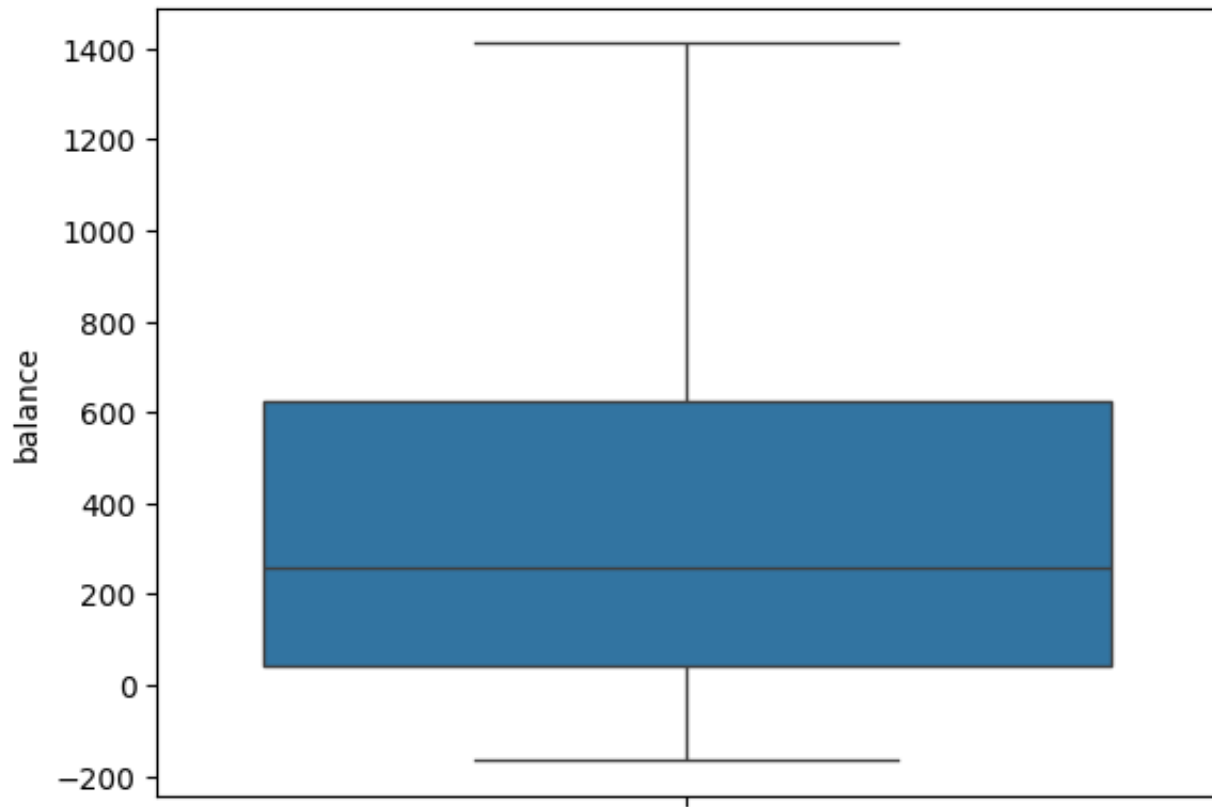


|      | job          | marital | education | balance | housing | loan |
|------|--------------|---------|-----------|---------|---------|------|
| 501  | admin.       | married | secondary | 2       | yes     | yes  |
| 2297 | technician   | married | tertiary  | 118     | no      | no   |
| 2659 | admin.       | married | secondary | 92      | yes     | no   |
| 4318 | blue-collar  | married | secondary | -32     | yes     | no   |
| 3403 | management   | married | tertiary  | 348     | no      | yes  |
| 2257 | blue-collar  | married | secondary | 383     | no      | no   |
| 1281 | entrepreneur | married | primary   | 0       | yes     | no   |
| 2613 | admin.       | married | secondary | 725     | yes     | no   |
| 1789 | blue-collar  | married | secondary | 641     | yes     | no   |
| 1639 | entrepreneur | single  | secondary | 376     | yes     | yes  |




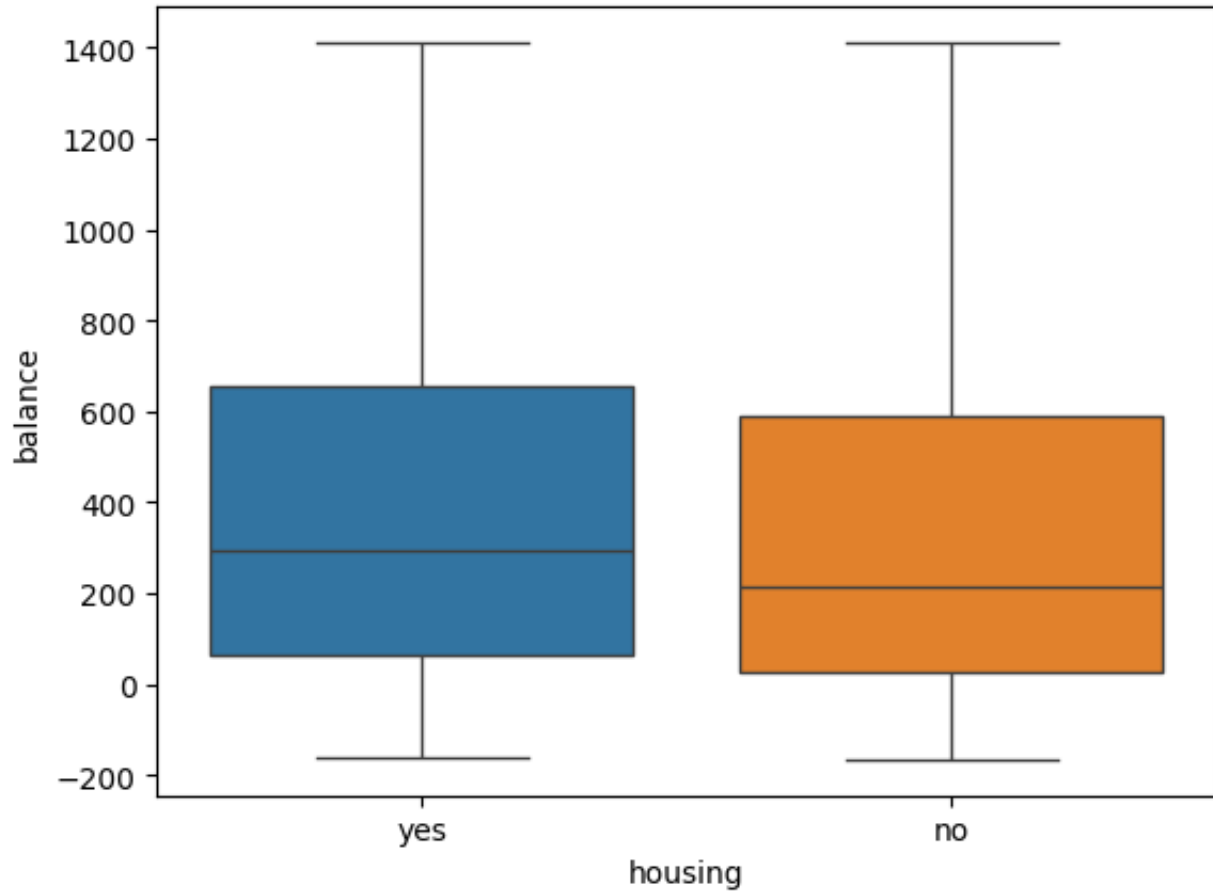
```
seaborn.boxplot(data2["balance"])
```

 <Axes: ylabel='balance'>




```
seaborn.boxplot(data2, x="housing", y="balance", hue="housing")
```

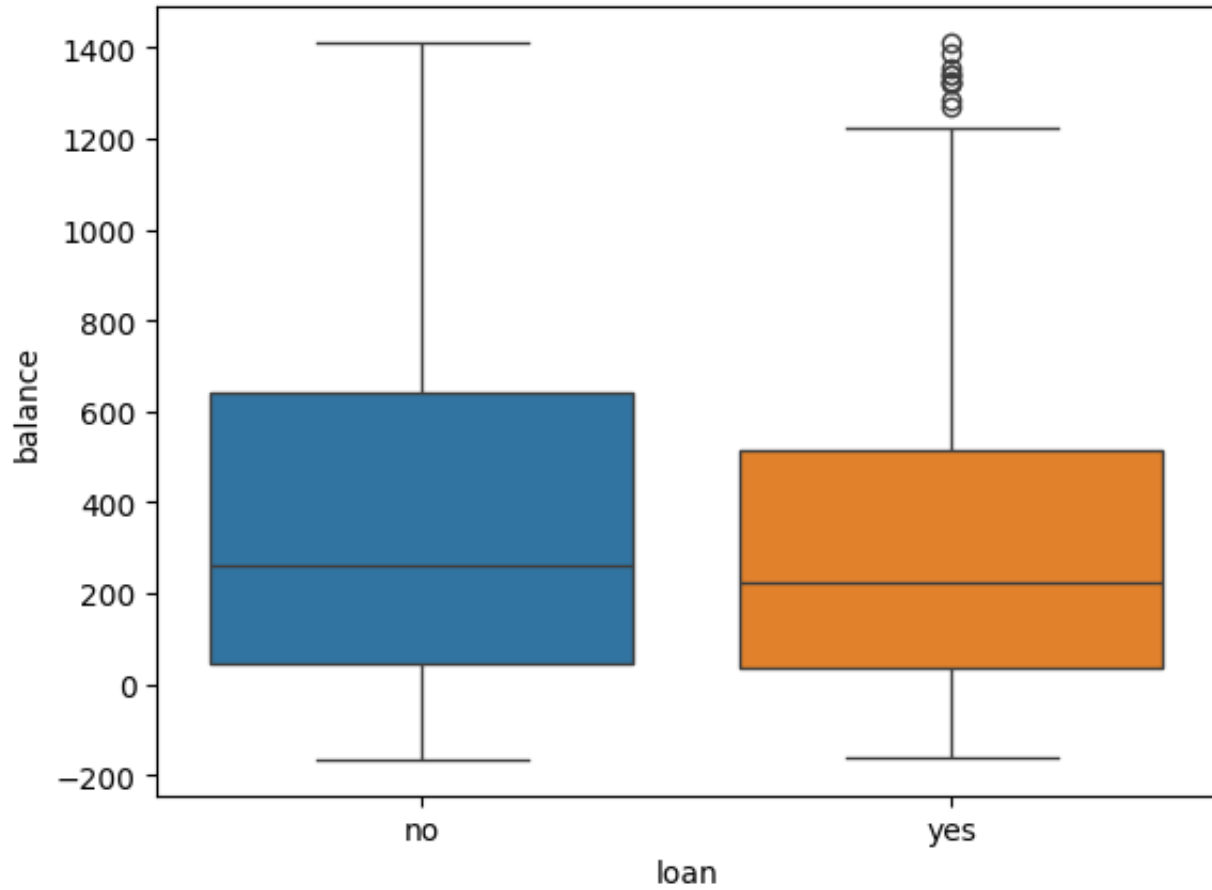
 <Axes: xlabel='housing', ylabel='balance'>






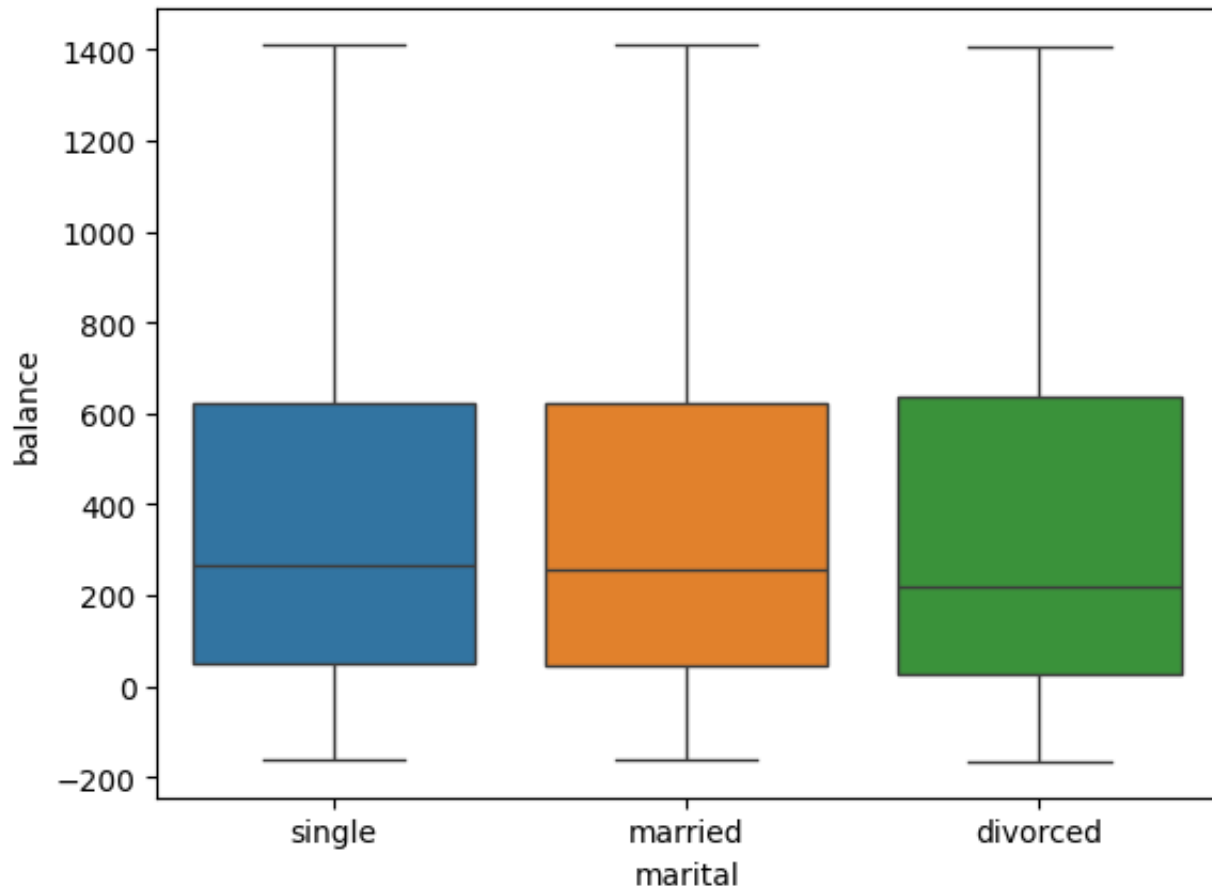
```
seaborn.boxplot(data2, x="loan", y="balance", hue="loan")
```

 <Axes: xlabel='loan', ylabel='balance'>




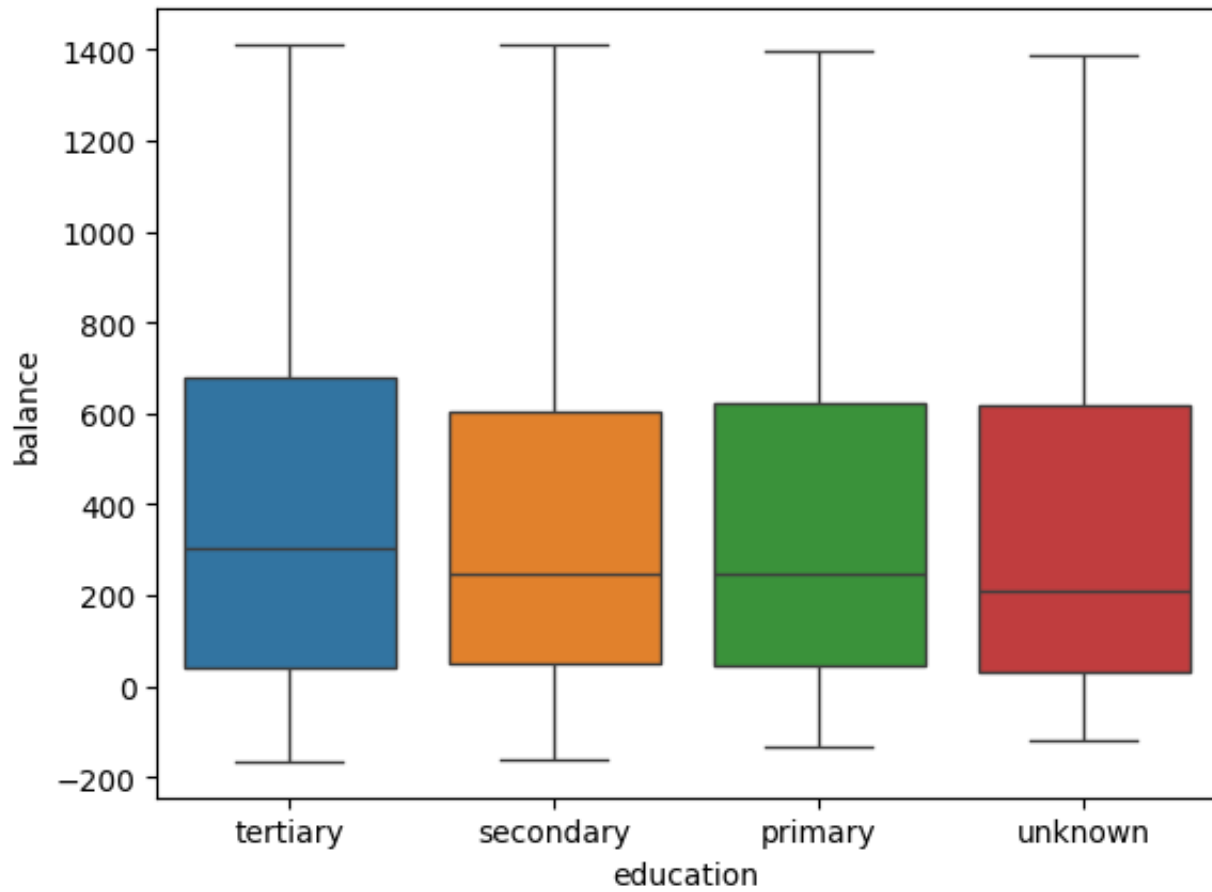
```
seaborn.boxplot(data2, x="marital", y="balance", hue="marital")
```

 <Axes: xlabel='marital', ylabel='balance'>



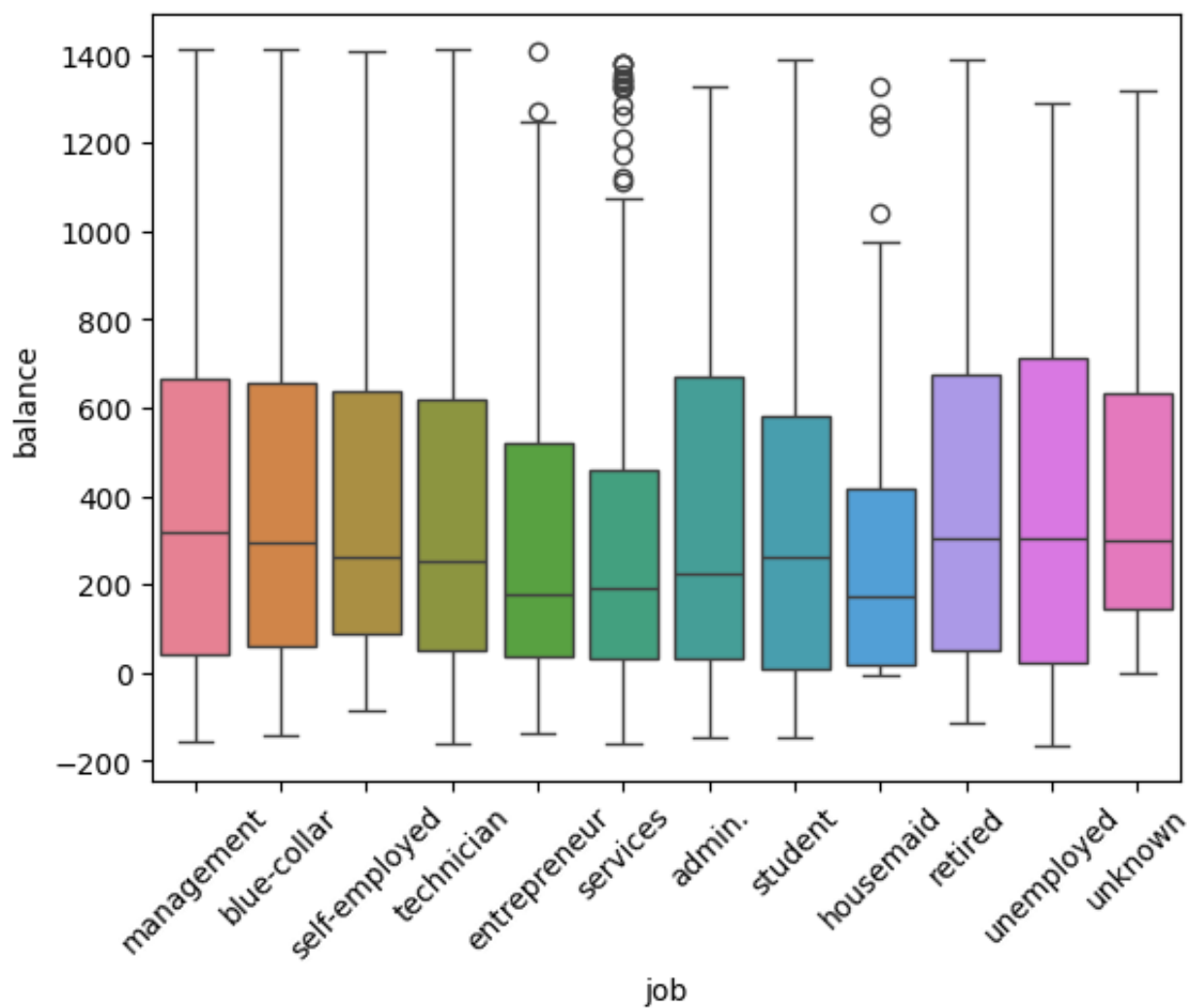
```
seaborn.boxplot(data2, x="education", y="balance", hue="education")
```

 <Axes: xlabel='education', ylabel='balance'>



```
import matplotlib.pyplot as pyplot
```

```
seaborn.boxplot(data2, x="job", y="balance", hue="job")  
pyplot.xticks(rotation=45)  
pyplot.show()
```




✓ Codificar una variable categórica en one-hot

```
import pandas
```

```
bank_marketing_one_hot = pandas.get_dummies(bank_marketing,  
                                             columns=["job", "education", "loan", "marital", "housing"]  
                                             ).dropna().reset_index(drop=True)
```

```
bank_marketing_one_hot
```



| campaign | pdays | previous | ... | education_secondary | education_tertiary | educat: |
|----------|-------|----------|-----|---------------------|--------------------|---------|
| 1        | -1    | 0        | ... | False               | False              |         |
| 1        | 339   | 4        | ... | True                | False              |         |
| 1        | 330   | 1        | ... | False               | True               |         |
| 4        | -1    | 0        | ... | False               | True               |         |
| 1        | -1    | 0        | ... | True                | False              |         |
| ...      | ...   | ...      | ... | ...                 | ...                | ...     |
| 5        | -1    | 0        | ... | True                | False              |         |
| 1        | -1    | 0        | ... | False               | True               |         |
| 11       | -1    | 0        | ... | True                | False              |         |
| 4        | 211   | 3        | ... | True                | False              |         |
| 2        | 249   | 7        | ... | False               | True               |         |

## ✓ Segmentación de datos

```

married_with_loan = bank_marketing_one_hot[
    bank_marketing_one_hot["marital_married"] &
    (bank_marketing_one_hot["loan_yes"])
].copy()
single_with_loan = bank_marketing_one_hot[
    bank_marketing_one_hot["marital_single"] &
    bank_marketing_one_hot["loan_yes"]
].copy()

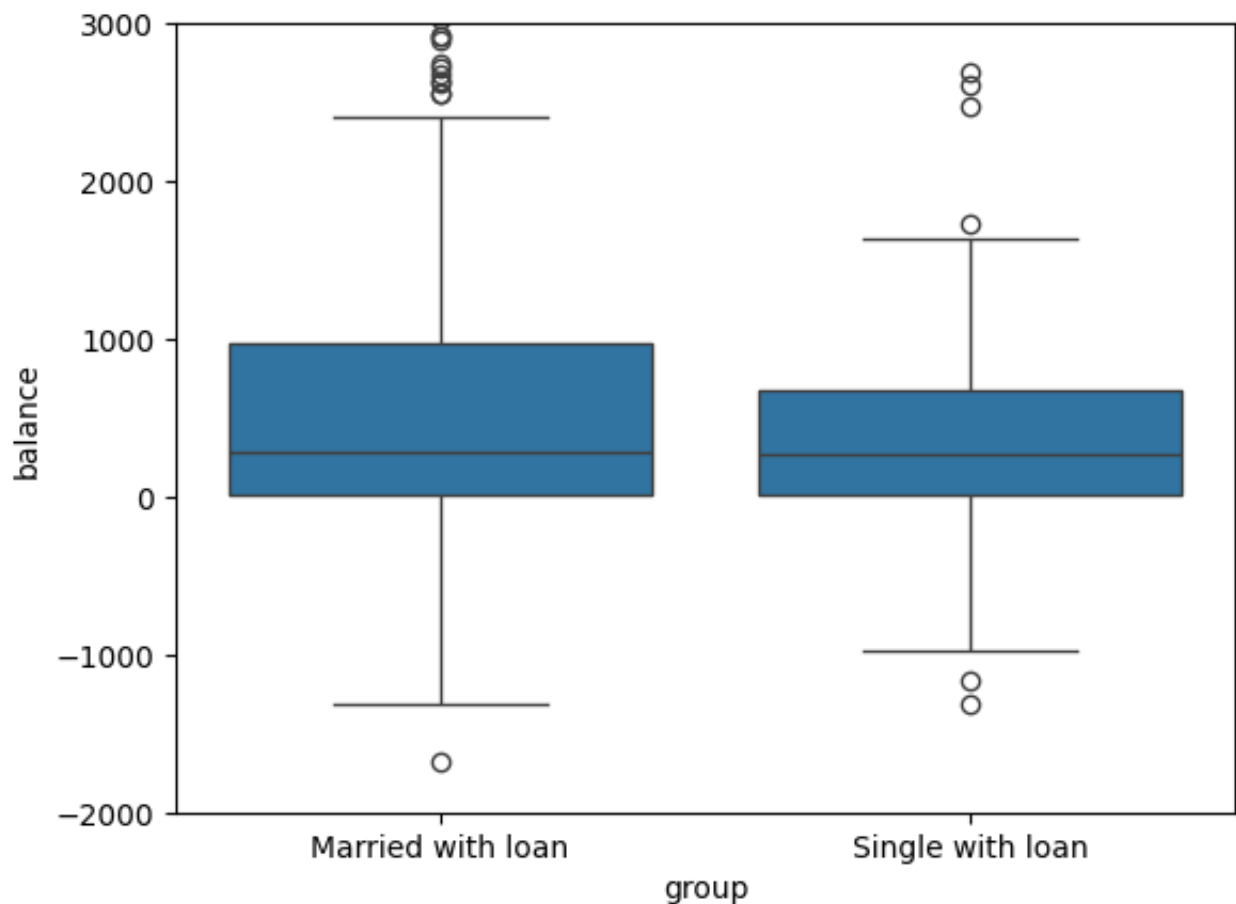
married_with_loan["group"] = "Married with loan"
single_with_loan["group"] = "Single with loan"

combined = pandas.concat([married_with_loan, single_with_loan])

import matplotlib.pyplot as pyplot

seaborn.boxplot(combined, x="group", y="balance")
pyplot.ylim(-2000, 3000)
pyplot.show()

```



bank\_marketing\_one\_hot.columns

```
⇒ Index(['age', 'default', 'balance', 'contact', 'day', 'month', 'duration',  
        'campaign', 'pdays', 'previous', 'poutcome', 'y', 'job_admin.',  
        'job_blue-collar', 'job_entrepreneur', 'job_housemaid',  
        'job_management', 'job_retired', 'job_self-employed', 'job_services',  
        'job_student', 'job_technician', 'job_unemployed', 'job_unknown',  
        'education_primary', 'education_secondary', 'education_tertiary',  
        'education_unknown', 'loan_no', 'loan_yes', 'marital_divorced',  
        'marital_married', 'marital_single', 'housing_no', 'housing_yes'],  
        dtype='object')
```

```

grupo1 = bank_marketing_one_hot[
    bank_marketing_one_hot["education_primary"]
].copy()
grupo2 = bank_marketing_one_hot[
    bank_marketing_one_hot["education_tertiary"]
].copy()

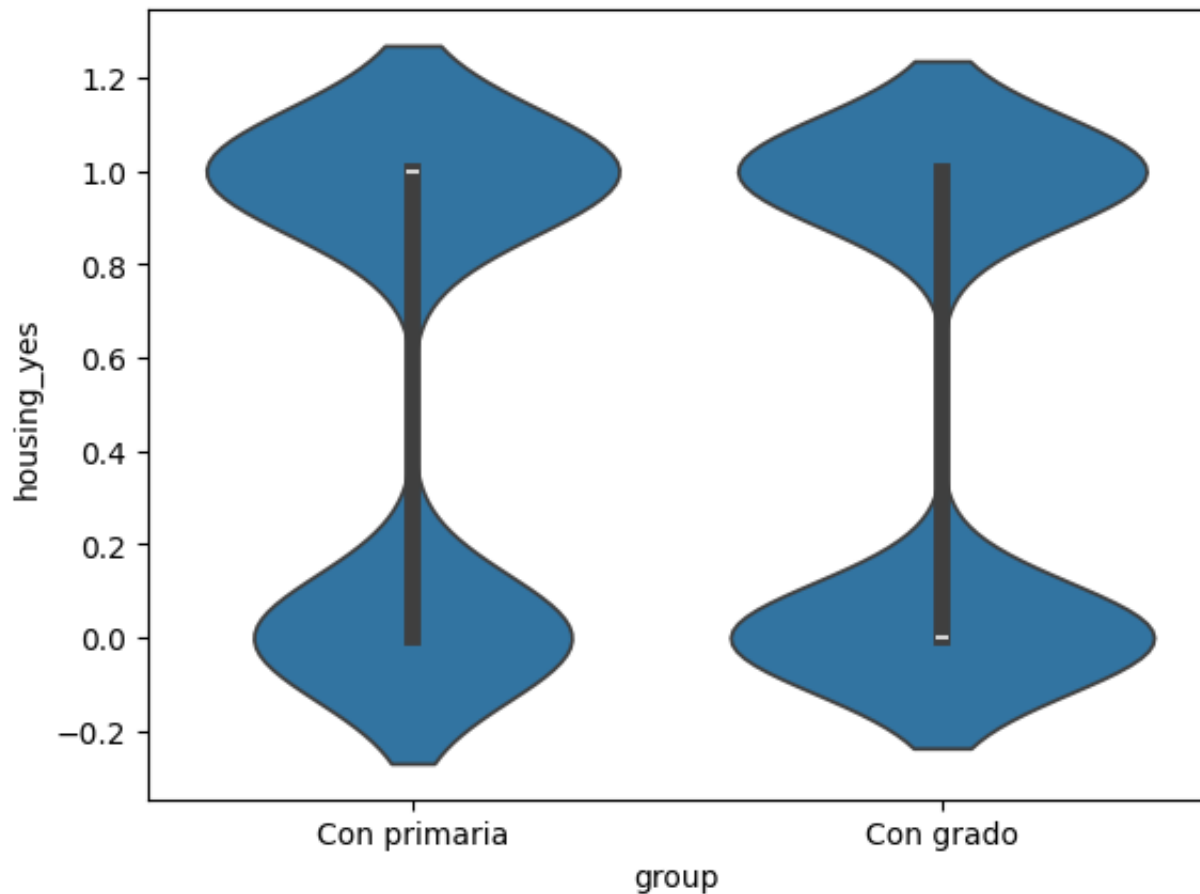
grupo1["group"] = "Con primaria"
grupo2["group"] = "Con grado"

combinado = pandas.concat([grupo1, grupo2])

import matplotlib.pyplot as pyplot

seaborn.violinplot(combinado, x="group", y="housing_yes")
pyplot.show()

```





```
reporte = combinado.groupby("group")["housing_yes"].mean()
```

```
reporte
```



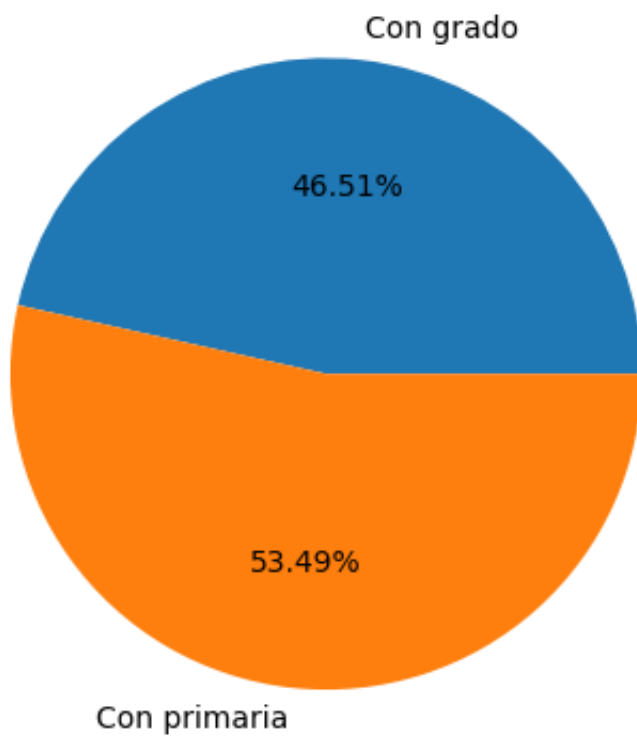
**housing\_yes**

**group**

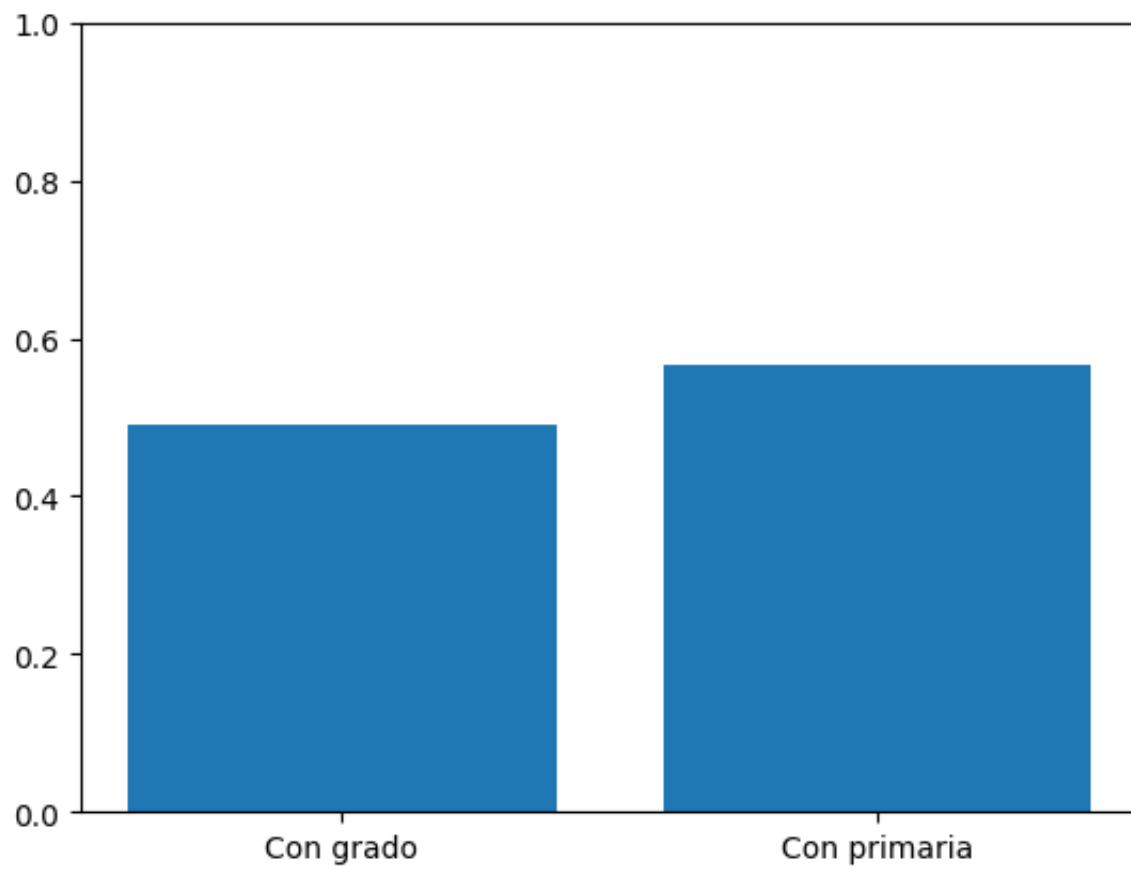
|                     |          |
|---------------------|----------|
| <b>Con grado</b>    | 0.491111 |
| <b>Con primaria</b> | 0.564897 |

**dtype:** float64

```
pyplot.pie(reporte, labels=reporte.index, autopct="%.2f%%")  
pyplot.show()
```



```
pyplot.bar(reporte.index, reporte)  
pyplot.ylim(0, 1)  
pyplot.show()
```



```
grupo1 = bank_marketing_one_hot[
    bank_marketing_one_hot["job_unemployed"]
].copy()
grupo2 = bank_marketing_one_hot[
    bank_marketing_one_hot["job_retired"]
].copy()

grupo1["group"] = "Jubilado"
grupo2["group"] = "Desempleado"

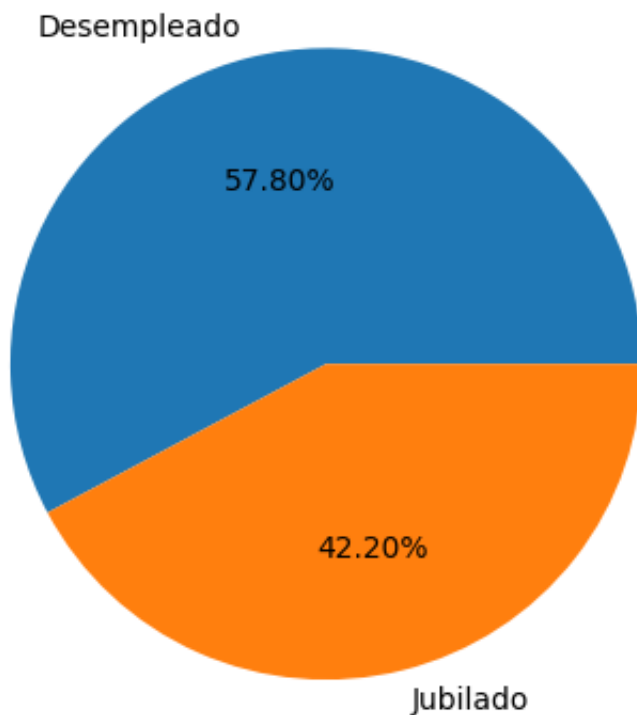
combinado = pandas.concat([grupo1, grupo2])

reporte = combinado.groupby("group")["loan_yes"].mean()

pyplot.pie(reporte, labels=reporte.index, autopct="%.2f%%")
pyplot.title("Tiene un préstamo")
pyplot.show()
```



### Tiene un préstamo



```

grupo1 = bank_marketing_one_hot[
    bank_marketing_one_hot["job_unemployed"]
].copy()
grupo2 = bank_marketing_one_hot[
    bank_marketing_one_hot["job_retired"]
].copy()

grupo1["group"] = "Jubilado"
grupo2["group"] = "Desempleado"

combinado = pandas.concat([grupo1, grupo2])

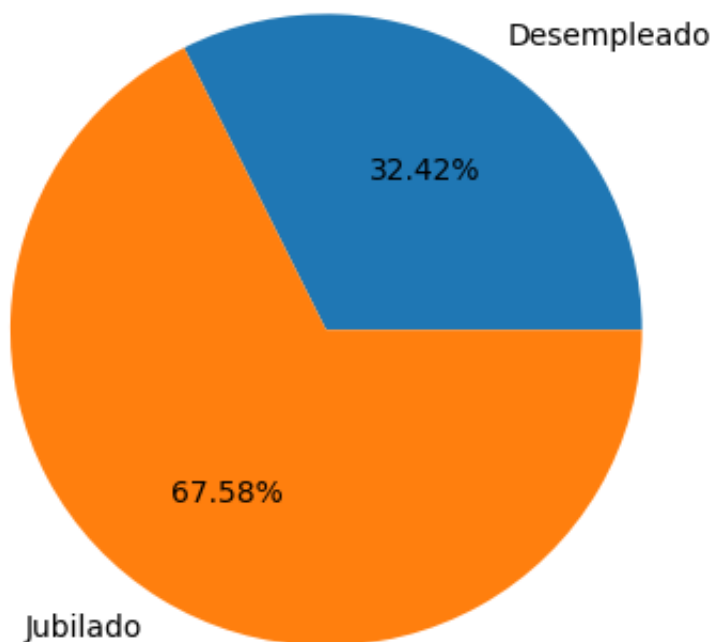
reporte = combinado.groupby("group")["housing_yes"].mean()

pyplot.pie(reporte, labels=reporte.index, autopct="%.2f%%")
pyplot.title("Tiene un crédito de vivienda")
pyplot.show()

```



## Tiene un crédito de vivienda



## Visualización de datos

Ahora trabajemos con la visualización individual nuevamente

Tipos de variables:

- **Cuantitativa (numérica)**

- Continua

- Actotada (en un rango)
    - No acotada (sin rango)
    - Positiva / Negativa
    - Probabilística

- Discreta

- Finita
    - Infinita
    - Positiva / Negativa
    - Binaria (Bernoulli)

- **Cualitativa (categórica)**

- Categórica

- Nominal (sin orden)
    - Ordinal (ordenada)

- Binaria

Tipos de datos:

- **Objetivos (reales)**
- **Subjetivos (ficticios / simulados / nulos)**

Variable aleatoria (numérica)

- Discreta / Continua
- Distribución (Uniforme, Normal, Binomial, Poisson, Gamma)
- Estadísticos

Eje de datos

- Valores numéricos: Discretos / Continuos
- Valores categóricos: Nominales / Ordinales

Gráficas:

- Histograma - Distribución 1 Eje Numérico ~ 1 Eje Categórico
- Violín - Distribución 1 Eje Categórico
- Barras - Distribución 1 Eje Categórico

- Pastel - Distribución 1 Eje Categórico
- Boxplot - Estadísticos 1 Eje Numérico ~ 1 Eje Categórico
- Joinplot - Distribución 2 Ejes Numéricos ~ 1 Eje Categórico (Segmentación)
- Pairplot - Distribución N Ejes Numéricos ~ 1 Eje Categórico (Segmentación)
- Heatmap - Matriz
- Imshow - Matriz Imagen