dcc
CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Análisis Exploratorio de Datos

Análisis Exploratorio de Datos

- El análisis exploratorio de datos o (EDA) engloba un conjunto de técnicas para poder comprender/resumir las características más importantes de un conjunto de datos o **dataset**.

- Esta metodología fue impulsada por el estadístico John Tukey.

- Se basa principalmente en dos criterios: Las **estadísticas de resumen** y la **visualización de datos**.

- En esta clase se verán ambos tipos de técnicas, además de su aplicación en **python** para datasets conocidos.

John Tukey

Python y Anaconda

- Python es un lenguaje de programación interpretado y de propósito general. Open Source: https://www.python.org/

- Es un lenguaje completo que soporta diferentes paradigmas de programación como programación orientada a objeto.

- Anaconda es una distribución de Python que incluye una gran colección de paquetes científicos preinstalados.

- Tiene una configuración de entornos para ciencia de datos y aprendizaje automático sea más sencilla y rápida.

NumPy y SciPy

- NumPy:
  - Proporciona arrays multidimensionales, que son más rápidos y menos pesados que las listas de Python tradicionales
  - Sirve como base para otras bibliotecas científicas y de análisis de datos como pandas, SciPy y scikit-learn, debido a su integración y rendimiento.

- SciPy:
  - SciPy se basa en NumPy y ofrece módulos adicionales para optimización, álgebra lineal, integración, interpolación, funciones especiales, procesamiento de señales y de imágenes.
  - Extiende las funcionalidades de NumPy con herramientas más especializadas

Pandas

- Es una biblioteca externa de Python para manipulación y análisis de datos.
- Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales.
- Permite una variedad de operaciones de manipulación de datos incluyendo:
  - Fusionar, unir y concatenar datos.
  - Filtrar, agrupar y transformar datos.
- Pandas introduce dos nuevas estructuras de datos a Python – DataFrame y Series, inspirados en los dataframes de R.

Scikit-Learn

- Es una biblioteca externa de Python que provee una gran variedad de algoritmos supervisados y no supervisados.

- Incluye funciones para preprocesamiento de datos, reducción de dimensionalidad, selección de modelos y validación cruzada, fundamentales para la preparación de datos y la evaluación de modelos.

- Funciona bien con otras bibliotecas de Python, como NumPy y SciPy, y es compatible con muchos formatos de datos, permitiendo una integración fluida en el flujo de trabajo de ciencia de datos.

El dataset Iris

- Trabajaremos con un dataset muy conocido en análisis de datos llamado Iris.
- El dataset se compone de 150 observaciones de flores de la planta iris.
- Existen tres tipos de clases de flores iris: virginica, setosa y versicolor.
- Hay 50 observaciones de cada una.
- Las variables o atributos que se miden de cada flor son:
  - El tipo de flor como variable categórica.
  - El largo y el ancho del pétalo en cm como variables numéricas.
  - El largo y el ancho del sépalo en cm como variables numéricas.

El dataset Iris

Virginica

Setosa

Versicolor

- El dataset se encuentra disponible en Scikit-Learn, una librería de Machine Learning en Python:

```
> from sklearn.datasets import load_iris
> import pandas as pd
> iris = load_iris()
> iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)
> iris_df.columns

Index(['sepal length (cm)', 'sepal width (cm)', 'petal length
(cm)',  'petal width (cm)', 'species'],   dtype='object')
```

Datos Categóricos en Pandas

- `Categoricals` son un tipo de datos de pandas que corresponden a variables categóricas en estadísticas.
- Una variable categórica asume un número limitado y usualmente fijo de valores posibles (categorías; niveles en R).
- Ejemplos son el género, clase social, tipo de sangre, afiliación a un país, tiempo de observación o calificación mediante escalas de Likert.

```
> iris_df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)
```

['setosa', 'versicolor', 'virginica']

Categories (3, object): ['setosa', 'versicolor', 'virginica']

Estadísticas de Resumen

- Las estadísticas de resumen son valores que explican propiedades de los datos.
- Algunas de estas propiedades incluyen: frecuencias, medidas de tendencia central y dispersión.
- Ejemplos:
  - Tendencia central: media, mediana, moda.
  - Dispersión: miden la variabilidad de los datos, como la desviación estándar, el rango, etc..
- La mayor parte de las estadísticas de resumen se pueden calcular haciendo una sola pasada por los datos.

Frecuencia y Moda

- La frecuencia de un valor de atributo es el porcentaje de veces que éste es observado.
- En pandas podemos contar las frecuencias de aparición de cada valor distinto dentro de una columna usando una función *value_counts*:

```
> iris_df['species'].value_counts()

setosa        50
versicolor    50
virginica     50
Name: species, dtype: int64
```

- Ejercicio: Calcular las frecuencias porcentuales del resultado anterior.

```
# Frecuencia porcentual
> (iris_df['species'].value_counts(normalize=True) *
100).round(2)
```

Frecuencia y Moda (2)

- La moda de un atributo es el valor más frecuente observado.
- En pandas, existe una función específica para calcular la moda:

```
> iris_df.mode()
```

	sepal length (cm)	sepal width (cm)	...	petal width (cm)	species
0	5.0	3.0	...	0.2	setosa

- Generalmente usamos la frecuencia y la moda para estudiar variables categóricas.

Medidas de Tendencia Central

- Estas medidas tratan de resumir los valores observados en único valor asociado al valor localizado en el centro.
- La media es la medida más común de tendencia central para una variable numérica.
- Si tenemos *m* observaciones se calcula como la media aritmética o promedio.

$$\text{mean}(x) = \overline{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$$

Medidas de Tendencia Central (2)

- El mayor problema de la media es que es una medida muy sensible a outliers o valores atípicos.
- Ejemplo: Tomamos un vector aleatorio de media 20 y luego le agregamos un elemento aleatorio que proviene de una distribución de media mucho mayor.
- Vemos que la media es fuertemente afectada por el ruido:

```
> import numpy as np
> vec = np.random.normal(20, 10, 10)
> mean_vec = np.mean(vec)
> vec_outlier = np.append(vec, np.random.normal(300, 100))
> mean_vec_outlier = np.mean(vec_outlier)
> (mean_vec, mean_vec_outlier)
(27.38023170728834, 53.4733340171792)
```

Medidas de Tendencia Central (3)

- Podemos robustecer la media eliminando una fracción de los valores extremos usando la **media truncada** o **trimmed mean**.
- En python podemos usar la librería scipy que implementa trim_mean
- El segundo parámetro define la fracción de elementos extremos a descartar.
- Ejemplo: Descartamos el 10 % de los valores extremos en el ejemplo anterior:

```
> from scipy import stats
> trimmed_mean_vec = stats.trim_mean(vec, 0.1)
> trimmed_mean_vec_outlier = stats.trim_mean(vec_outlier, 0.1)
(27.645770484954127, 29.286121763516398)
```

Medidas de Tendencia Central (4)

- La mediana representa la posición central de la variable que separa la mitad inferior y la mitad superior de las observaciones.
- Intuitivamente, consiste en el valor donde para una mitad de las observaciones todos los valores son mayores que ésta, y para la otra mitad todos son menores.

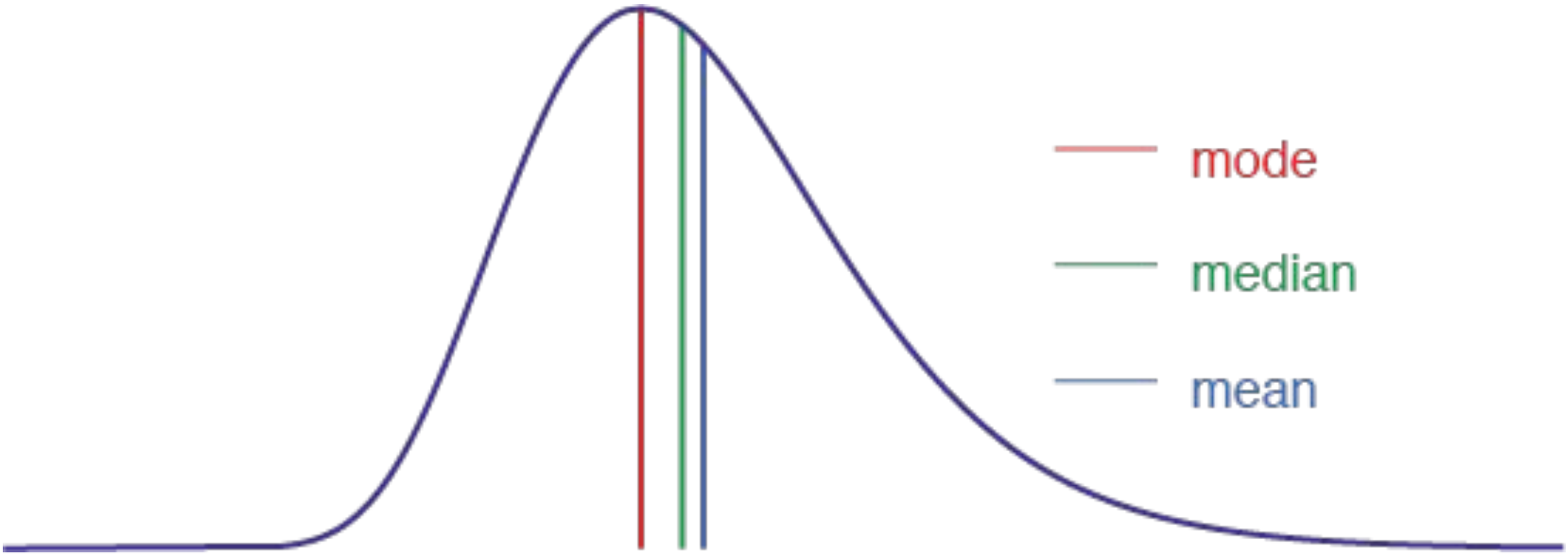
$$\text{median}(x) = \begin{cases} x_{r+1} & \text{Si } m \text{ es impar con } m = 2r+1 \\ \frac{1}{2}(x_r + x_{r+1}) & \text{Si } m \text{ es par con } m = 2r \end{cases}$$

- Para el ejemplo anterior, vemos que la mediana es más robusta al ruido que la media:

```
> median_vec = np.median(vec)
> median_vec_outlier = np.median(vec_outlier)
> (median_vec, median_vec_outlier)
```

```
(26.803434597319807, 29.500884175255894)
```

Comparación entre la moda, la mediana y la media



Percentiles o Cuantiles

- El k-ésimo percentil de una variable numérica es un valor tal que el k% de las observaciones se encuentran debajo del percentil y el (100 - k) % se encuentran sobre este valor.

- En estadística se usan generalmente los cuantiles que son equivalentes a los percentiles expresados en fracciones en vez de porcentajes.

- En pandas se calculan con el comando quantile de numpy:

```
> percentiles = iris_df['sepal length
(cm)'].quantile([i/100 for i in range(101)])
0.00    4.300
0.01    4.400
0.02    4.400
0.03    4.547
0.04    4.600
Name: sepal length (cm), dtype: float64
```

Percentiles o Cuantiles (2)

- Además es muy común hablar de los cuartiles que son tres percentiles específicos:
    - El primer cuartil Q1 (lower quartile) es el percentil con k = 25.
    - El segundo cuartil Q2 es con k = 50 que equivale a la mediana.
    - El tercer cuartil Q3 (upper quartile) es con k = 75.

El mínimo, los tres cuartiles y el máximo

```
> iris_df['sepal length (cm)'].quantile([0, 0.25, 0.5, 0.75, 1])
```

0% 25% 50% 75% 100%
4.3   5.1   5.8   6.4   7.9
Name: sepal length (cm), dtype: float64

Resumiendo un Data Frame (1)

- En pandas podemos resumir varias estadísticas de resumen de una variable de un *DataFrame* usando el comando *describe*.
- Para las variables numéricas nos entrega el mínimo, los cuartiles, la media y el máximo.

```
> iris_df.describe()
```

	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Ejercicio

- Usando el comando groupby analice distintas estadísticas de resumen para las tres especies de Iris para las cuatro variables.

- ¿Nota alguna diferencia en las distintas especies?

```
iris_df.groupby('species')['sepal length (cm)'].describe()
iris_df.groupby('species')['sepal width (cm)'].describe()
iris_df.groupby('species')['petal length (cm)'].describe()
iris_df.groupby('species')['petal width (cm)'].describe()
```

Respuesta:

```
> iris_df.groupby('species')['petal length (cm)'].describe()

            count   mean      std  min  25%   50%    75%  max
species
setosa       50.0  1.462  0.173664  1.0  1.4  1.50  1.575  1.9
versicolor   50.0  4.260  0.469911  3.0  4.0  4.35  4.600  5.1
virginica    50.0  5.552  0.551895  4.5  5.1  5.55  5.875  6.9
```

Medidas de Dispersión

- Estas medidas nos dicen qué tan distintas o similares tienden a ser las observaciones respecto a un valor particular.

- Generalmente este valor particular se refiere a alguna medida de tendencia central.

- El rango es la diferencia entre el valor máximo y el mínimo:

```
> iris_df['sepal length (cm)'].max() - iris_df['sepal length
(cm)'].min()
```

3.6

Medidas de Dispersión (2)

- La desviación estándar es la raíz cuadrada de la varianza que mide las diferencias cuadráticas promedio de las observaciones con respecto a la media.

$$\mathrm{var}(x) = \frac{1}{m-1}\sum_{i=1}^{m}(x_i - \overline{x})^2$$

$$\mathrm{sd}(x) = \sqrt{\mathrm{var}(x)}$$

```
> iris_df['sepal length (cm)'].var()
0.6856935
> iris_df['sepal length (cm)'].std()
0.8280661
```

- ¿Por qué (m - 1)?: Bessel's correction

Medidas de Dispersión (3)

- Al igual que la media, la desviación estándar es sensible a outliers.

- Las medidas más robustas se basan generalmente en la mediana.

- Sea m(x) una medida de tendencia central de x (usualmente la mediana), se define la **desviación absoluta promedio** o **average absolute deviation** (AAD) como:

$$\mathrm{AAD}(x) = \frac{1}{m} \sum_{i=1}^{m} |x_i - m(x)|$$

Medidas de Dispersión (4)

- Ejercicio: programe la desviación absoluta promedio en python como una función llamada *aad*.

- La función recibe un vector *x* y una función de media central *fun*.

- El valor absoluto se calcula con el comando *abs*.

- Solución:

```
import numpy as np

def aad(x, fun=np.median):
    return np.mean(abs(x - fun(x)))

result1 = aad(iris_df['sepal length (cm)'])
result2 = aad(iris_df['sepal length (cm)'], np.mean)
print(result1)
print(result2)
```

Medidas de Dispersión (5)

- Sea *b* una constante de escala se define la **desviación mediana absoluta** o **median absolute deviation** como:

$$\text{MAD}(x) = b \times \text{median}(|x_i - \text{m}(x)|)$$

- En python se calcula con la función *scipy.stats.median_abs_deviation* (no confundir con la función *mad* de pandas, que calcula *mean absolute deviation*, y que además está deprecada)

```
> stats.median_abs_deviation(iris_df["sepal length (cm)"])
```
0.7

- Finalmente, se define el rango intercuartil (IQR) como la diferencia entre el tercer y el primer cuartil (Q3 – Q1).

```
> iris_df['sepal length (cm)'].quantile(0.75) - iris_df['sepal length
(cm)'].quantile(0.25)
```
1.3

Estadísticas de Resumen Multivariadas

- Para comparar cómo varía una variable respecto a otra, usamos medidas multivariadas.

- La covarianza *cov(x, y)* mide el grado de variación lineal conjunta de un par de variables *x*, *y* :

$$cov(x, y) = \frac{1}{m-1} \sum_{i=1}^{n} (x - \overline{x})(y - \overline{y})$$

- Donde *cov(x, x) = var(x)*
- En pandas se calcula con el comando *cov*:

```
> iris_df['sepal length (cm)'].cov(iris_df['sepal width (cm)'])
```

-0.042434

Estadísticas de Resumen Multivariadas (2)

- Para datasets de varias columnas numéricas uno puede calcular una matriz de covarianza.
- Cada celda i,j de esta matriz contiene la covarianza entre los atributos i y j.
- Esta matriz es simétrica.
- En pandas esta matriz se obtiene al aplicar el comando cov a una a un data.frame de variables numéricas:

```
> iris_df.drop(columns=["species"]).cov()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal length (cm)	0.685694	-0.042434	1.274315	0.516271
sepal width (cm)	-0.042434	0.189979	-0.329656	-0.121639
petal length (cm)	1.274315	-0.329656	3.116278	1.295609
petal width (cm)	0.516271	-0.121639	1.295609	0.581006

Estadísticas de Resumen Multivariadas (3)

- Si dos variables son independientes entre sí, su covarianza es cero.
- Para tener una medida de relación que no dependa de la escala de cada variable, usamos la **correlación lineal**.
- Se define a la correlación lineal o coeficiente de correlación de **Pearson** r(x, y) como:

$$r(x,y) = \frac{cov(x,y)}{sd(x)sd(y)}$$

- La correlación lineal varía entre –1 a 1.
- Un valor cercano a 1 indica que mientras una variable crece la otra también lo hace en una proporción lineal.
- Un valor cercano a -1 indica una relación inversa (una crece la otra decrece).

Estadísticas de Resumen Multivariadas (4)

- Una correlación cercana a 0 no implica que no pueda haber una relación no-lineal entre las variables.

- La correlación lineal se calcula en pandas con el comando *corr.*

- Podemos usarla de forma análoga a la covarianza para obtener una matriz de correlaciones.

> iris_df.drop(columns=["species"]).corr()

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
sepal length (cm)	1.000000	-0.117570	0.871754	0.817941
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126
petal length (cm)	0.871754	-0.428440	1.000000	0.962865
petal width (cm)	0.817941	-0.366126	0.962865	1.000000

Tablas de Contingencia

- Para analizar la relación entre variables de naturaleza categórica usamos tablas de contingencia.

- La tabla se llena con las frecuencias de co-ocurrencia de todos los pares de valores entre dos variables categóricas.
- En pandas se crean con el comando crosstab sobre atributos definidos como Categorical:

```
# Crear un DataFrame de ejemplo con datos categóricos
data = {'sexo': ['Hombre', 'Hombre', 'Mujer', 'Mujer'],
        'estudios': ['universitario', 'secundario', 'secundario', 'universitario']}
df = pd.DataFrame(data)

# Convertir las columnas a tipo 'Categorical'
df['sexo'] = pd.Categorical(df['sexo'])
df['estudios'] = pd.Categorical(df['estudios'])

# Crear una tabla de contingencia usando crosstab
contingency_table = pd.crosstab(df['sexo'], df['estudios'])
```

estudios	secundario	universitario
sexo		
Hombre	1	1
Mujer	1	1

Tablas de Contingencia (2)

Veamos ahora para el dataset iris:

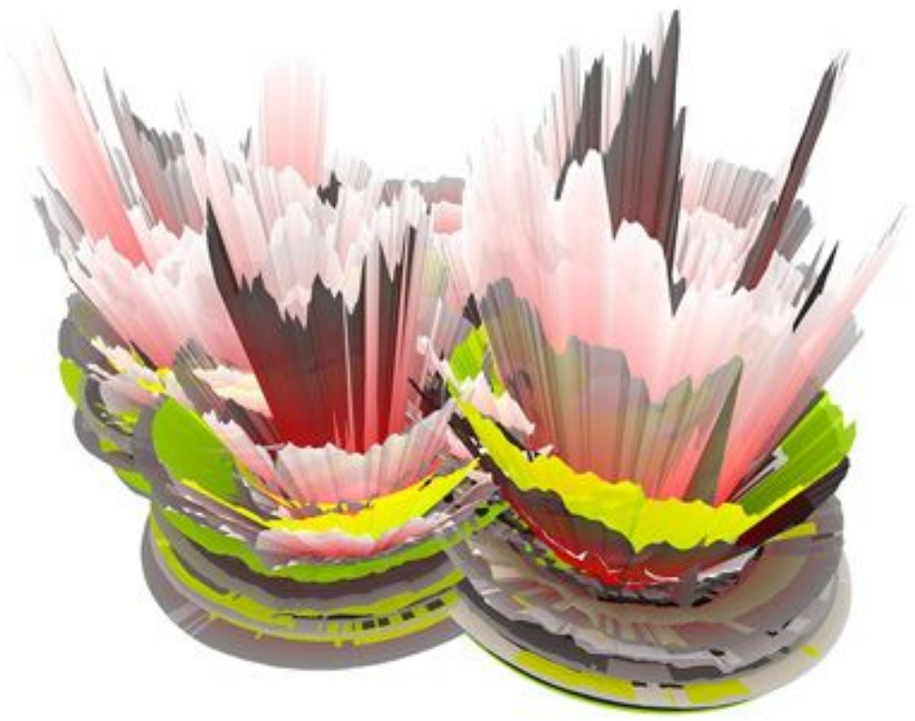
```
# Para el propósito de este ejemplo, vamos a categorizar la 'sepal length (cm)'
# en 'corto', 'medio', 'largo' usando pd.cut para crear categorías basadas en cuantiles
iris_df['sepal_length_cat'] = pd.cut(iris_df['sepal length (cm)'], 3, labels=["corto", "medio", "largo"])

# Ahora, vamos a cruzar las categorías de 'sepal length cat' con las especies usando pd.crosstab
crosstab_result = pd.crosstab(iris_df['sepal_length_cat'], iris_df['species'])

# Obteniendo:
```

species	setosa	versicolor	virginica
sepal_length_cat			
corto	47	11	1
medio	3	36	32
largo	0	3	17

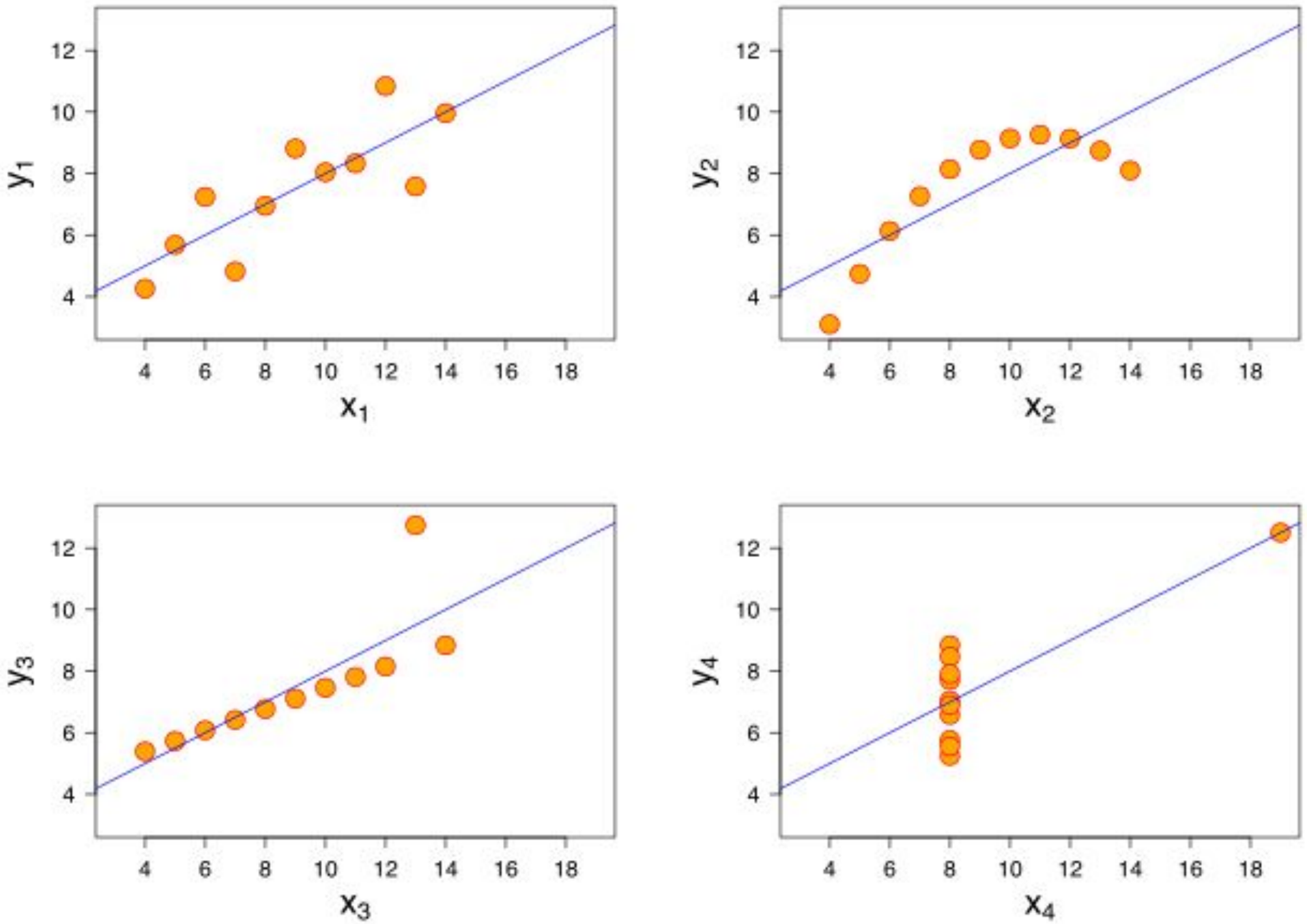
Visualización de Datos

- La visualización de datos es la transformación de un dataset a un formato visual que permita a las personas identificar las características y las relaciones entre sus elementos (columnas y files).

- La visualización permite que las personas reconozcan patrones o tendencias en base a su criterio o conocimiento en el dominio de aplicación.

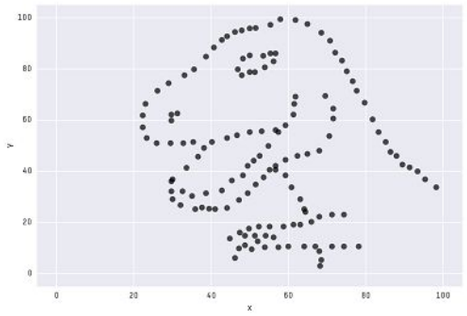


Visualización de Datos

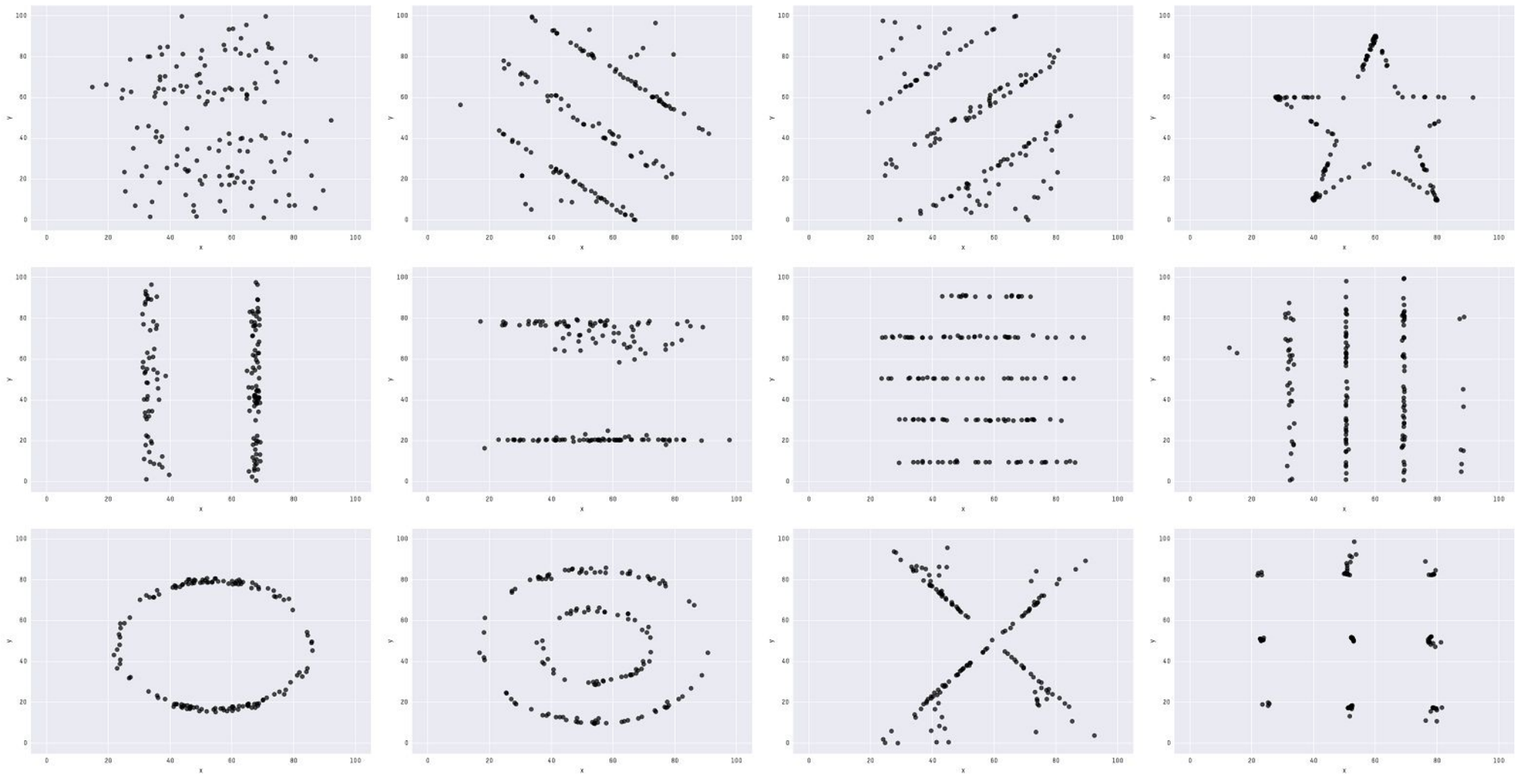


Fuente: Anscombe's quartet

Visualización de Datos



```
X Mean: 54.26
Y Mean: 47.83
X SD  : 16.76
Y SD  : 26.93
Corr. : -0.06
```



Fuente: [Same Stats, Different Graphs]

Matplotlib

- Matplotlib es una biblioteca de gráficos en Python que permite la creación de una amplia gama de gráficos y figuras estáticos, animados e interactivos.

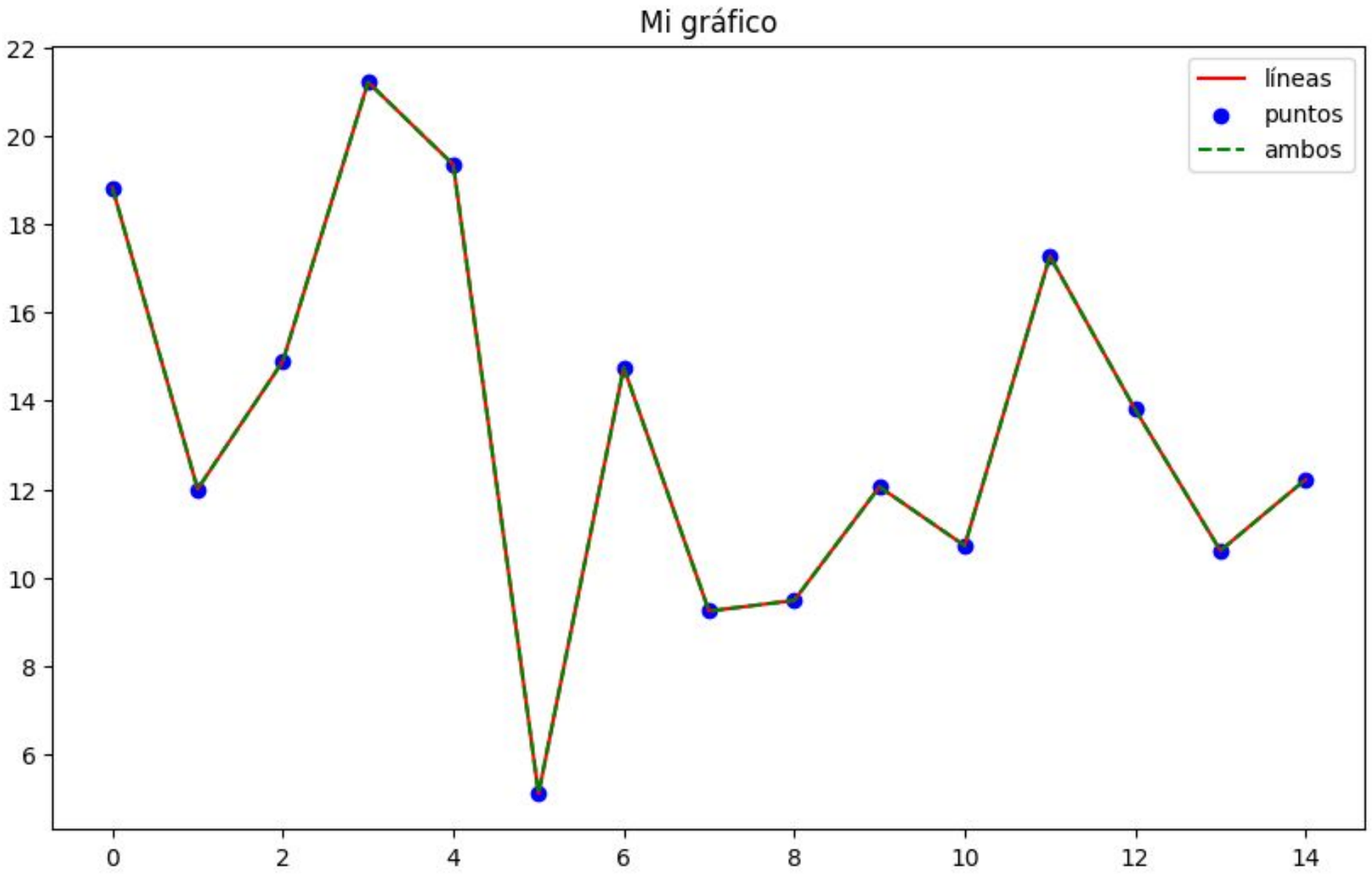
- Se integra bien con pandas y NumPy, facilitando la visualización de datos de DataFrames y arrays, respectivamente.

- Contiene módulos y funciones para trazar una variedad de gráficos, desde histogramas y gráficos de dispersión hasta gráficos complejos en 3D.
- Sin embargo, para gráficos más elaborados existen otras librerías como plotly y seaborn

Graficando en Python

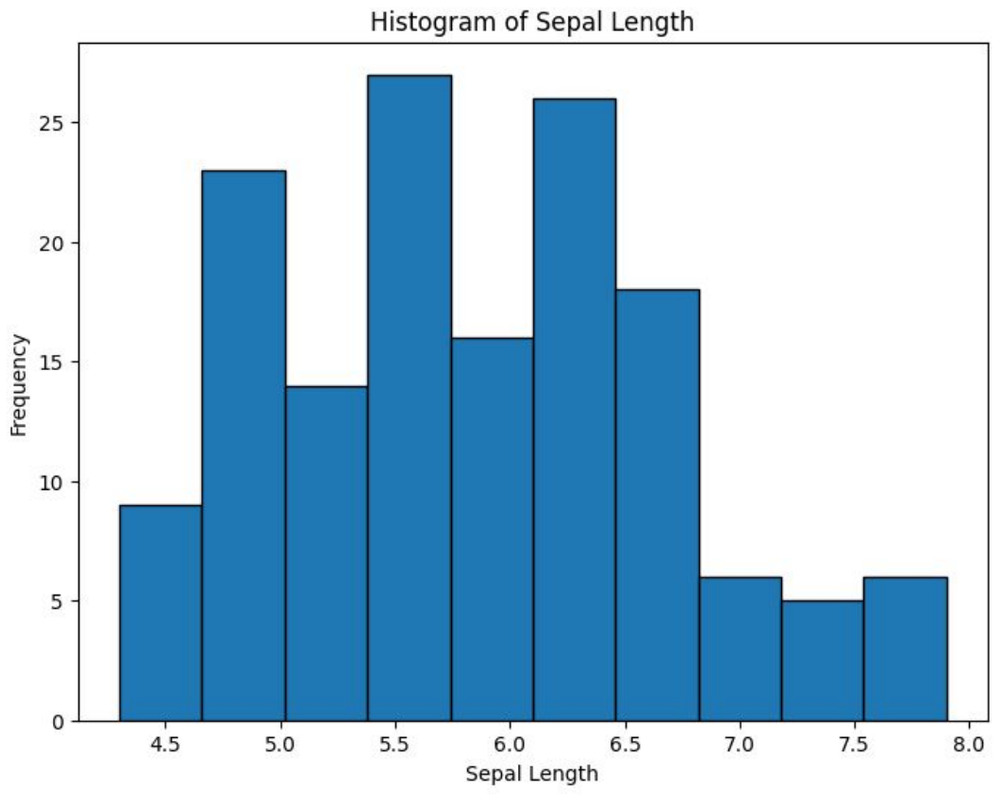
- En matplotlib para graficar utilizamos la clase pyplot.
- Pyplot permite realizar distintas visualizaciones.
- Veamos un ejemplo:

```
import matplotlib.pyplot as plt
import numpy as np

# Establece la semilla aleatoria para reproducibilidad.
np.random.seed(0)

# Genera datos aleatorios utilizando la distribución normal con media 10 y desviación estándar 5
data = np.random.normal(10, 5, 15)

# Crear el plot
plt.figure(figsize=(10, 6))

# Add the lines for each type
plt.plot(data, color='red', label='líneas')                # Línea roja
plt.scatter(range(len(data)), data, color='blue', label='puntos')  # Puntos azules
plt.plot(data, 'g--', label='ambos')                       # Línea discontinua verde que muestra
ambos.

# Título y legendas
plt.title('Mi gráfico')
plt.legend(loc='upper right')

# mostrar el plot
plt.show()
```


Mi gráfico
líneas
puntos
ambos
22
20
18
16
14
12
10
8
6
0
2
4
6
8
10
12
14

Histogramas

- Muestran la distribución de los valores de una variable.
- Los valores de los elementos se dividen en contenedores (bins) y se crean gráficos de barra por cada contenedor.
- La altura de cada barra indica el número de elementos o frecuencia del contenedor.
- En pyplot se crean con el comando *hist*.



Histogramas (2)

Código:

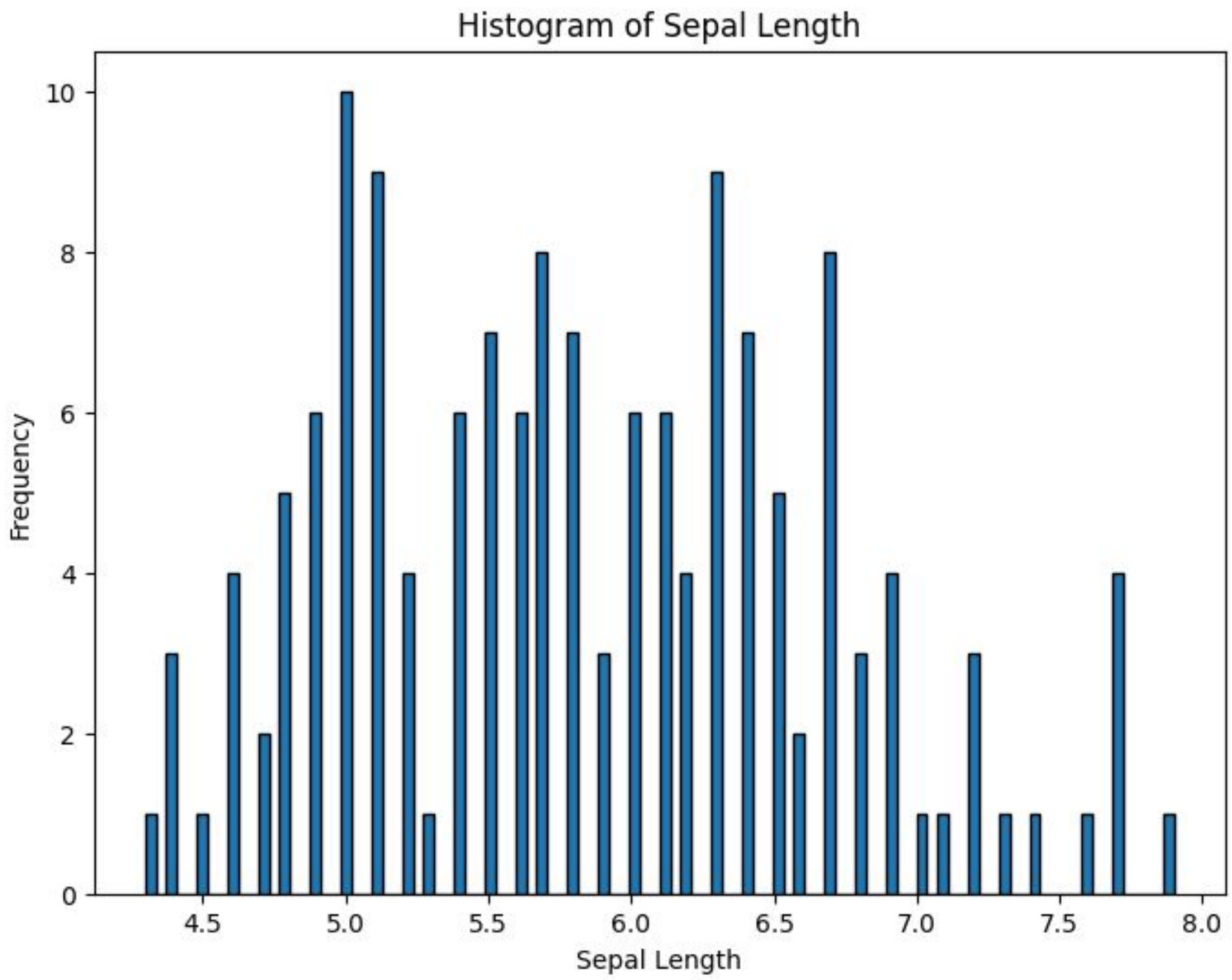
```
import matplotlib.pyplot as plt


# Selecciona la columna sepal length
sepal_length = iris_df['sepal length (cm)']

# Crea un histograma de sepal length
plt.figure(figsize=(8, 6))
plt.hist(sepal_length, edgecolor='black', )

# Título y legendas
plt.title('Histogram of Sepal Length')
plt.xlabel('Sepal Length')
plt.ylabel('Frequency')


plt.show()
```

Histogramas (3)

- La forma del histograma depende del número de contenedores.
- En pyplot se puede definir esa cantidad con el parámetro *bins*.

```
plt.figure(figsize=(8, 6))
plt.hist(sepal_length, edgecolor='black', bins=100)

# Título y legendas
plt.title('Histogram of Sepal Length')
plt.xlabel('Sepal Length')
plt.ylabel('Frequency')

# Show the plot
plt.show()
```

Histogramas (4)



Densidad

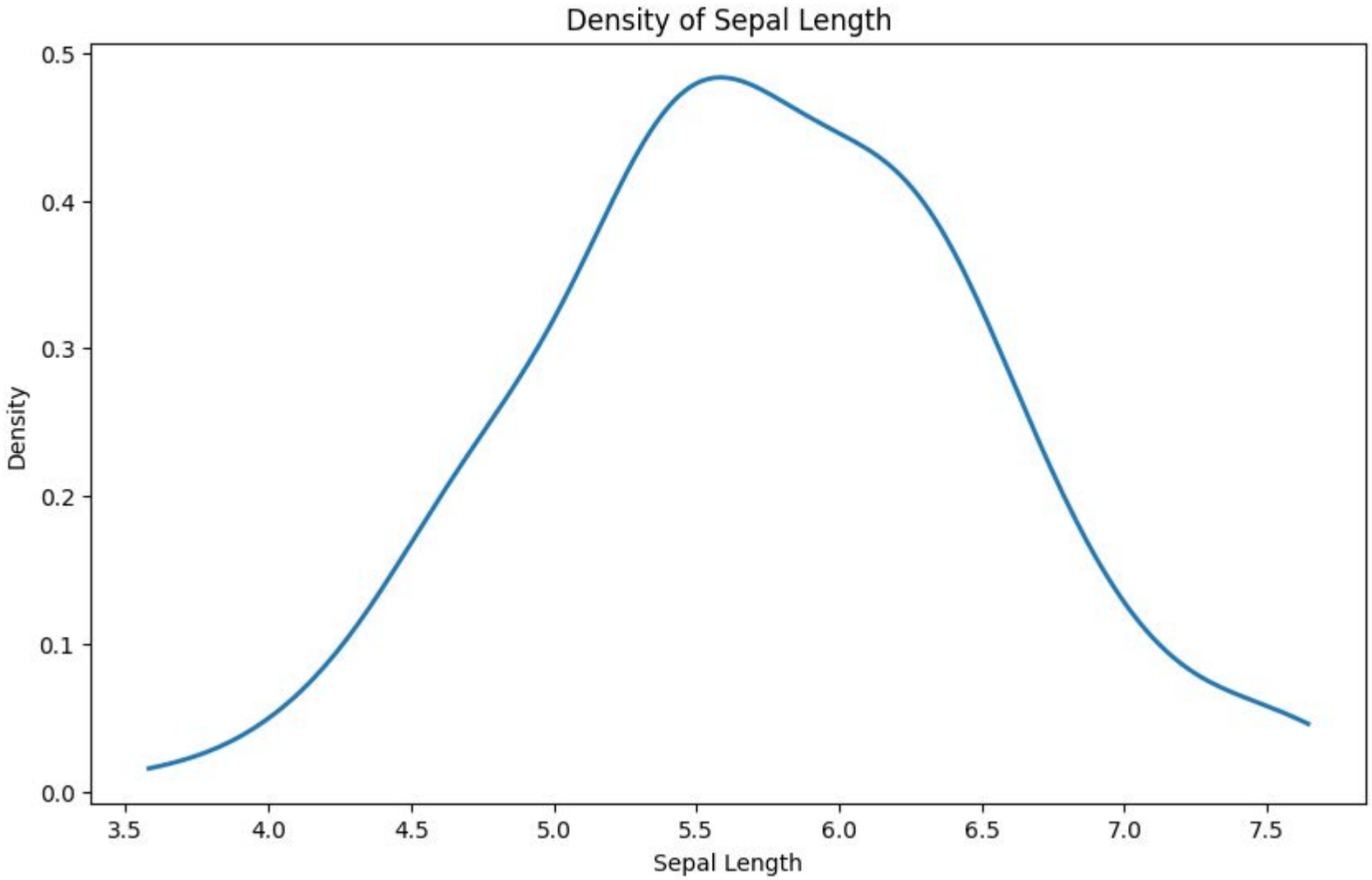
- Otra forma de visualizar cómo se distribuyen los datos es estimando una densidad.

- Se calculan usando técnicas estadísticas no paramétricas llamadas estimación de densidad de **kernel**.

- La densidad es una versión suavizada del histograma y nos permite determinar más claramente si los datos observados se comportan como una densidad conocida ej: normal.

Densidad (1)

- En `python` se crean con el comando *gaussian_kde* de scipy para luego visualizarlas con el comando *plot*.



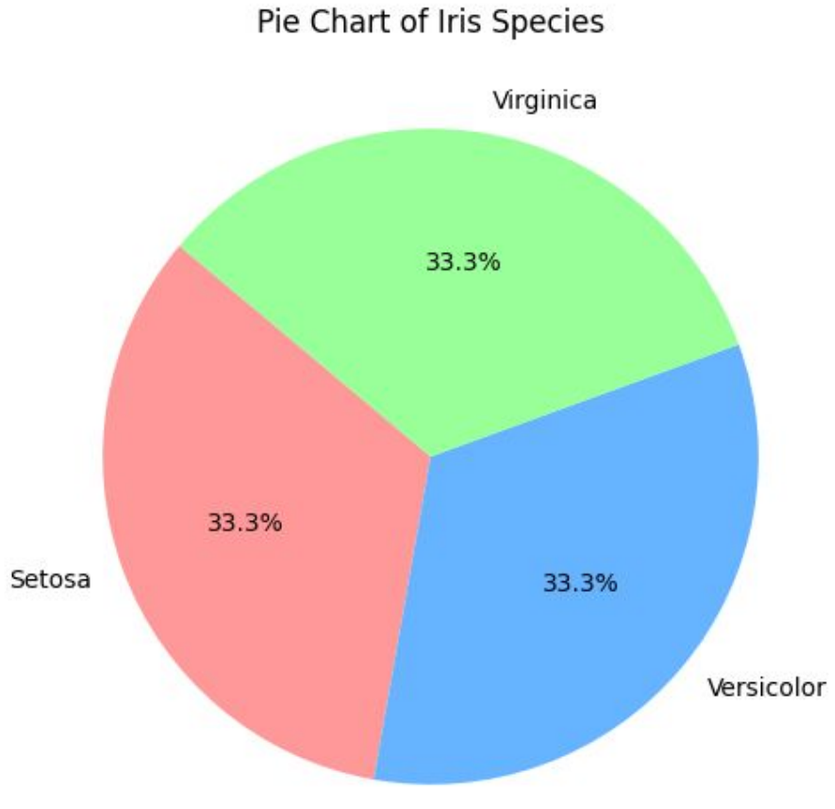
Densidad (2)

Código:

```
from scipy import stats

# Calcula la densidad de la función
density = stats.gaussian_kde(sepal_length)
# Genera un rango de valores para los cuales queremos estimar la densidad.
xs = np.linspace(sepal_length.min(), sepal_length.max(), 200)
# Calcula la densidad para cada valor en xs.
density_values = density(xs)
# Crea la figura
plt.figure(figsize=(10, 6))
plt.plot(xs, density_values, linewidth=2)
# Título y legendas
plt.xlabel('Sepal Length')
plt.ylabel('Density')
plt.title('Density of Sepal Length')
```

Gráficos de Torta o Pie Charts

- Los gráficos de torta, gráficos circulares o pie charts representan la frecuencia de los elementos en un círculo.
- Cada elemento tiene una participación proporcional a su frecuencia relativa.
- Se usan generalmente para variables categóricas.
- No se recomienda mucho su uso, pues pueden entregar información engañosa.



Gráficos de Torta o Pie Charts (2)

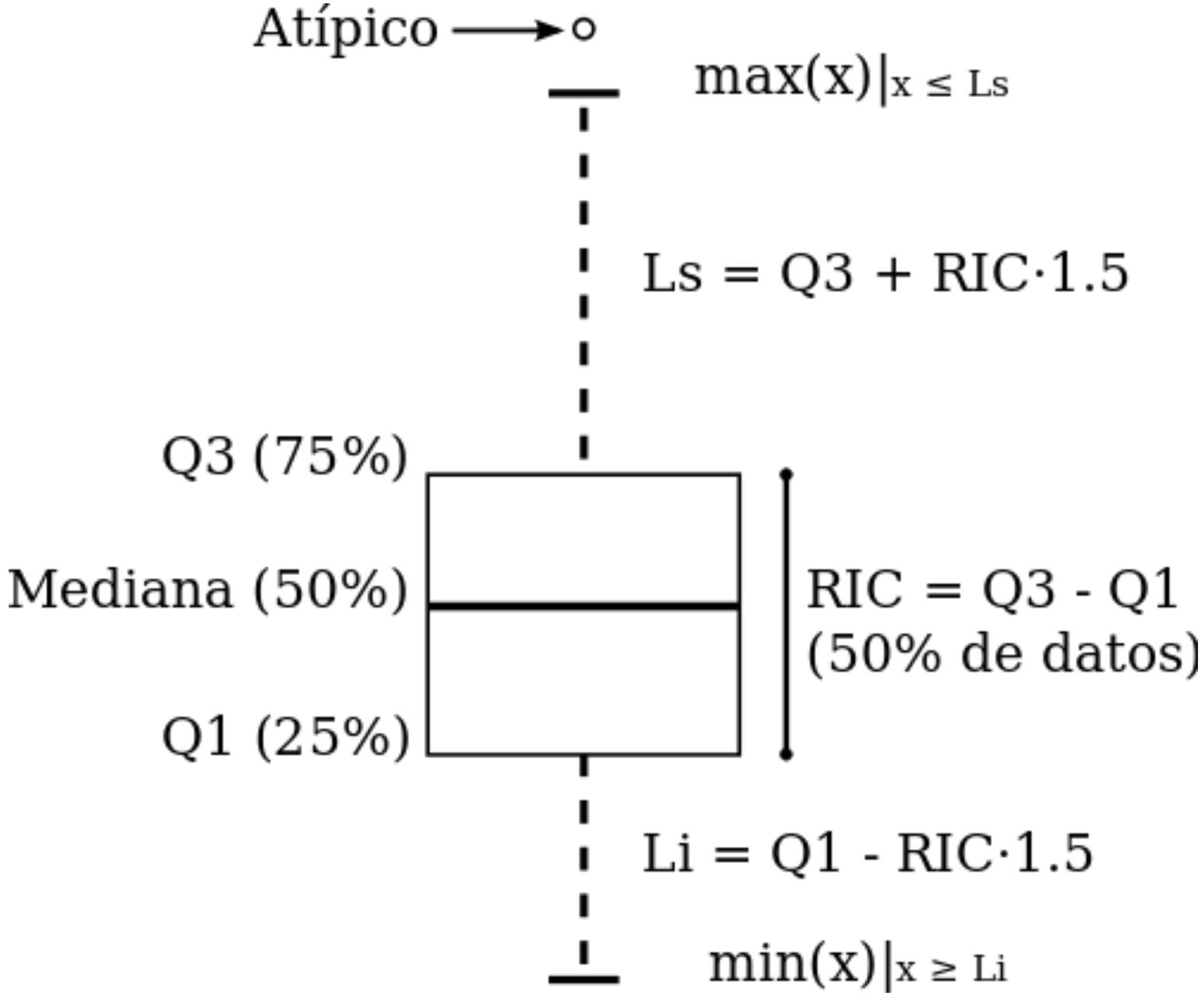
Código:

```
species_counts = iris_df["species"].value_counts()
species_names = iris_df['species'].cat.categories.tolist()

plt.figure(figsize=(8, 6))
plt.pie(species_counts, labels=species_names, autopct='%1.1f%%', startangle=140,
colors=['#ff9999','#66b3ff','#99ff99'])
# Título
plt.title('Pie Chart of Iris Species')
plt.show()
```

Boxplots

- Los Boxplots o diagramas de caja se construyen a partir de los percentiles.
- Se construye una rectángulo usando entre el primer y el tercer cuartil (Q1 y Q3).
- La altura del rectángulo es el rango intercuartil RIC (Q3 – Q1).
- La mediana es una línea que divide el rectángulo.
- Cada extremo del rectángulo se extiende con una recta o brazos de largo Q1 – 1,5∗RIC para la recta inferior y Q3 + 1,5∗RIC para la recta superior.
- Los valores más extremos que el largo de los brazos son considerados atípicos.
- El boxplot nos entrega información sobre la simetría de la distribución de los datos.
- Si la mediana no está en el centro del rectángulo, la distribución no es simétrica.
- Son útiles para ver la presencia de valores atípicos u outliers.

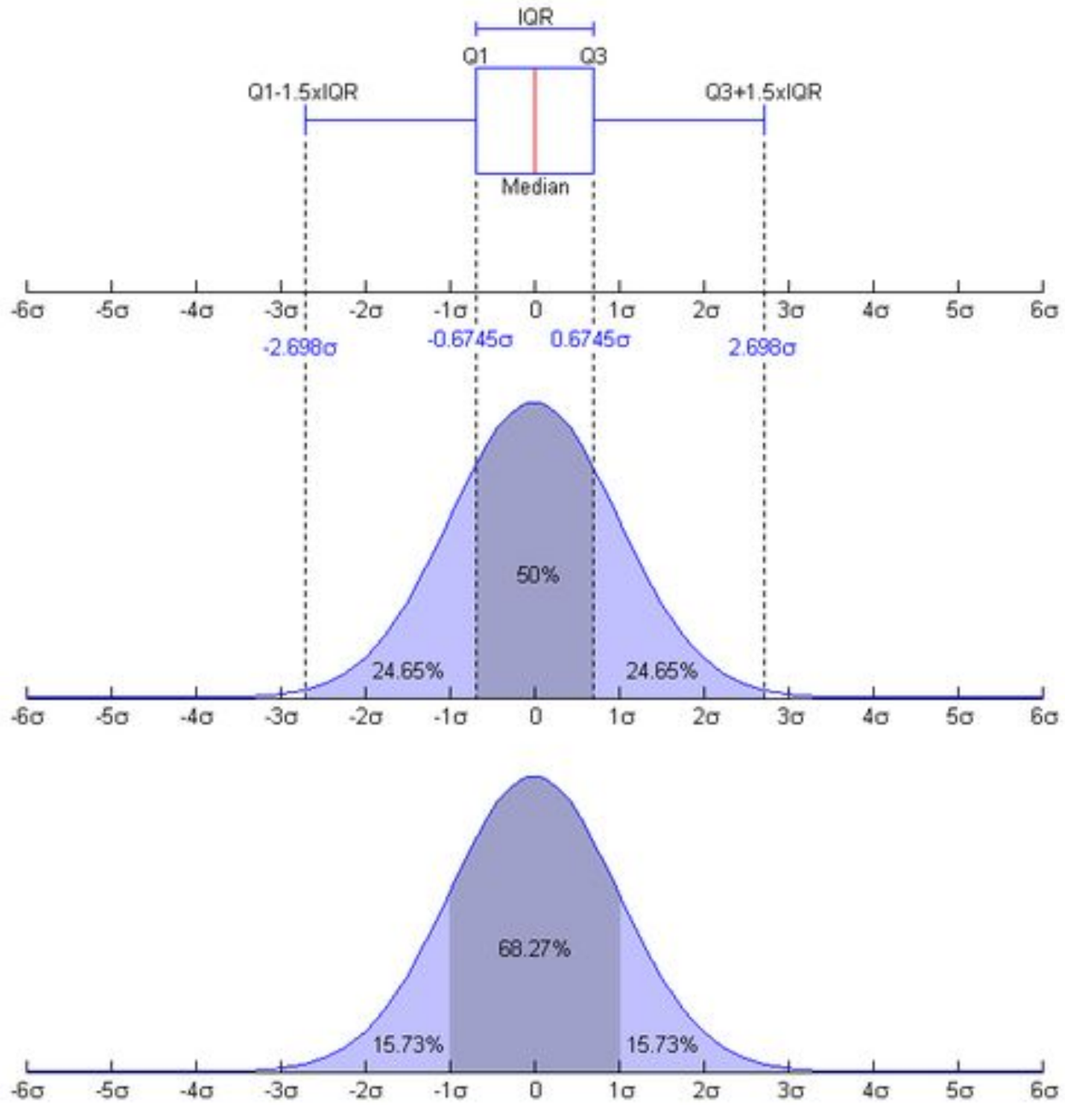
Boxplots (2)



Fuente:
http://commons.wikimedia.org/wiki/File:Boxplot.svg

Boxplots (3)

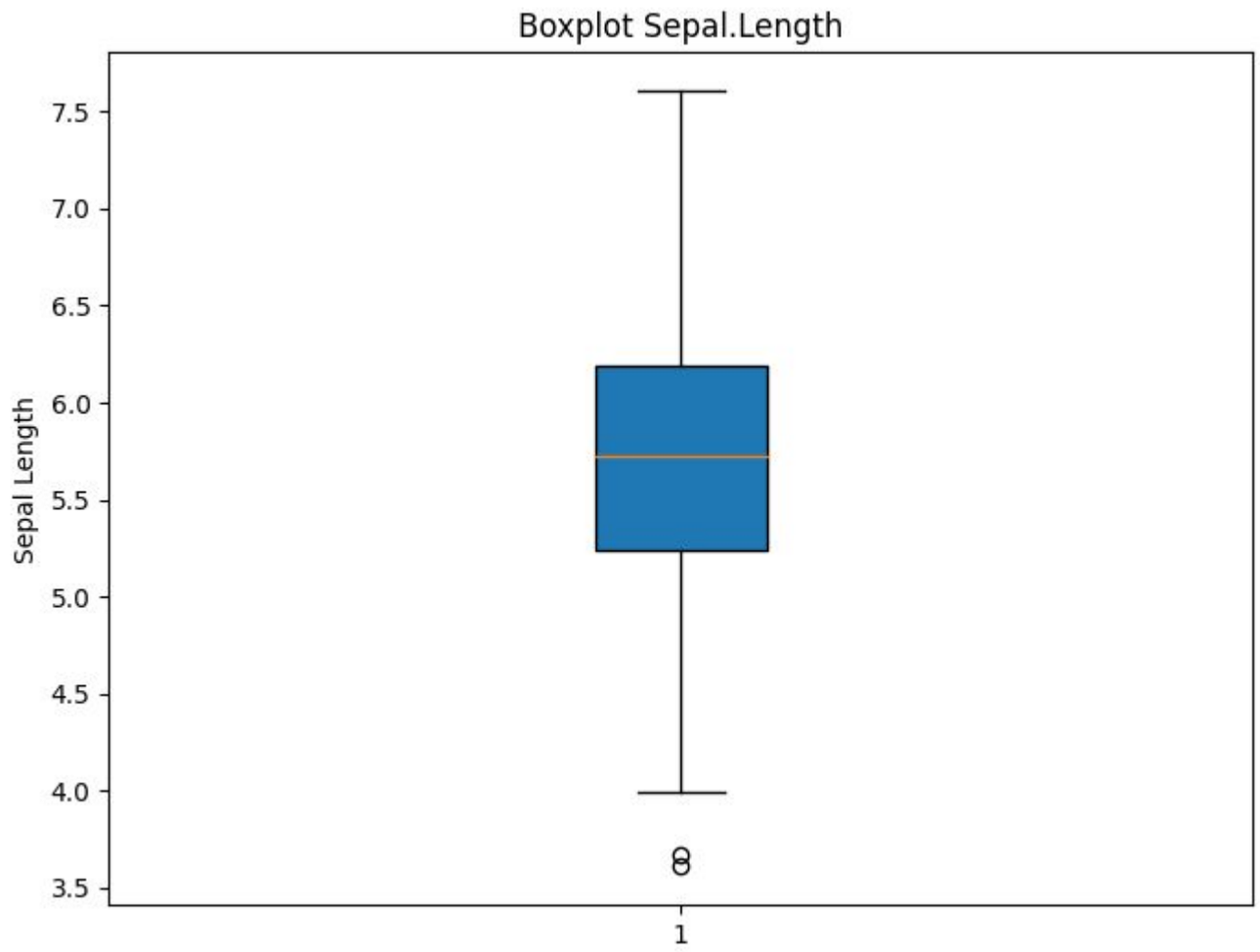
El largo de los brazos así como el criterio para identificar valores atípicos se basa en el comportamiento de una normal.



Boxplots (4)

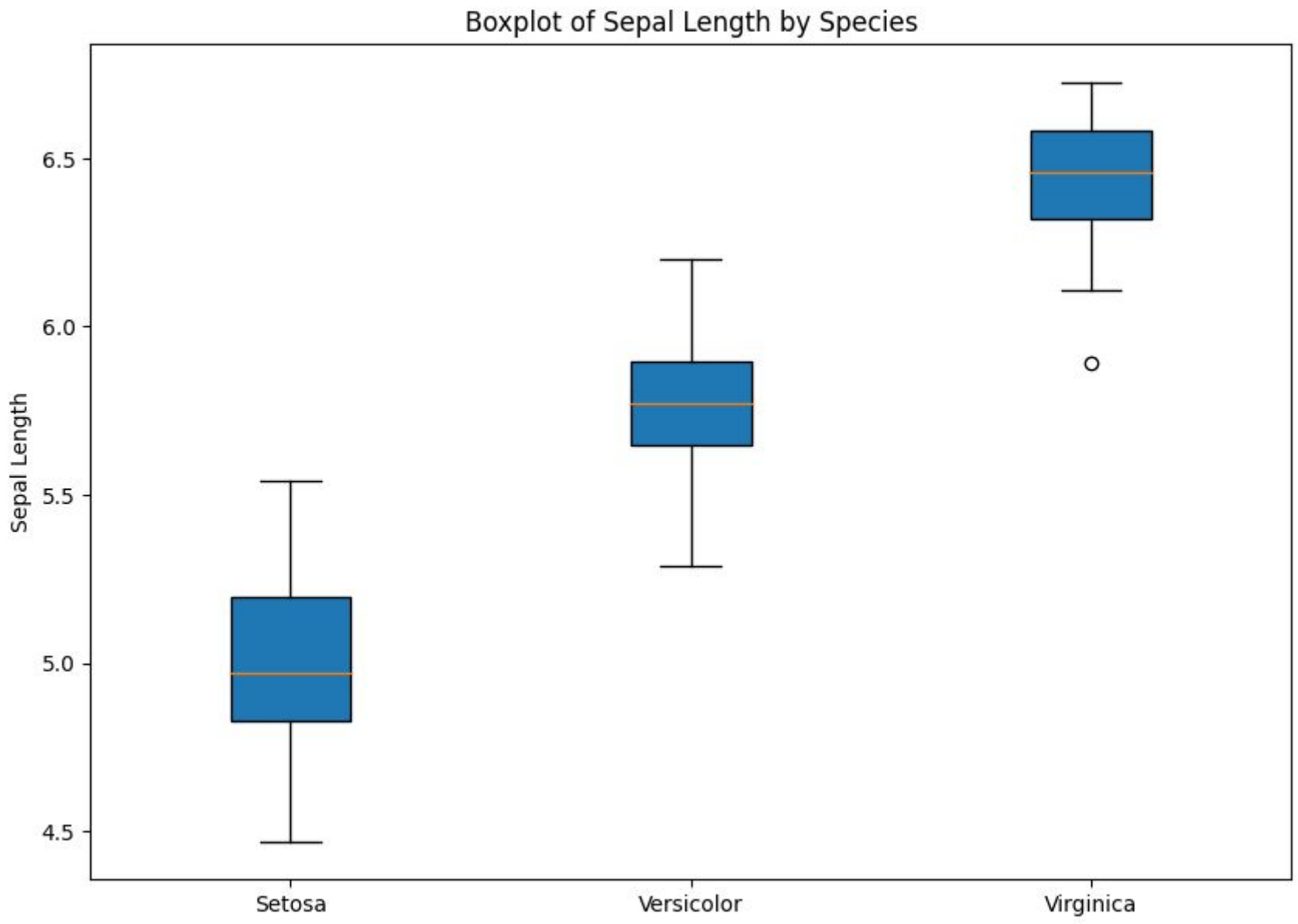
- En matplotlib los boxplots se grafican con el función *boxplot*:

```
# Crea a boxplot
plt.figure(figsize=(8, 6))
plt.boxplot(sepal_length, patch_artist=True)

# Títulos y leyendas
plt.ylabel('Sepal Length')
plt.title('Boxplot Sepal.Length')

# Muestra the plot
plt.show()
```



Boxplots (5)

- Si tenemos una variable categórica es útil crear un boxplot para cada valor de ésta con respecto a otra variable númerica:



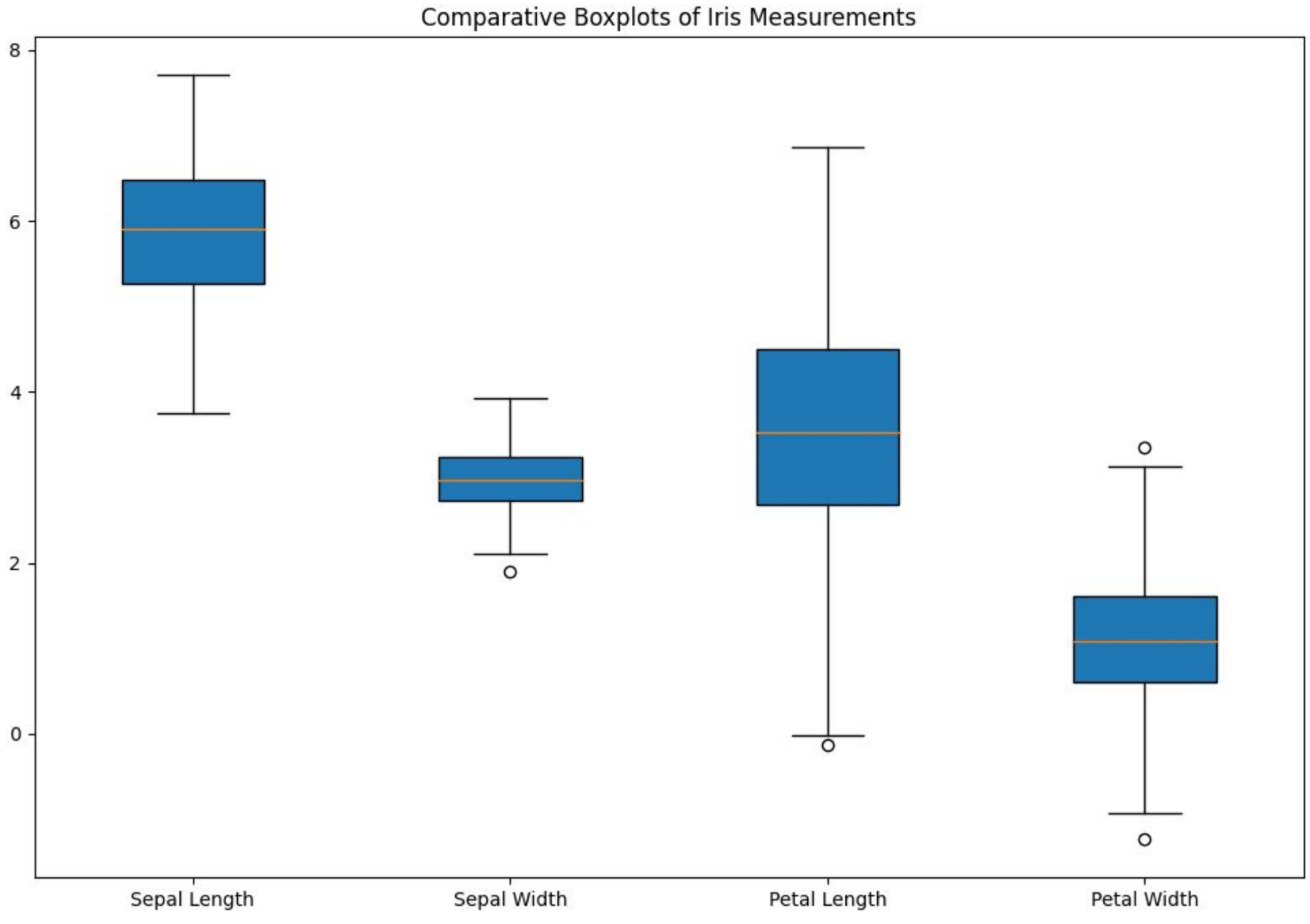
Boxplots (6)

Codigo:

```
# Combina los ejemplos en un sólo dataset
data_to_plot = [setosa_sample, versicolor_sample, virginica_sample]
species = ['Setosa', 'Versicolor', 'Virginica']

# Crea un boxplot agrepado
plt.figure(figsize=(10, 7))
plt.boxplot(data to plot, patch artist= True)

# Agrega las etiquetas
plt.xticks([1, 2, 3], species)
plt.ylabel('Sepal Length')

# Título
plt.title('Boxplot of Sepal Length by Species' )

# Mostrar the plot
plt.show()
```

Boxplots (7)

- También podemos podemos comparar varios boxplots en un mismo gráfico:



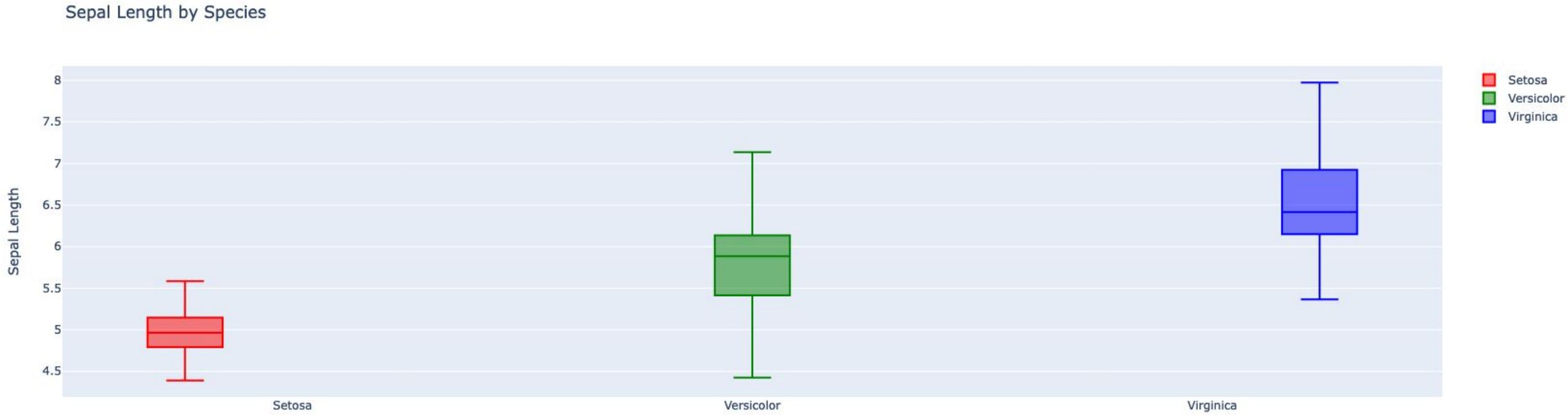
Boxplots (8)

Codigo:

```
# Combina los ejemplos en un sólo dataset
data_to_plot = [sepal_length_sample, sepal_width_sample,
petal_length_sample, petal_width_sample]
measurements = ['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal
Width']

# Crea boxplots comparativos
plt.figure(figsize=(12, 8))
plt.boxplot(data_to_plot, patch_artist=True)

# Agregar etiquetas
plt.xticks([1, 2, 3, 4], measurements)
plt.title('Comparative Boxplots of Iris Measurements')
```

Boxplots (9)

- Ahora usando plotly:



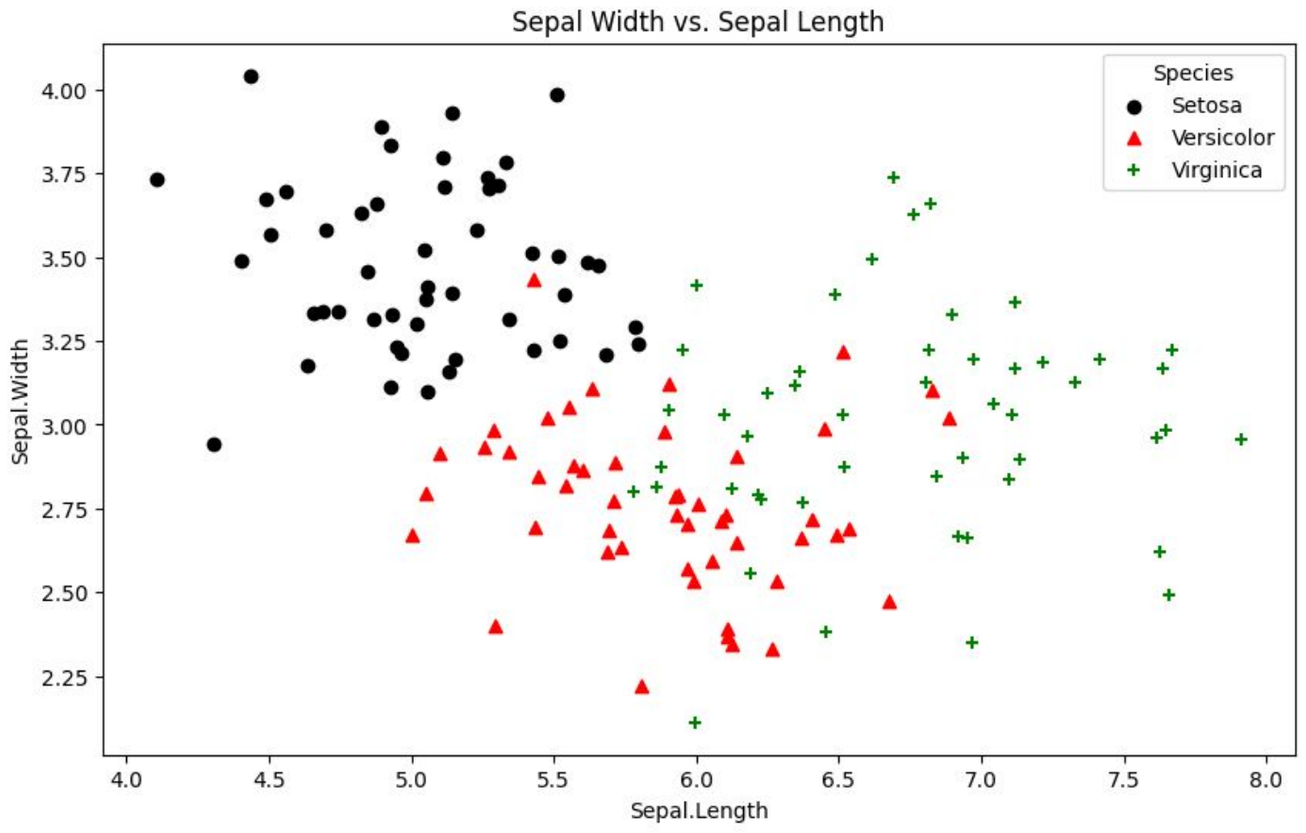
Boxplots (10)

Código:

```
# Crear una figura
fig = go.Figure()

# Agregar los boxplots para cada especie
fig.add_trace(go.Box(y=setosa_sepal_length, name='Setosa',
marker_color='red'))
fig.add_trace(go.Box(y=versicolor_sepal_length, name='Versicolor',
marker_color='green'))
fig.add_trace(go.Box(y=virginica_sepal_length, name='Virginica',
marker_color='blue'))

# Agregar títulos y etiquetas
fig.update_layout(
    title='Sepal Length by Species',
    yaxis_title='Sepal Length',
    boxmode='group'  # Agrupar juntas las cajas de los diferentes
trazos para cada categoría
)
```

Diagramas de Dispersión

- Los diagramas de dispersión o scatter plots usan coordenadas cartesianas para mostrar los valores de dos variables numéricas del mismo largo.
- Los valores de los atributos determinan la posición de los elementos.
- Otros atributos pueden usarse para definir el tamaño, la forma o el color de los objetos.
- En `matplotlib` podemos graficar un scatterplot de dos variables numéricas usando el comando scatter*(x,y)*, que sería y vs x.



Diagramas de Dispersión (2)

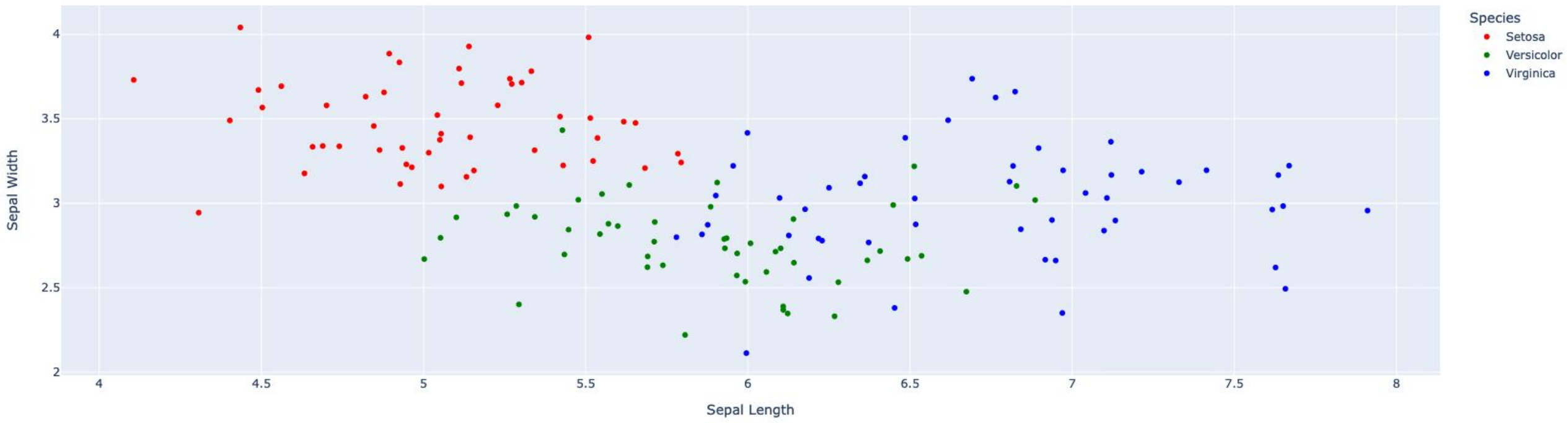
Código:

```
# Plot
plt.figure(figsize=(10, 6))

# Setosa Plot
plt.scatter(setosa_sepal_length, setosa_sepal_width, c='black', marker='o', label='Setosa')
# Versicolor Plot
plt.scatter(versicolor_sepal_length, versicolor_sepal_width, c='red', marker='^',
label='Versicolor')
# Virginica Plot
plt.scatter(virginica_sepal_length, virginica_sepal_width, c='green', marker='+',
label='Virginica')

# Legendas
plt.legend(title='Species')

# Título y ejes
plt.xlabel('Sepal.Length')
plt.ylabel('Sepal.Width')
plt.title('Sepal Width vs. Sepal Length')
```

Diagramas de Dispersión (3)

Lo mismo usando plotly:

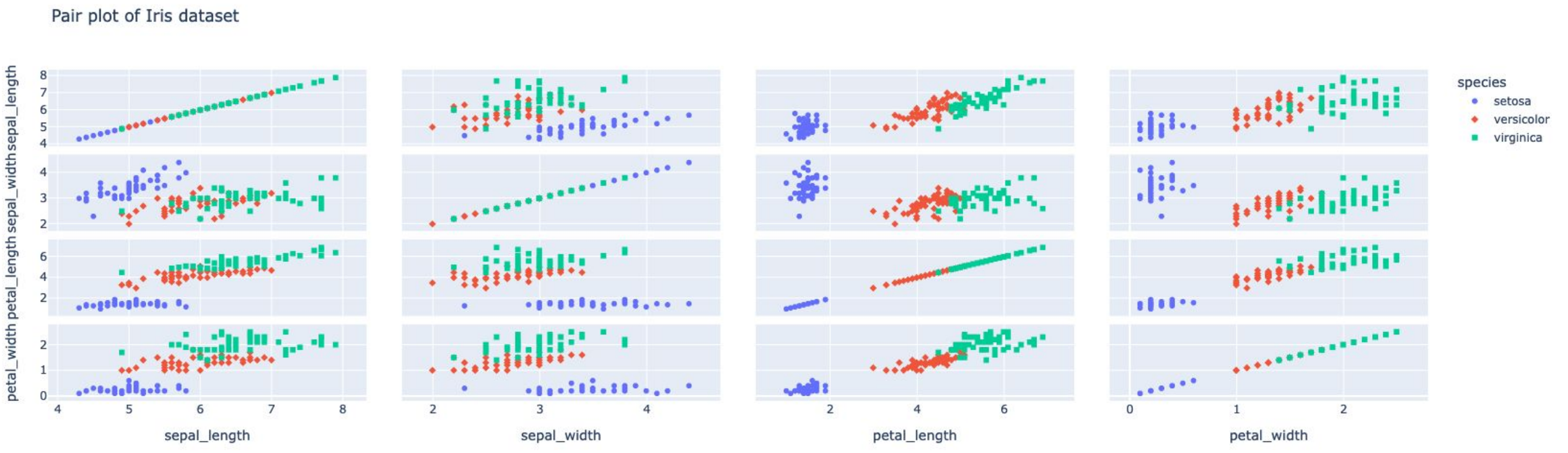
```
fig = px.scatter(iris_df, x='sepal length (cm)', y='sepal width (cm)',
color='Species', color_discrete_map={'Setosa': 'red', 'Versicolor':
'green', 'Virginica': 'blue'})
```



Diagramas de Dispersión (4)

Para graficar todos los pares de las 4 variables del dataset iris usando un color y un carácter distinto para cada especie, no es posible hacerlo directamente en `matplotlib`, pero en `plotly`:

```
fig = px.scatter_matrix(df,
    dimensions=['sepal_length', 'sepal_width', 'petal_length', 'petal_width'],
    color='species',
    symbol='species',
    title='Pair plot of Iris dataset'
)
```

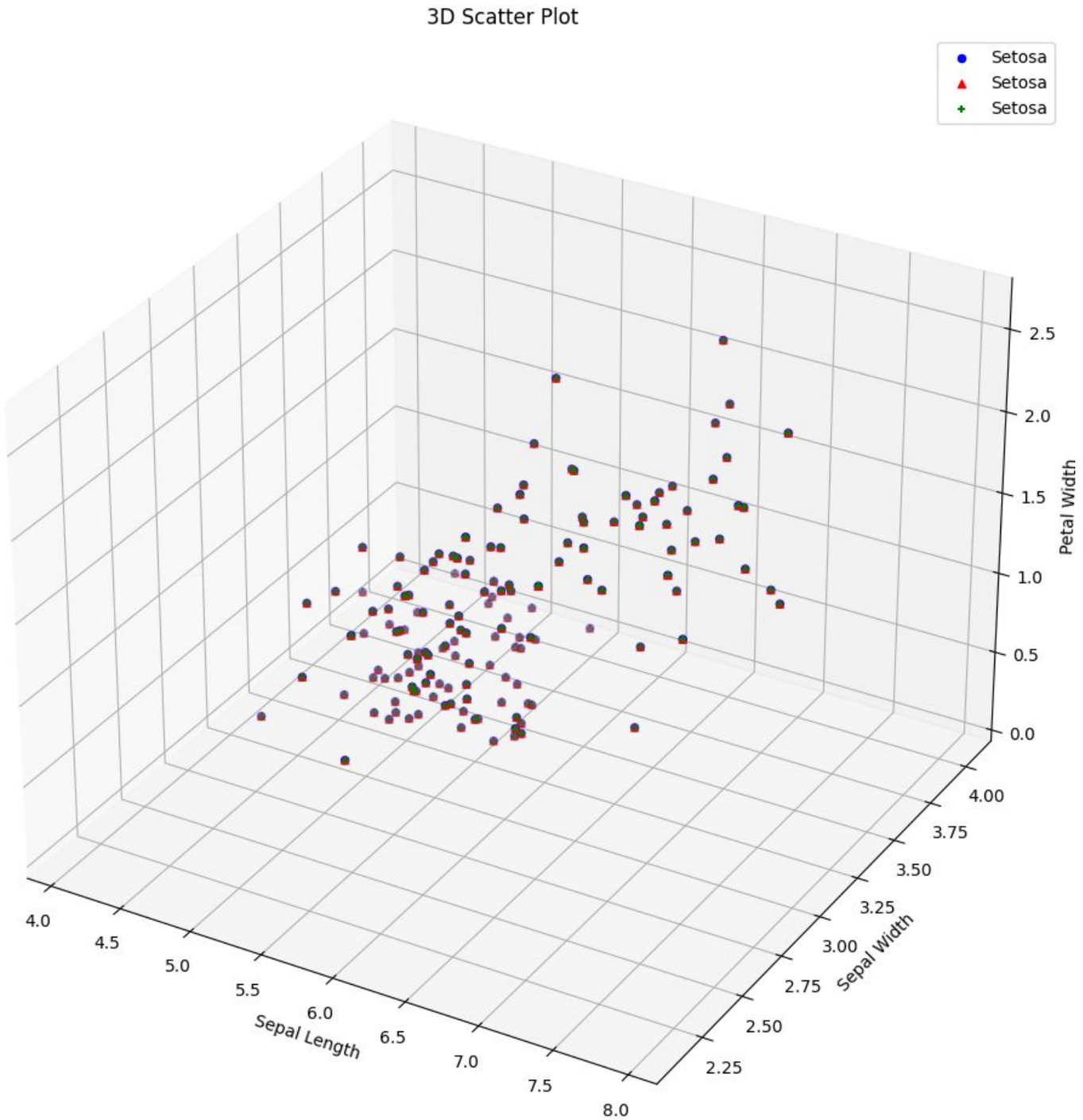


Diagramas de Dispersión (5)

- También se pueden crear scatterplots en tres dimensiones usando matplotlib.
- Para esto hay que importar el submódulo `mpl_toolkits.mplot3d`
- Un scatterplot 3d para el ancho del pétalo, el largo del sépalo y el ancho del sépalo:

```
# Plot the values
colors = {'Setosa': 'b', 'Versicolor': 'r', 'Virginica': 'g'}
markers = {'Setosa': 'o', 'Versicolor': '^', 'Virginica': '+'}

for s, c, m in zip(species, colors.values(), markers.values()):
    ax.scatter(x, y, z, c=c, marker=m, label=s)

ax.set_xlabel('Sepal Length')
ax.set_ylabel('Sepal Width')
ax.set_zlabel('Petal Width')

plt.legend()
plt.title('3D Scatter Plot')
```

Diagramas de Dispersión (6)



Gráficos de Coordenadas Paralelas

- Los gráficos de coordenadas paralelas ofrecen una alternativa para visualizar datos multidimensionales.

- En vez de usar ejes perpendiculares (x-y-z) usamos varios ejes paralelos entre sí.

- Cada atributo es representado por uno de los ejes paralelo con sus respectivos valores.

- Los valores de los distintos atributos son escalados para que cada eje tenga la misma altura.

- Cada observación representa una línea que une los distintos ejes de acuerdo a sus valores.

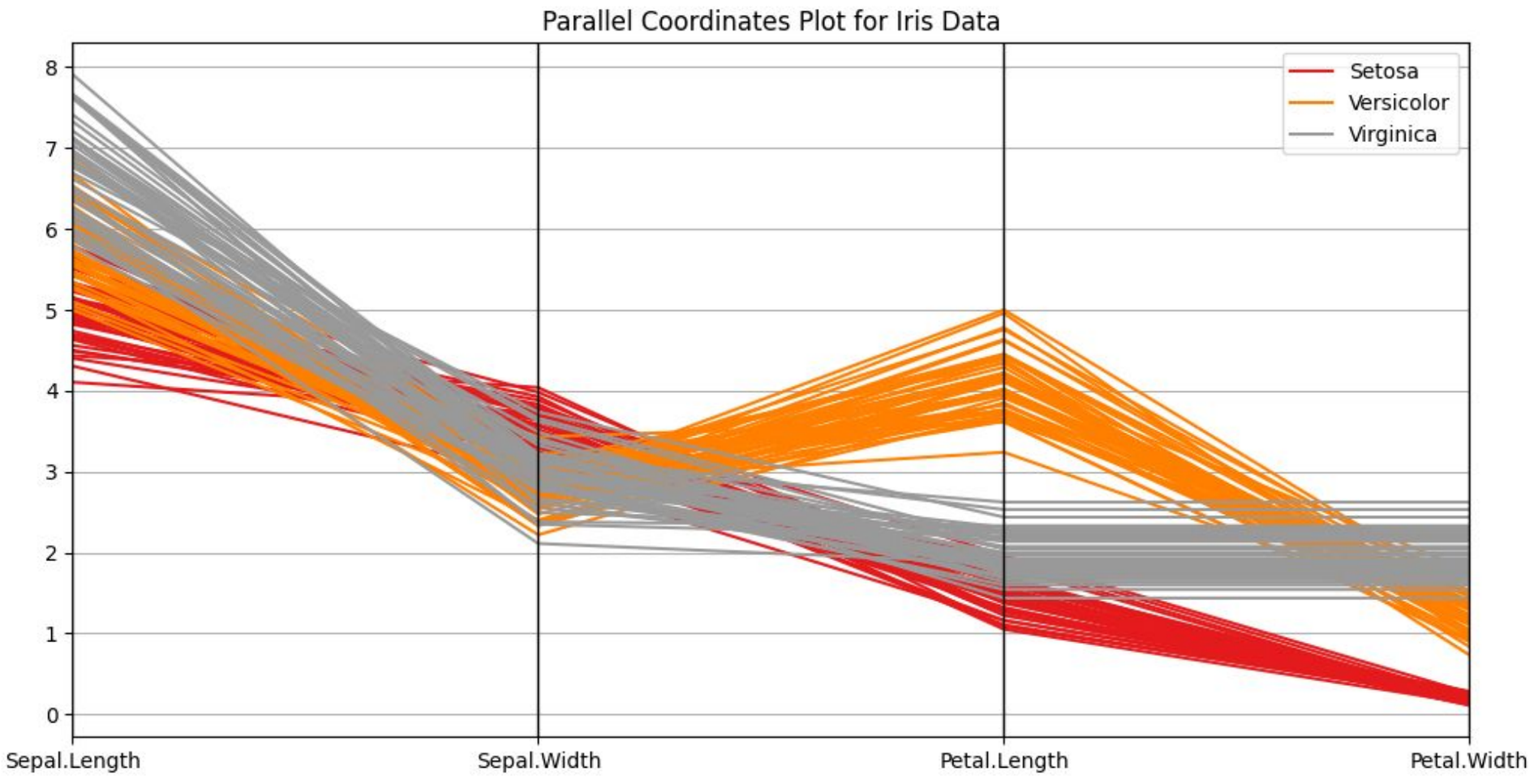
- De esta manera, objetos similares entre sí tienden a agruparse en líneas con trayectoria similar.

- En muchas ocasiones es necesario realizar un reordenamiento de los ejes para poder visualizar un patrón.

Gráficos de Coordenadas Paralelas (2)

- En Python podemos crear gráficos de coordenadas paralelas con el comando *`parallel_coordinades`* de la librería `pandas` y utilizando `matplotlib` para visualizar con `pyplot`.

```
pd.plotting.parallel_coordinates(data, 'Species', colormap=plt.get_cmap("Set1"))

plt.title('Parallel Coordinates Plot for Iris Data')
```



Gráficos de Estrellas o Radar Charts

- Los gráficos de estrella o gráficos radiales representan cada ejemplo como estrella.

- Cada variable se representa como un eje que parte del centro de la estrella y se extiende hacia afuera en diferentes direcciones, como las agujas del reloj.

- El valor de cada variable se representa mediante una línea o segmento que conecta el centro de la estrella con el punto correspondiente en el eje.

- El tamaño de cada línea o segmento en relación con el centro de la estrella refleja el valor reescalado de la variable.

- Si una línea es más larga, indica un valor más alto, mientras que una línea más corta indica un valor más bajo.

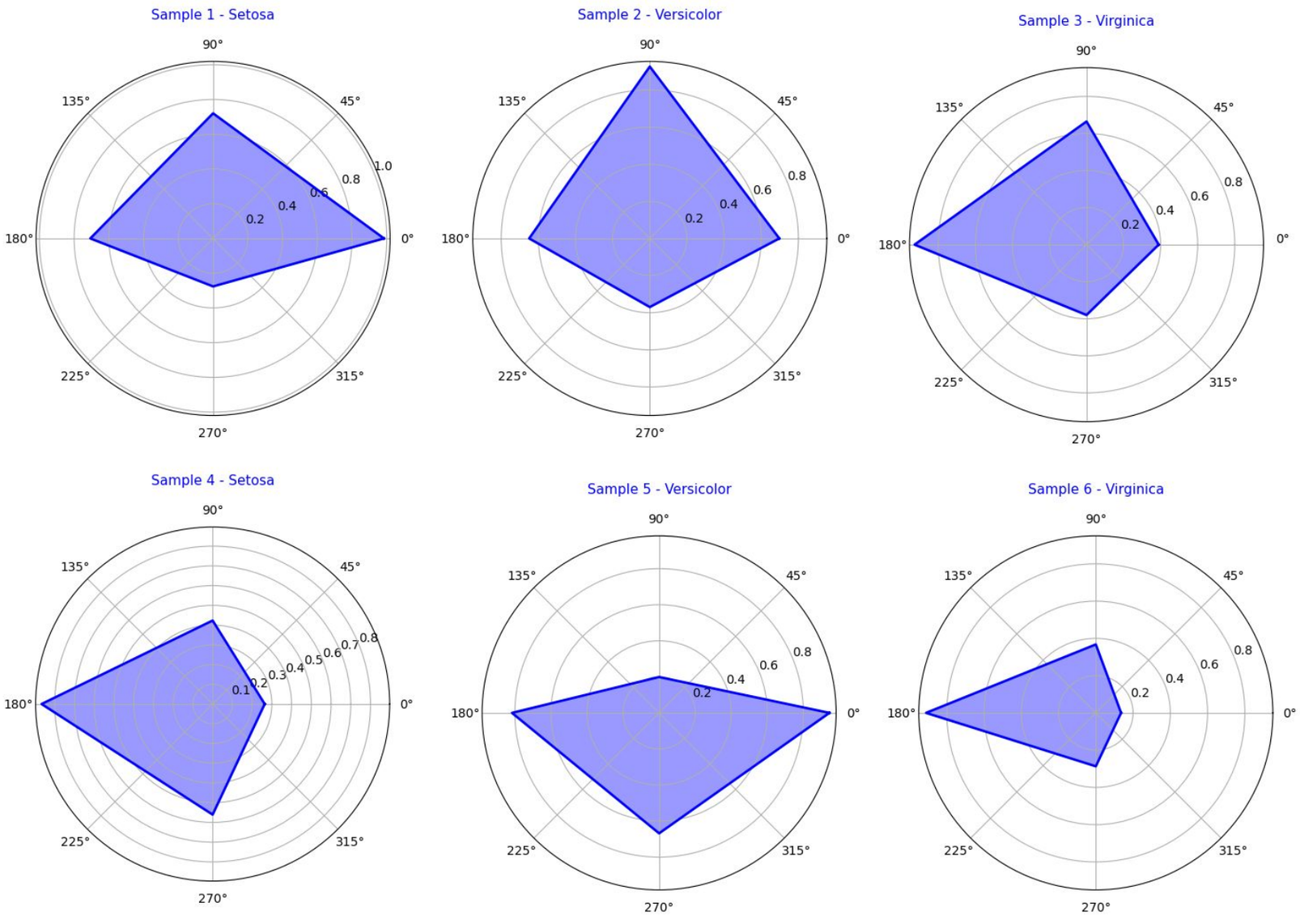
- Al unir todos los puntos correspondientes a cada variable, se forma un polígono que representa el perfil o características del objeto o ejemplo que se está representando.

- Sirve para comparar objetos o detectar valores atípicos.

Gráficos de Estrellas o Radar Charts

- Estos gráficos son útiles para comparar objetos o detectar valores atípicos porque permiten visualizar rápidamente cómo se distribuyen los valores en múltiples variables.

  Ejemplo:



Gráficos de Estrellas o Radar Charts

```
# Número de variables
num_vars = len(data.columns) - 1

# Calcular los ángulos para cada eje
angles = [n / float(num_vars) * 2 * pi for n in range(num_vars)]
angles += angles[:1]  # to complete the loop

# Función Radar chart
def make_spider(row, title, color):
    values = data.iloc[row].drop('Species').values.flatten().tolist()
    values += values[:1]
    ax.plot(angles, values, color=color, linewidth=2, linestyle='solid')
    ax.fill(angles, values, color=color, alpha=0.4)

    # Beautify el plot
    plt.title(title, size=11, color=color, y=1.1)

# Inicializa spider plot
fig = plt.figure(figsize=(10, 15))

# Crea un radar chart for cada plot
for i in range(data.shape[0]):
    ax = plt.subplot(3, 2, i+1, polar=True)
    make_spider(i, f'Sample {i+1} - {data.iloc[i]["Species"]}', 'blue')
```

dcc

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

www.dcc.uchile.cl

/ DCCUCHILE