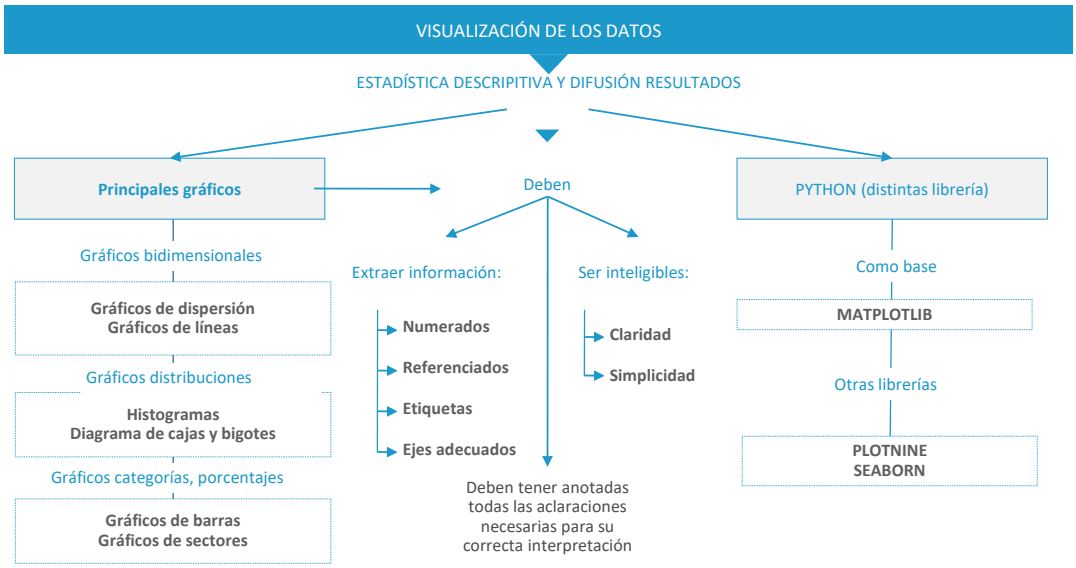
Técnicas Multivariantes

Visualización de los datos

Índice

Esquema



Ideas clave

10.1 Introducción y objetivos

En este tema se va a describir la importancia de una correcta visualización de los datos. Por un lado, la visualización de los datos cobra una gran relevancia como parte del análisis descriptivo. Por otro lado, sirve para poder representar de una manera sencilla los resultados obtenidos. Dependiendo del objetivo, pueden variar las técnicas y métodos empleados para la visualización de los datos aunque en ámbos casos existen unas recomendaciones comunes.

Además, se van a detallar los distintos modos en los que se pueden representar los datos según su naturaleza. Posteriormente se tratarán algunos de los problemas más frecuentes que nos podemos encontrar a la hora de acometer la representación de la visualización de los datos. Por último, se presentarán las instrucciones concretas que se pueden emplear para realizar los distintos gráficos en las librerías más utilizadas en *python*.

10.2 Visualización como análisis descriptivo

Uno de los papeles fundamentales de la visualización de los datos es el propio análisis descriptivo de los mismos. Mediante una correcta visualización de los datos se va a adquirir un mayor conocimiento de los mismos. Por lo tanto, se puede decir que la visualización de los datos forma parte del análisis estadístico descriptivo y es una de las primeras cosas que se deberían hacer a la hora de estudiar un nuevo conjunto de datos. Además, visualizar adecuadamente los datos va a proporcionar mucha más información

que si se trabajase con los datos en crudo, ya que el ser humano no tiene capacidades cognitivas para poder sacar conclusiones a partir de ellos. Una demostración sencilla de este hecho se puede comprobar a través del cuarteto de Ascombe. En 1973, F. J. Anscombe (profesor del departamento de Estadística de la Universidad de Yale) publicó un conjunto de datos compuesto por cuatro parejas de datos X e Y, cada una consistente de 11 observaciones y que compartían las mismas propiedades estadísticas. En la Tabla 1 se muestran este conjunto de datos.

Tabla 1: Cuarteto de Anscombe (Valores)

Cuarteto de Anscombe							
I		II		III		IV	
X	Y	X	Y	X	Y	X	Y
10	8.04	10	9.14	10	7.46	8	6.58
8	6.95	8	8.14	8	6.77	8	5.76
13	7.58	13	8.74	13	12.74	8	7.71
9	8.81	9	8.77	9	7.11	8	8.84
11	8.33	11	9.26	11	7.81	8	8.47
14	9.96	14	8.10	14	8.84	8	7.04
6	7.24	6	6.13	6	6.08	8	5.25
4	4.26	4	3.10	4	5.39	19	12.50
12	10.84	12	9.13	12	8.15	8	5.56
7	4.82	7	7.26	7	6.42	8	7.91
5	5.68	5	4.74	5	5.73	8	6.89

Como ya se ha comentado, si observamos los números en crudo es difícil extraer información sobre los datos. Si no pensaramos en visualizar los datos y únicamente acudiesemos a los estadísticos descriptivos típicos obtendríamos los resultados que se muestran en la Tabla 2.

Tabla 2: Cuarteto de Anscombe (Estadísticos)

Cuarteto de Anscombe		
Propiedad	**Valor**	**Precisión**
Media de x	9	Exacto
Varianza muestral de x: σ^2	11	Exacto
Media de y	7.50	Hasta el segundo decimal
Varianza muestral de y: σ^2	4.125	±0.003
Correlación entre x e y	0.816	Hasta el tercer decimal
Recta de regresión lineal	y = 3.00 + 0.50x	Hasta el segundo decimal
Coeficiente de determinación R^2	0.67	Hasta el segundo decimal

Atendiendo a estos estadísticos podríamos pensar que los datos son en esencia iguales, ya que las medias de x e y son las mismas, las varianzas también e incluso la correlación entre x e y también tienen los mismos valores. La sorpresa llega cuando se comprueba de forma visual cada uno de los 4 conjuntos de datos del cuarteto de Anscombe, tal y como se muestra en la Figura 1.

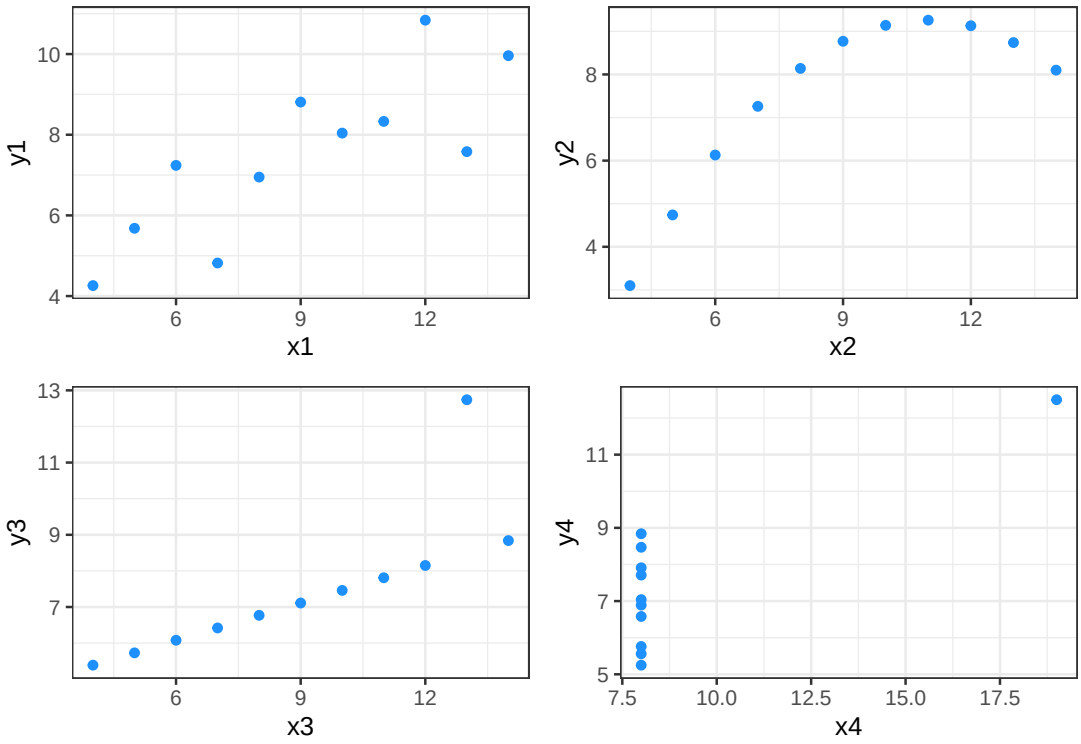


Figura 1: Cuarteto de Anscombe

Una vez se han representado los datos se observa que los 4 conjuntos, aún teniendo los mismos estadísticos descriptivos, son muy diferentes entre sí:

- En el primer conjunto de datos existe una correlación lineal moderada entre x e y.
- En el segundo conjunto de datos la correlación entre x e y es no lineal.
- En el tercer conjunto de datos existe una correlación lineal muy fuerte entre x e y exceptuando uno de los puntos que tiene un comportamiento anómalo.
- Por último, en el cuarto conjunto todos los datos tienen el mismo valor de x excepto un valor alojado que produce un fuerte apalancamiento de los estadísticos descriptivos.

El cuarteto de Anscombe es una exageración de unos conjuntos de datos para que se obtengan los mismos estadísticos descriptivos ante conjuntos de datos muy diferentes,

pero sirve para resaltar la gran importancia que tiene la inspección visual en el análisis de los datos.

10.3 Visualización para la difusión de los resultados

La otra gran utilidad de la visualización de los datos estriba en la difusión de los resultados. Es posible que se hayan analizado muy bien unos datos, y que la persona que los ha estado analizando los conozca a la perfección, pero seguramente deberán compartir la información que han extraído de los mismos con otras personas, como por ejemplo:

- En la publicación de resultados científicos.
- En la divulgación de resultados en prensa.
- Para realizar un informe para otro sector de la empresa o para un superior.
- Presentación de resultados a un cliente.
- Exposición de los resultados de un trabajo de investigación ante un tribunal de un trabajo fin de estudios.

En todos estos casos, la información que se puede transmitir no es la misma si los resultados únicamente se comentan en un texto, o como mucho se muestran en una tabla, que si se plasman correctamente en un gráfico. En un gráfico podemos ser capaces de condensar mucha información y de mostrar patrones, tendencias y hallazgos de una forma rápida y directa.

Para realizar correctamente los gráficos en los que van a basarse la visualización de los datos enfocados a terceras personas es necesario seguir las siguientes pautas:

- Es importante que el usuario o cliente encuentre el gráfico sencillo e intuitivo, además de que pueda comprender la información reflejada en la visualización.

- Se deben seguir guías de estilo. Además, todos los gráficos de un mismo documento deben ser consistentes y mantener una coherencia entre sí. Por ejemplo, emplear tamaños similares, utilizar mismos tipos de fuentes, usar mismos colores de los fondos, etc.

- En la medida de lo posible se deben utilizar técnicas de visualización que puedan amplificar la cognición, incrementando la capacidad de procesamiento de los datos y reduciendo la búsqueda de información en el texto.

La visualización utiliza técnicas de codificación que facilitan la comunicación y la comprensión, trasladando representaciones gráficas a representaciones perceptivas. Ya sea para gráficos simples o para representar grandes volúmenes de datos, uno de los requisitos esenciales es que dichos datos deben ser interpretados y entendidos. Por este motivo, es esencial la obtención de esta información desde una perspectiva general a una más detallada. Por ejemplo, se puede mostrar en una primera figura la información más general, y en una segunda figura mostrar un detalle (o un *zoom*) de la primera.

Por otro lado, hay que tener en cuenta que antes de poder representar un conjunto de datos es necesario realizar una limpieza de los datos y un análisis de posibles problemas. Los problemas existentes que se pueden tratar antes de la visualización son:

- Valores anómalos. Ya se ha tratado en temas anteriores como lidiar con ellos.

- Atributos dependientes con valores contradictorios. (por ejemplo, que aparezca en la misma observación una persona que tenga: status de *fumador = nunca fumadora* y *número de cigarrillos al día = 20*)

- Identificadores no únicos. Que dos observaciones tengan el mismo identificador

cuando no estamos tratando con datos longitudinales.

- Valores ausentes. Habría que valorar imputar.

Por lo tanto es imprescindible antes de la representación de los datos/resultados realizar una limpieza de la base de datos empleada. Generalmente, en el proceso de limpiar los datos en bruto de una base de datos se pueden realizar los siguientes pasos:

- Análisis de información. Se necesita analizar la información del conjunto de datos para poder detectar los problemas anteriormente mencionados. Se necesita conocer la estructura de los datos, qué variables están relacionadas entre sí, codificaciones de variables categóricas como numéricas, los rangos de valores posible de las variables y los tipos de datos que contienen (si esperamos un valor numérico que no sea un carácter), etc.

- Verificación. Una vez que se tiene suficiente conocimiento sobre la base de datos, se deben verificar que los datos no presentan incosistencias. En caso de detectar estas inconsistencias, corregirlas.

- Almacenamiento de datos limpios. Una vez que se ha limpiado el conjunto de datos, se puede guardar la base de datos limpia (sin sobreescribir los datos en bruto, por si en el fúturo se tuviese que volver a ellos).

10.4 Principales tipos de datos y tipos de gráficos

Las técnicas de visualización utilizan representaciones gráficas de datos para obtener una mejor comprensión de un problema a partir de los datos utilizados. Los principales tipos de datos a ser visualizados son:

- Datos unidimensionales. Tipos de datos lineares incluyendo código fuente de un

programa, textos, y listas alfabéticas organizadas de una manera secuencial. La utilidad de su visualización depende de su cantidad. Se pueden representar en una línea o eje.

- Datos bidimensionales. Son los datos más representados. Estos tipos de datos en los que se consideran dos dimensiones diferenciadas representados en el espacio también incluyen a los datos planos o mapas incluyendo mapas geográficos o planos de planta. Habitualmente, se emplean para representar en los ejes de abscisas y ordenadas dos variables que pueden tener una relación entre sí.

- Datos tridimensionales. Representaciones de datos con tres dimensiones. Se suelen utilizar para representar objetos del mundo real con volumen. La utilidad de realizar estas representaciones tridimensionales en estos objetos reales es obtener modelos en los que seamos capaces de explorar y manipular elementos en el que la posición y la orientación son un factor importante, con la capacidad de realizar experimentos sobre objetos del mundo real, pudiendo predecir comportamientos a través de simulaciones. Otra aplicación, es representar una variable frente a otras dos. El problema de este tipo de gráficos es que en muchas ocasiones la perspectiva puede ser algo confusa y para una comprensión adecuada es necesario poder cambiar la rotación del gráfico de manera interactiva. Por lo tanto, su utilización para la impresión en papel se ve en cierto modo limitada.

- Datos multidimensionales. Algunos conjuntos de datos consisten en más de tres atributos por lo que no permiten una visualización simple. Para poder representar estos datos es necesario realizar sobre ellos alguna transformación, como por ejemplo un análisis de componentes principales.

- Datos temporales. Datos que utilizan técnicas temporales para su representación. Los datos que son ordenados cronológicamente pueden estar superpuestos (si un dato ocurre en el mismo momento) y los datos tienen un comienzo y un final. Un valor unidimensional temporal se puede representar como si fuera un dato bidimensional.

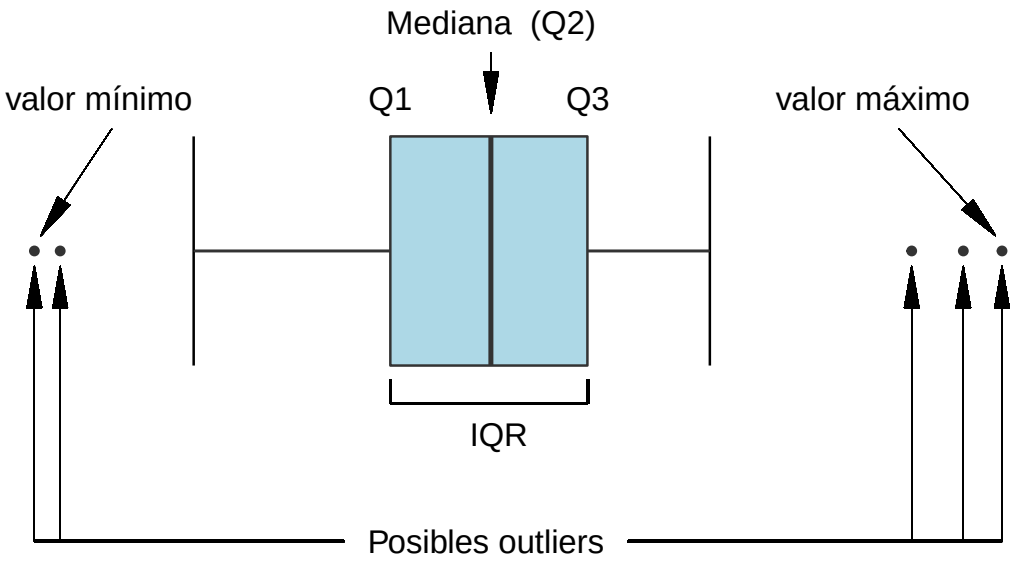
Para la visualización de todos estos tipos de datos existen números gráficos disponibles: desde las gráficas de 2 o 3 dimensiones que utilizan los ejes x-y o x-y-z, hasta técnicas más sofisticadas. Los más habituales son:

- **Gráfico de líneas**. Es un gráfico de dos dimensiones donde cada pareja de valores se une con la siguiente mediante una línea. Se suele emplear en las series temporales.

- **Gráfico de dispersión** o *scatterplot*. Es un gráfico de dos dimensiones donde cada pareja de valores se representa mediante un punto.

- **Histograma**. El histograma representa las frecuencias de una variable discreta (o la discretización de una variable continua) por intervalos, con lo que podemos obtener una representación de la distribución de la variable. A partir del histograma es posible representar la función de densidad de la variable representada en un **diagrama de kernel de densidad**, que consiste en una suavización del histograma.

- **Diagrama de cajas y bigotes** o *boxplot*. Es un gráfico que permite representar algunos de los estadísticos descriptivos claves de una distribución. Está formado por 1 línea y 2 cajas. Los extremos de las cajas representan los percentiles 25 y 75 de la distribución mientras que la línea que las separa marca la mediana. En los extremos de las líneas aparecen los valores correspondientes a los percentiles 0 y 100, exceptuando a los valores que son considerados como outliers que son representados como puntos. Estos valores son los que tienen un valor superior al percentil 75 más 1.5 el rango intercuartílico (también conocido como IQR, y que es igual al percentil 75 menos el percentil 25) o un valor inferior al percentil 25 menos 1.5 el rango intercuartílico. En la figura 2 se muestran los elementos que componen un diagrama de cajas y bigotes.



El extremo del bigote de la izquierda se obtiene como max(min valor, Q1−1.5·IQR), el extremo del bigote de la derecha se obtiene como min(max valor, Q3 + 1.5 · IQR).

Figura 2: Elementos de un diagrama de cajas y bigotes

- **Diagrama de barras**. Representa la frecuencia de diferentes categorías de una variable o diferentes valores que son comparables. También se puede emplear para representar variables discretas que no tengan muchos valores.

- **Diagrama de sectores**. Representa porcentajes de una variable en diferentes categorías. Se utiliza únicamente, si se puede visualizar correctamente. Es decir, si todas las categorías tienen valores distinguibles. Se recomienda no emplearlo si existen muchos valores similares, ya que en ese caso es preferible utilizar un diagrama de barras.

10.5 Gráficos en Python

En Python existen numerosas librerías que permiten la visualización de los datos. En este tema se van a describir algunas de las funciones y de los métodos disponibles para realizar los gráficos más habituales, dejando la puerta abierta a que el alumno

interesado pueda profundizar en alguna de ellas.

Matplotlib

Matplotlib es la librería "de base" que se emplea en Python y con la que se suele comenzar a realizar los gráficos. Tiene un manejo sencillo y es bastante intuitiva. Permite realizar muchos de los gráficos más empleados en el análisis estadístico. El inconveniente es que el formato de algunos de sus gráficos no es muy atractivo y puede resultar algo monótona. En Matplotlib se pueden introducir los valores que se quieren representar a través de listas, o también se pueden utilizar **arrays** de la librería **numpy**, lo cual nos permite obtener mucha flexibilidad.

- **Gráfico de líneas**. El gráfico de líneas, es la opción por defecto que emplea la librería **Matplotlib** en su función plot. Por tanto, para realizar un gráfico de líneas con **Matplotlib** simplemente se debe introducir en la función **plot** la lista de valores que se quieran representar, como ,por ejemplo, en la Figura 3, que se representa el tiempo medio de 10 clases de UNIR en minutos, y mostrarlo mediante la función **show**. Ahora bien, esté gráfico sería claramente deficiente, ya que no tiene nombres en los ejes de coordenadas que hagan referencia a las variables que se están representando. Además, es necesario modificar los números que está marcando el eje (X), ya que al sólo introducir una variable, **Matplotlib** está representando en el eje (X) los índices de la lista. En este ejemplo, el eje horizontal (X) contiene 10 números, del 0 a 9 (los índices de la lista); mientras que el eje vertical contiene los tiempos de las sesiones (Y) medidos en minutos. En un primer momento, la librería escoge una escala que comprende desde 46 hasta 56 minutos. Veamos ahora cómo se puede nombrar estos ejes, cambiar la escala y añadir una rejilla que permita observar más fácilmente los resultados.

```
# cargar librerias-----------------------------------------------
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
# crear lista----------------------------------------------------
clases = [45, 48, 50, 55, 45, 46, 47, 49, 48, 49]
plt.plot(clases);
plt.show()
```

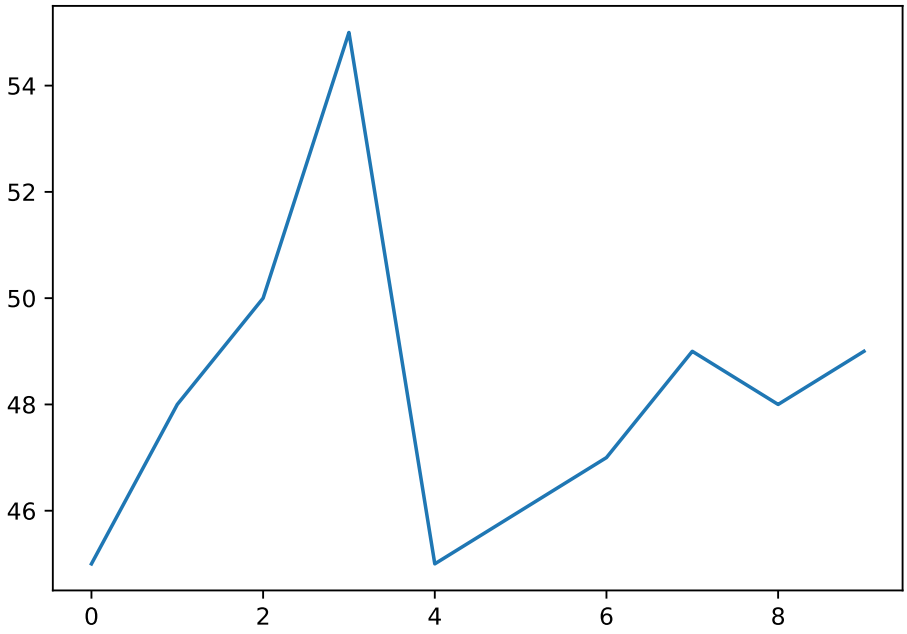


Figura 3: Gráfico de líneas de las duraciones de las clases

Para poder modificar los ejes del gráfico, se va a emplear el siguiente código, con el cual conseguiremos obtener el resultado que se muestra en la Figura 4.

- Primero se utilizan las funciones **xlabel** y **ylabel** para introducir nombres a los ejos.

- Posteriormente, se introduce el indicador de la sesión en el eje para cambiar del 0 al 9 por del 1 al 10 en (X). Este cambio se logra con la función **xticks**, en la cual hay que introducir un primer argumento para marcar los **breaks**, es decir, las posiciones numéricas en el eje, y en un segundo argumento, las etiquetas que se van a corresponder a cada uno de esos breaks.

- Además, se cambian los límites del eje (Y) con la función **ylim**. Se añade un título con la función **plt.title** (Normalmente, en lugar de añadir el título del gráfico, se introduce esta información en el pie de la figura) y se añade una leyenda con la función **plt.legend** donde se pasa como argumento la etiqueta que queremos que aparezca en dicha leyenda.

- Por último, se introduce una rejilla para cada marca de los ejes (X) e (Y) de modo que sea más fácil de visualizar los resultados. Se emplea la función **grid(True)**, donde se ha añadido el argumento **alpha** que permite definir la transparencia (0 = totalmente transparente; 1 = totalmente opaco) de la rejilla.

```
# definir ejes
nombres_x = range(1,11)
# formatear graficos----------------------------------------------
plt.plot(clases);
# nombrar ejes----------------------------------------------------
plt.xlabel("Sesión");
plt.ylabel("Tiempo (minutos)");
# poner marcas en el eje x----------------------------------------
plt.xticks(range(0,10), range(1,11));
plt.ylim(40, 58);
plt.grid(True, alpha = 0.5);
plt.title("Clases UNIR");
plt.legend(["Clases Optimizacion"]);
plt.show();
```

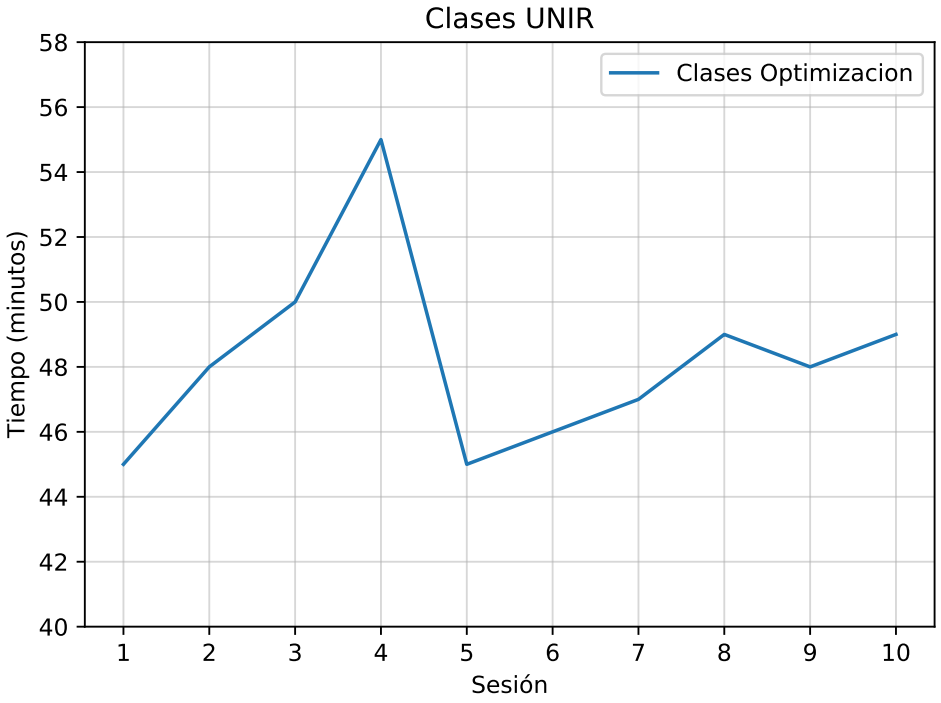


Figura 4: Gráfico de líneas de las duraciones de las clases formateado

- **Gráfico de dispersión** (*Scatterplot*)

De modo similar al gráfico de líneas, se puede representar cada pareja de datos mediante un gráfico de dispersión, en el cual los puntos no van unidos. En este caso, podemos:

- Emplear la función *plt.scatter*, como en la Figura 5, donde es necesario introducir como argumentos los valores de x e y.

```
# formatear graficos-----------------------------------------
plt.scatter(range(0,10), clases);
# nombrar ejes-----------------------------------------------
plt.xlabel("Sesión");
plt.ylabel("Tiempo (minutos)");
# poner marcas en el eje x-----------------------------------
plt.xticks(range(0,10), range(1,11));
plt.ylim(40, 58);
```

```
plt.grid(True, alpha = 0.5);
plt.title("Clases UNIR");
plt.legend(["Clases Optimizacion"]);
plt.show();
```

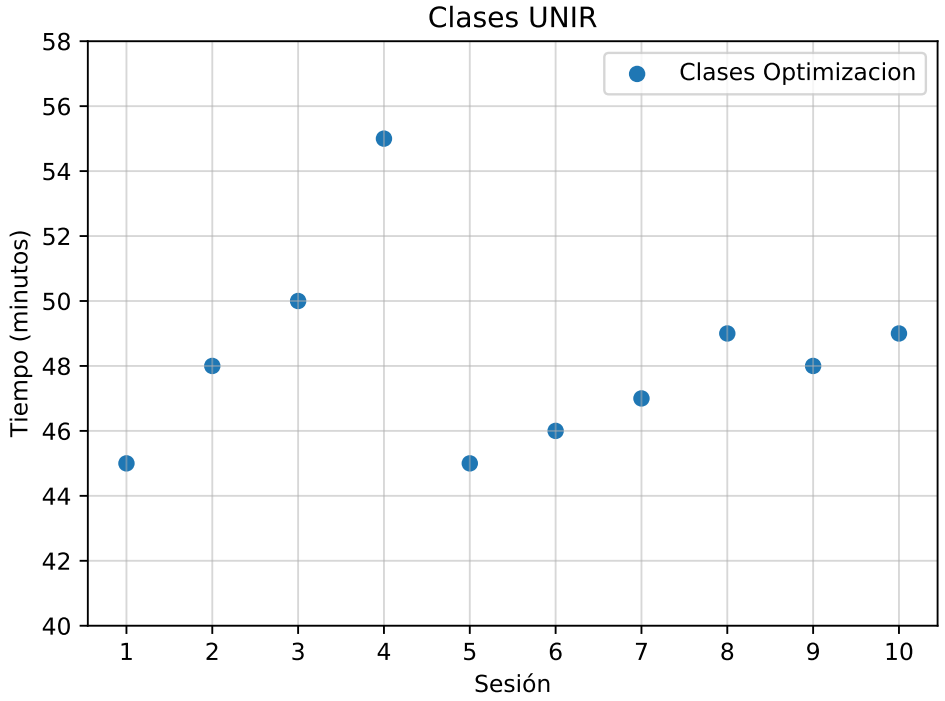


Figura 5: Gráfico de dispersión de de las duraciones de las clases formateado

- Emplear la función *plt.stem*, como en la Figura 6, donde para cada valor se representa una línea vertical hasta el valor del punto para cada observación.

```
# definir ejes
nombres_x = range(1,11)
# formatear graficos-----------------------------------------
plt.stem(clases, use_line_collection = True);
# nombrar ejes-----------------------------------------------
```

```
## <StemContainer object of 3 artists>
```

```
plt.xlabel("Sesión");
plt.ylabel("Tiempo (minutos)");
# poner marcas en el eje x-----------------------------------
```

```
plt.xticks(range(0,10), range(1,11));
plt.ylim(40, 58);
plt.grid(True, alpha = 0.5);
plt.title("Clases UNIR");
plt.legend(["Clases Optimizacion"]);
plt.show();
```

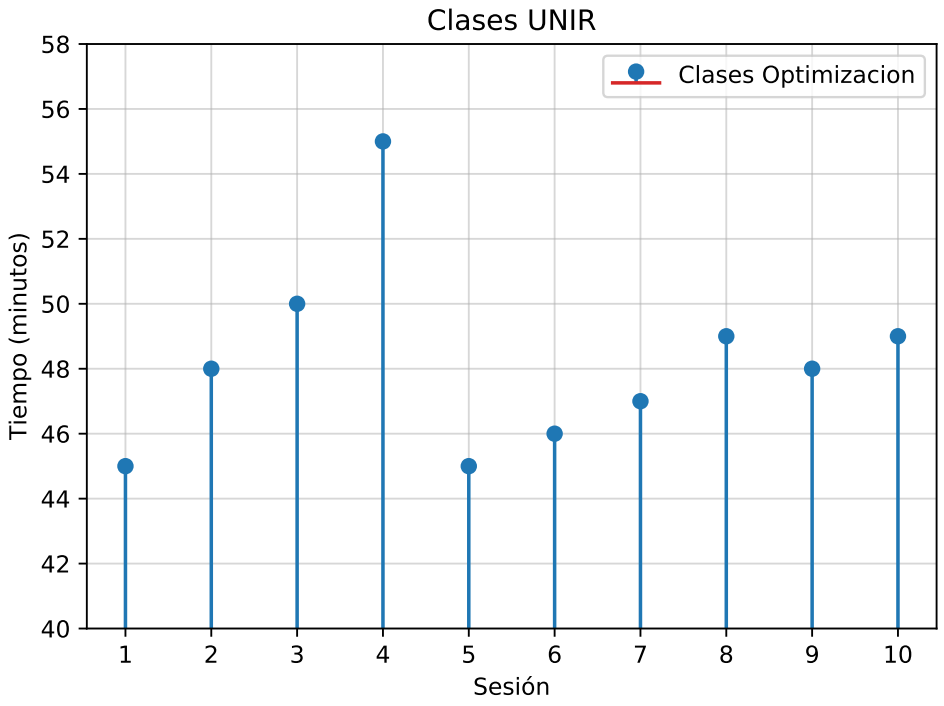


Figura 6: Gráfico de líneas verticales de las duraciones de las clases formateado

- **Histograma**. En *matplotlib* el histograma se realiza mediante la función **plt.hist**. Se va a representar un ejemplo de un histograma en la Figura 7 de unos datos simulados de una distribución normal de 500 posibles alturas. El argumento *bins* marca el número de barras a aparecer en el histograma.

```
alturas = np.random.normal(175, 10, 500)
plt.hist(alturas, bins = 10, edgecolor = "black");
plt.title("Distribución de 500 alturas")
plt.xlabel("Altura (cm)")
plt.ylabel("n")
plt.show()
```

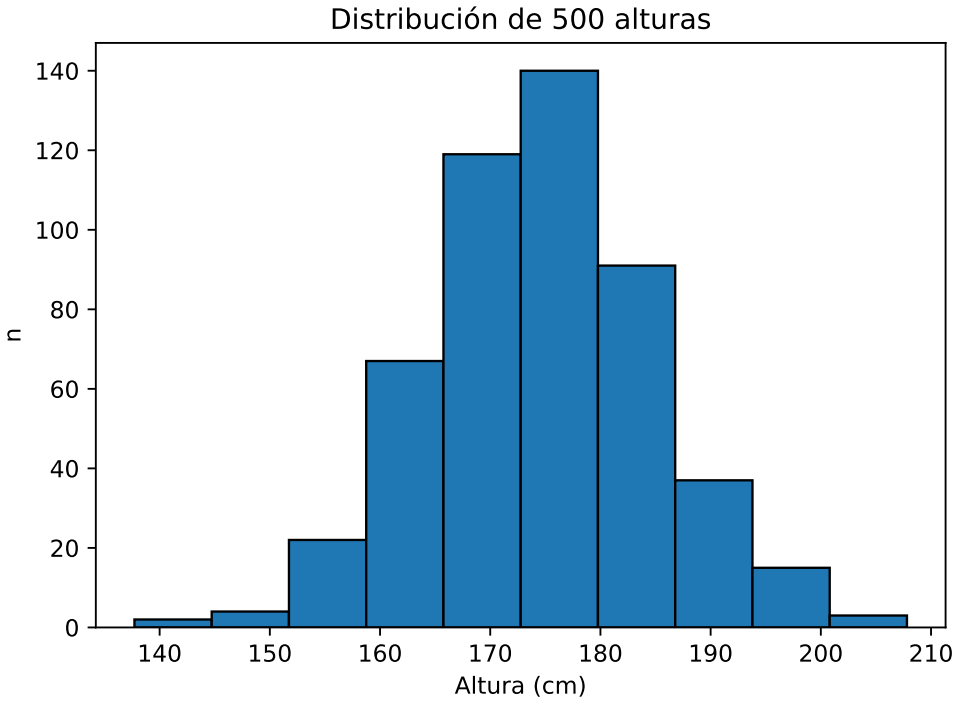


Figura 7: Ejemplo de histograma

- **Diagrama de cajas y bigotes** o *boxplot*. En *matplotlib* se realiza mediante la función **plt.boxplot**. Vamos a ver un ejemplo de su realización a partir de datos simulados. En concreto, se han simulado de una distribución uniforme 3 grupos de 100 posibles alturas. A las alturas del grupo 1 se le ha añadido un outlier, que aparece como un punto en la Figura 8.

```
# marcamos una semilla para obtener siempre los mismo resultados
np.random.seed(3)
# simulamos 3 grupos de alturas
alturas_1_0 = np.random.uniform(164, 175, 100)
# introducir outlier
alturas_1 = np.append(alturas_1_0, 180)
alturas_2 = np.random.uniform(175, 185, 100)
alturas_3 = np.random.uniform(170, 185, 100)
# cambiar los colores
box = plt.boxplot([alturas_1, alturas_2, alturas_3]);
```

```
plt.xticks([1, 2, 3], ["1", "2", "3"]);
plt.ylabel("Alturas (cm)");
plt.xlabel("Grupo");
plt.show()
```

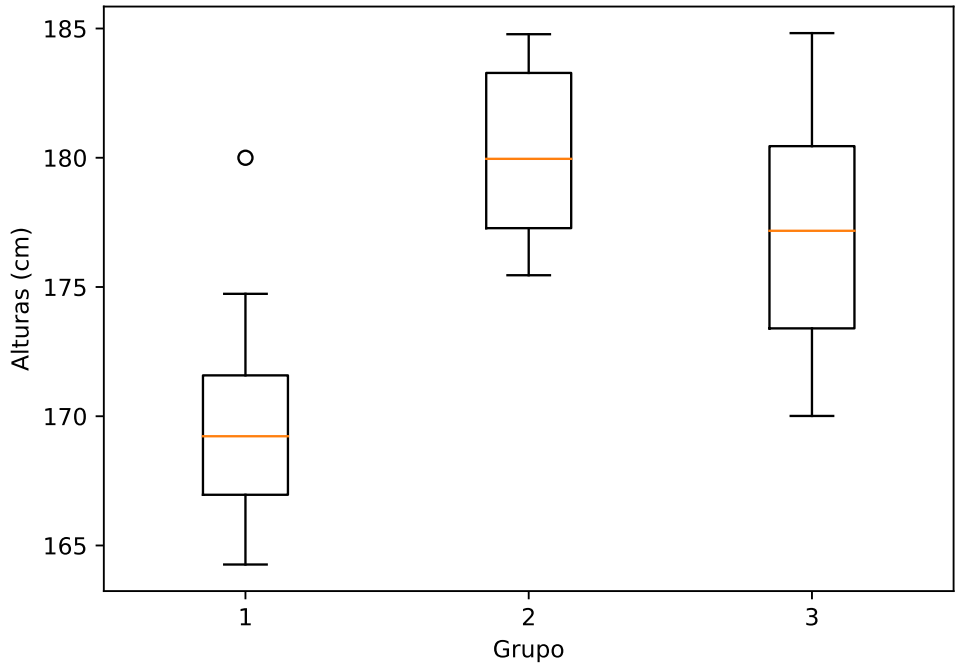


Figura 8: Ejemplo de diagrama de cajas y bigotes

- **Diagrama de barras**. En *matplotlib*, el diagrama de barras se realiza con la función *plt.bar*. Para realizar un ejemplo de un diagrama de barras se van a usar las duraciones de las clases de optimización empleadas anteriormente, tal y como se muestra en la Figura 9.

```
plt.bar(range(1,11), clases, edgecolor = "black");
plt.xticks(range(1,11));
plt.title("clases UNIR");
# definir límite eje y----------------------------------------
plt.ylim(min(clases)-1, max(clases)+1);
# nombrar ejes-----------------------------------------------
```

```
## (44.0, 56.0)
```

```
plt.xlabel("Sesión");
plt.ylabel("Tiempo (minutos)");
plt.show()
```

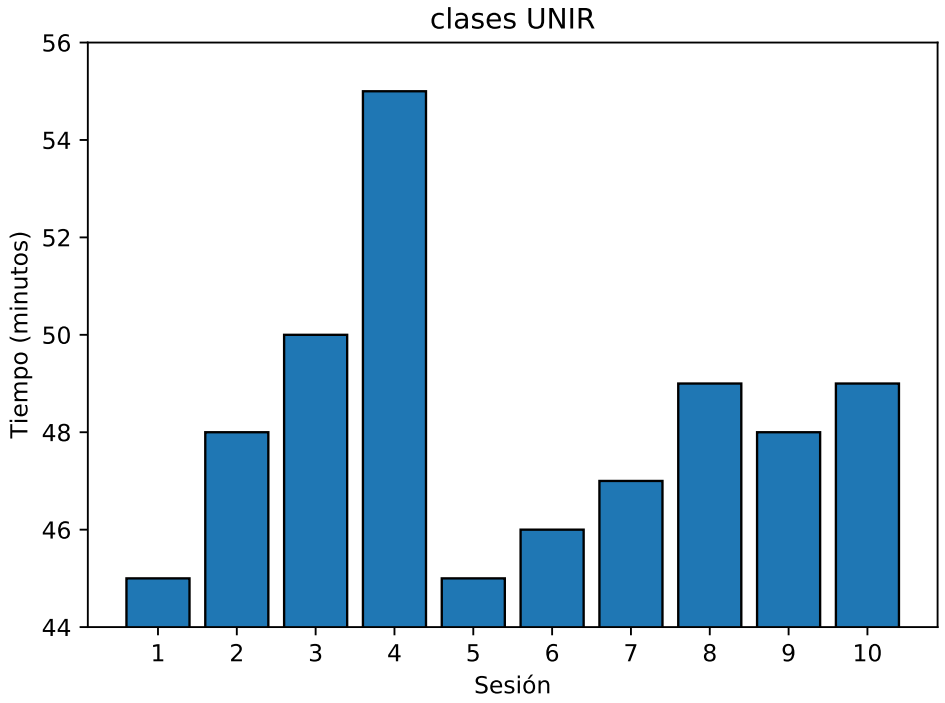


Figura 9: Ejemplo de diagrama de barras

- **Diagrama de sectores**. En *matplotlib* el diagrama de sectores se realiza con la función plt.pie. Para mostrar un ejemplo se ha empleado las duraciones de las clases tal y como se muestra en la Figura 10. Se observa que es preferible otro tipo de representación, ya que todos los sectores tienen un tamaño muy similar y es difícil apreciar diferencias.

```
sum(clases)
```

```
## 482
```

```
porcentajes_clases = [x / sum(clases) for x in clases]
print(porcentajes_clases)
```

```
plt.pie(porcentajes_clases, labels=range(1, 11),
        autopct='%1.1f%%', shadow=True, startangle=90)

plt.title("clases UNIR");
# nombrar ejes----------------------------------------------------
plt.show()
```

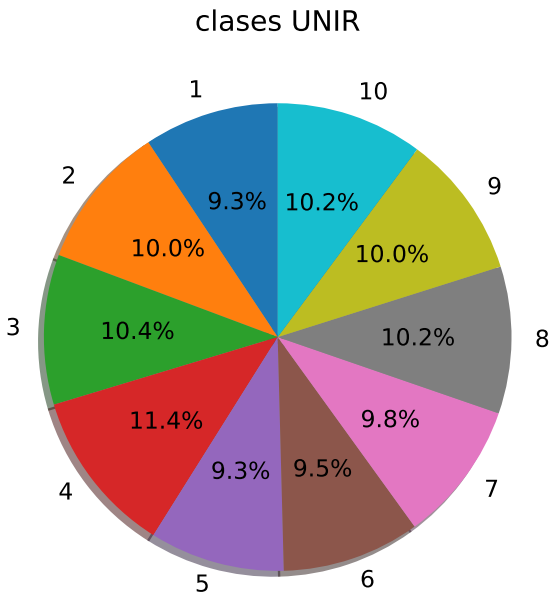


Figura 10: Ejemplo de diagrama de sectores

10.6 Otras librerías en Python

Nineplot (ggplot)

Nineplot está basada en el *Grammar of graphics* y la sintaxis que utiliza es la misma que la famosa librería *ggplot2* de R. Por lo tanto, puede ser una gran herramienta en Python para los conocedores de ambos lenguajes. Permite realizar gráficos sofisticados y elegantes, y aunque su manejo puede no resultar tan intuitivo como Matplotlib, una

vez que se adquiere la suficiente práctica, permite combinar múltiples elementos en un mismo gráfico de manera sencilla.

Seaborn

Seaborn nace como una librería para realizar gráficos modernos y ponerse a la vanguardia de la visualización de datos. La sintaxis que emplea es propia, y su manejo se recomienda a los usuarios que tienen algo de experiencia tanto en el manejo de Python como en la visualización de datos.

Material audiovisual

Accede al vídeo: Visualización de los datos

10.7 Referencias bibliográficas

Andrienko, G.; Andrienko, N. (2013). Visual analytics of movement: An overview of methods, tools and procedures. 12:3–24.

Andrienko, G.; Andrienko, N. W. S. (2007). Visual analytics tools for analysis of movement data. 9:38–46.

Anscombe, F. J. (1973). Graphs in statistical analysis. 27:17–21.

Schneiderman, B. (1996). The eyes have it: a task by data type taxonomy for information visualizations. In *In Proceedings 1996 IEEE Symposium on Visual Languages, 336-43.*

10.8 Ejercicios resueltos

Ejercicio 1.

Modifica el gráfico presentado en el tema sobre las duraciones de las sesiones en UNIR con los siguientes cambios:

a) Indica en el eje x en lugar del número de clases, el día de la semana en que ha sido impartido. Utiliza la siguiente lista:

[“martes”, “lunes”, “martes”, “lunes”, “viernes”, “jueves”, “lunes”, “viernes”, “viernes”, “martes”]

b) Modifica el valor del límite del eje (Y) fijándolo en 60.

c) Cambia el color de la línea a rojo.

Solución

El primer paso, como viene siendo habitual es importar las librerías que se van a emplear en la resolución de los ejercicios. En este caso vamos a necesitar las librerías *numpy*, *matplotlib* y *matplotlib.pyplot*.

```
# cargar librerias-----------------------------------------------
import numpy as np
```

```
import matplotlib as mpl
import matplotlib.pyplot as plt
```

A continuación se presenta el código y la Figura 11 resultante de aplicar los cambios.

a) Para cambiar los valores de los nombres del eje X, primero se definen los *nombres_x* como una lista de valores con los días de la semana correspondientes a la clase. Después, en la función plt.xticks, en el primer argumento se indica las posiciones en las cuales tienen que aparecer los nombres del eje x, y en el segundo argumento se llama a dichos nombres. Además, para que quepa en la figura el texto del eje x, antes de crear la figura, se define el tamaño de la misma con el argumento *figsize* de la función *plt.figure*, donde se especifica en el primer componente del argumento la anchura y en el segundo componente la altura del gráfico en pulgadas.

b) Para modificar el valor máximo que aparece en el eje y, se utiliza la función *plt.ylim*, donde el valor que antes aparecía como 58 se ha modificado hasta 60.

c) Por último, para cambiar los valores de los colores de la línea de la figura se introduce el argumento *color* de la función *plt.plot*.

```
# definir nombres-------------------------------------------------
nombres_x = ["martes", "lunes", "martes", "lunes", "viernes",
"jueves", "lunes", "viernes", "viernes", "martes"]
# cambiar tamaño figura-------------------------------------------
plt.figure(figsize = (7, 5.25));
plt.plot(clases, color = "red");
# nombrar ejes----------------------------------------------------
plt.xlabel("Sesión");
plt.ylabel("Tiempo (minutos)");
# poner marcas en el eje x----------------------------------------
plt.xticks(range(0,10), nombres_x, rotation = 60);
# cambiar limite eje y--------------------------------------------
```

```
plt.ylim(40, 60);
plt.grid(True, alpha = 0.5);
plt.title("Clases UNIR");
plt.legend(["Clases Optimizacion"]);
plt.show();
```

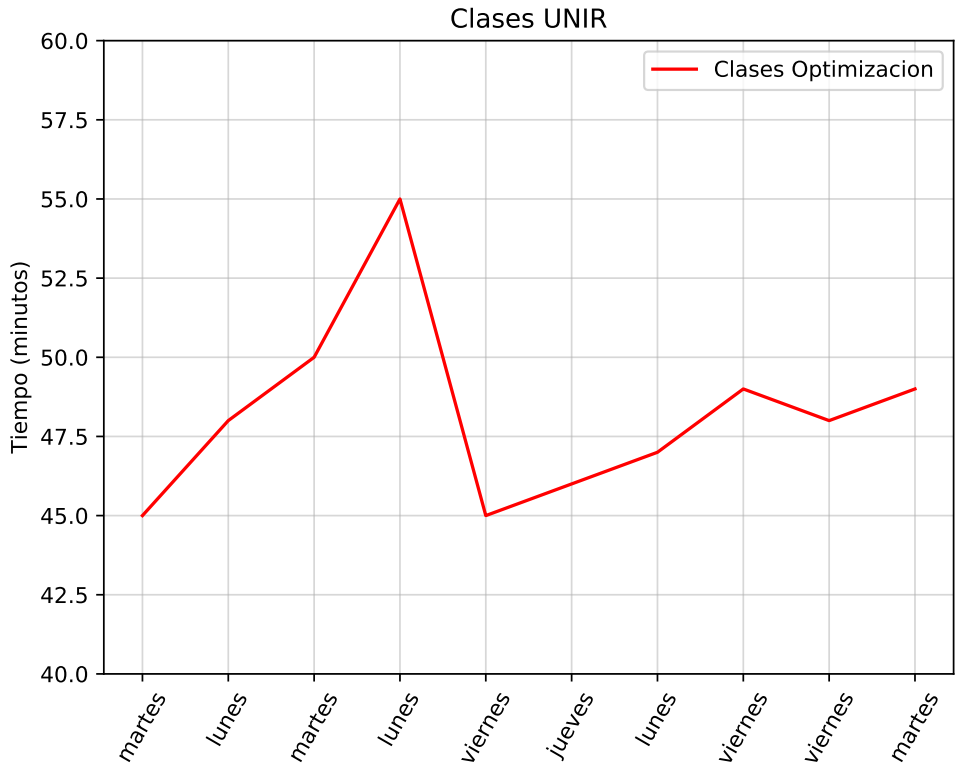


Figura 11: Gráfico de líneas formateado según instrucciones del Ejercicio 1

Ejercicio 2

Partiendo del diagrama de cajas y bigotes realizado en el tema como ejemplo, aplica los siguientes cambios sobre su formato:

a) Que los cuerpos del histograma estén mellados (notched).

b) Que la altura 1 sea de color azul, la altura 2 de color verde y la altura 3 de color naranja.

c) Que el color de la mediana, la línea vertical y de los bigotes sea morado.

Solución

A continuación se muestra el código y la Figura 12 resultante de aplicar los cambios propuestos.

a) Para cambiar la forma de las cajas del diagrama se activa la opción *notch = True*.

b) Para colorear las cajas se activa el argumento *patch_artist = True*. Por defecto, se colorearían todos los grupos del mismo color (azul), por lo que para escoger diferentes colores para cada una de ellas, primero se define una lista con los colores escogidos (*colors*) y después se emplea un bucle *for* en el cual se define para cada patch y color que el *patch.set_facecolor* sea el color escogido. Para poder realizar esta operación, se ha guardado el gráfico en el objeto *box*.

c) Para cambiar los colores de las líneas, medianas y bigotes se emplean las propiedades de los mismos dentro de la función *plt.boxplot*, es decir, *capprops*, *medianprops* y *whiskerprops*.

```
# cambiar formato-----------------------------------------------
box = plt.boxplot([alturas_1, alturas_2, alturas_3],
notch = True, # mellar cajas
patch_artist=True, # dar color a las cajas
# cambiar colores de lineas, mediana y bigotes
            capprops=dict(color="purple"),
            medianprops=dict(color="purple"),
            whiskerprops=dict(color="purple"));
plt.xticks([1, 2, 3], ["1", "2", "3"]);
plt.ylabel("Alturas (cm)");
plt.xlabel("Grupo");
# cambiar los colores de las cajas por grupo--------------------
```

```
colors = ["blue", "green", "orange"]
for patch, color in zip(box["boxes"], colors):
    patch.set_facecolor(color)

plt.show()
```

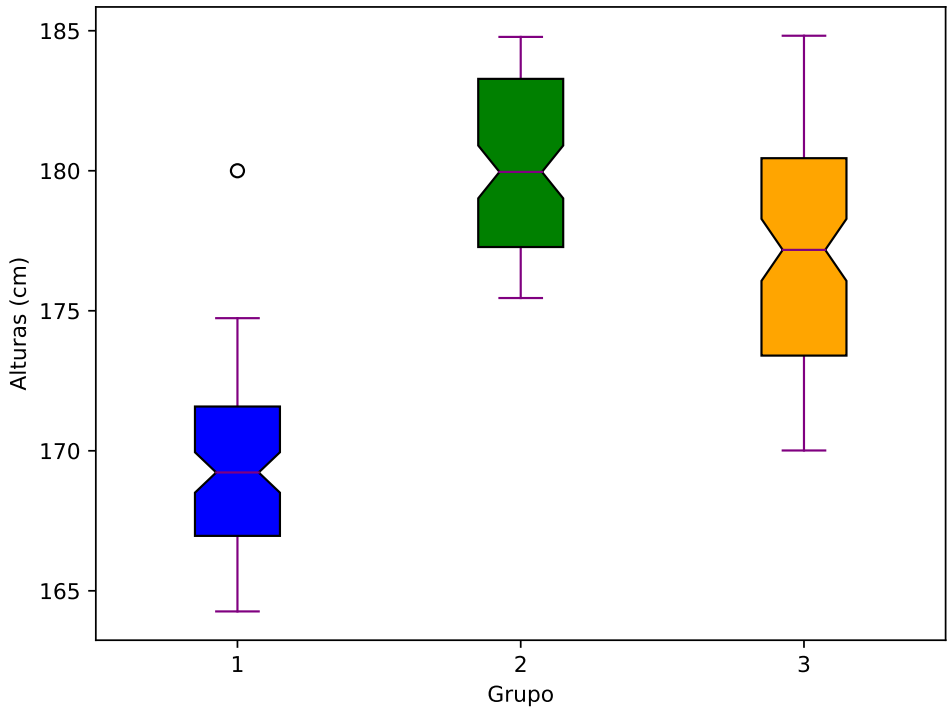


Figura 12: Ejemplo de boxplot formateado

Ejercicio 3

Realiza una figura, compuesta de varias subfiguras. En particular, realiza el histograma de las alturas simulado mediante normales en la parte superior de la figura y los 3 histogramas de las alturas simuladas mediante uniformes en la parte inferior. Varía los colores de cada uno de los histogramas.

Solución

Agrupar distintas subfiguras en una única figura es una de las propiedades más potentes de la visualización, la cual nos permite reunir distinta información en una única figura. Para obtener este tipo de representaciones se puede realizar cada imagen por separado y luego unirlas con el editor de texto en el cual estemos realizando el trabajo o informe. Sin embargo, en muchas ocasiones el resultado no acaba de ser bueno o es muy costoso que cuadren bien las distintas figuras al tratar de agruparlas. Una posible solución, es realizar todas las figuras que se quieren agrupar en un mismo plot. Para realizar esta acción en *matplotlib* se puede utilizar la función *plt.subplots*.

A continuación se muestra el código empleado para la obtención de la Figura 13. En la función *plt.subplot* se define cada subplot mediante tres argumentos: el primero define el número de columnas de la composición de subplots que va a tener el plot, el segundo el número de filas y el tercero el índice que va a ocupar el subplot. En nuestro caso, se va a querer dividir el espacio en 2 filas por 3 columnas. Sin embargo, para el primer subplot, le vamos a indicar 2 filas y 1 columna, puesto que va a ocupar el espacio de las 3 columnas de esa primera fila (la fila superior). Para el resto de subplots sí que se va a indicar que la figura se compone de 2 filas y 3 columnas y las posiciones que van a ocupar los histogramas de las alturas 1, 2 y 3 van a ser las correspondientes a los índices 4, 5 y 6 respectivamente. Para cambiar el color de cada uno de los histogramas, se emplea el argumento *color* de la función *plt.hist*. Además, se han definido el argumento *edgecolor*, que marca el color de las lineas de cada una de las barras, y el argumento *bins*, que indica el número de barras de cada histograma. Por último, es necesario ajustar mediante el argumento *figsize* de la función *plt.figure* a un tamaño adecuado para que se pueda representar bien la composición de las subfiguras.

```
# cambiar tamaño de la figura-----------------------------------
plt.figure(figsize = (12, 13))


# definir histograma de la primera subfigura--------------------
plt.subplot(2, 1, 1)
```

```
plt.hist(alturas, bins=10, edgecolor = "black", color = "blue");
plt.title("Distribución de 500 alturas simuladas de una normal")
plt.xlabel("Altura (cm)")
plt.ylabel("n")

# definir histograma de la segunda subfigura----------------------
plt.subplot(2, 3, 4)
plt.hist(alturas_1, bins=10, edgecolor = "black", color = "green");
plt.title("Distribución alturas 1")
plt.xlabel("Altura (cm)")
plt.ylabel("n")

# definir histograma de la tercera subfigura----------------------
plt.subplot(2, 3, 5)
plt.hist(alturas_2, bins=10, edgecolor = "black", color = "purple");
plt.title("Distribución alturas 2")
plt.xlabel("Altura (cm)")
plt.ylabel("n")

# definir histograma de la cuarta subfigura----------------------
plt.subplot(2, 3, 6)
plt.hist(alturas_3, bins=10, edgecolor = "black", color = "red");
plt.title("Distribución alturas 3")
plt.xlabel("Altura (cm)")
plt.ylabel("n")


plt.show()
```

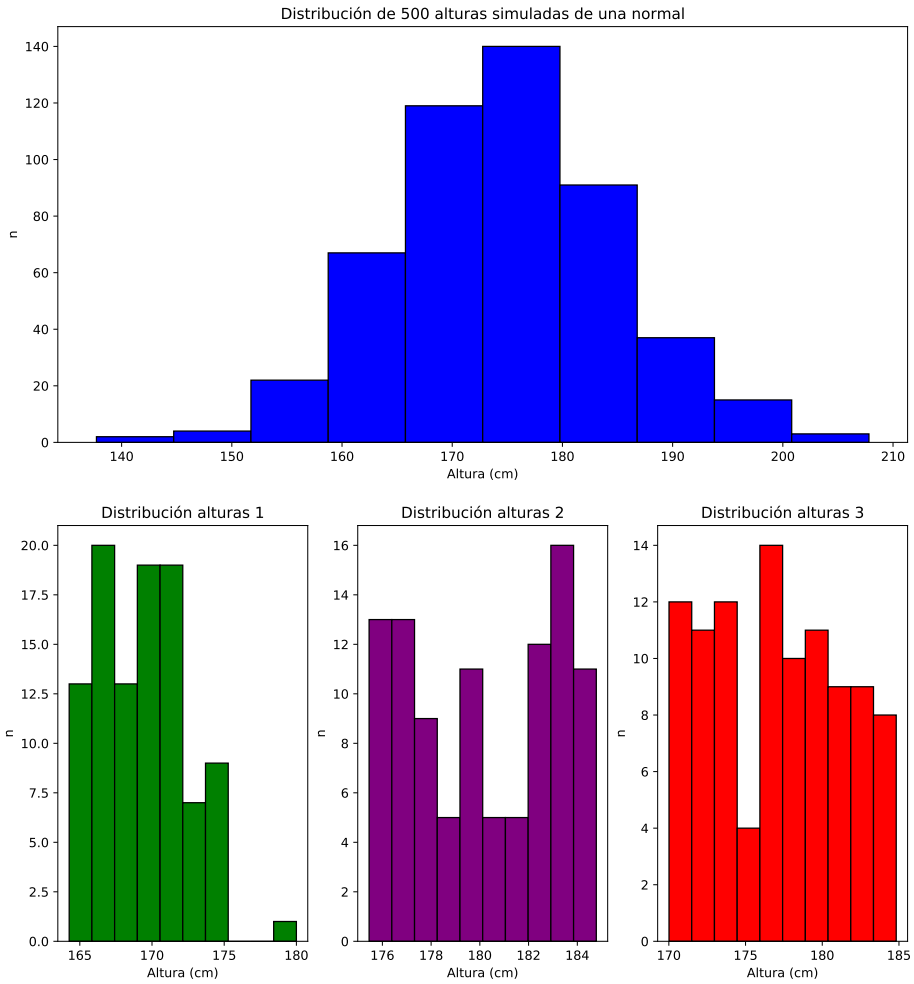


Figura 13: Ejemplo de composición de subfiguras