dcc

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Exploratory Data Analysis

Exploratory Data Analysis

- Exploratory data analysis (EDA) encompasses a set of techniques to understand/summarize the most important characteristics of a data set or dataset.

- This methodology was pioneered by the statistician John Tukey.

- It is mainly based on two criteria: Summary statistics and data visualization.

- In this class we will see both types of techniques, as well as their application in Python for known datasets.

John Tukey

Python and Anaconda

- Python is a general purpose interpreted programming language. Open Source: https://www.python.org/

- It is a full-featured language that supports different programming paradigms such as object-oriented programming.

- Anaconda is a Python distribution that includes a large collection of pre-installed science packages.

- It has a configuration of environments for data science and machine learning to be simpler and faster.

NumPy and SciPy

- NumPy:
  - Provides multidimensional arrays, which are faster and less cumbersome than traditional Python lists
  - Serves as the basis for other scientific and data analysis libraries such as pandas, SciPy and scikit-learn, due to its integration and performance.
- SciPy:
  - SciPy is based on NumPy and offers additional modules for optimization, linear algebra, integration, interpolation, special functions, signal and image processing.
  - It extends the functionality of NumPy with more specialized tools.

Pandas

- It is an external Python library for data manipulation and analysis.
- It provides data structures and operations for manipulating numerical tables and time series.
- It allows a variety of data manipulation operations including:
    - Merge, join and concatenate data.
    - Filtering, grouping and transforming data.
- Pandas introduces two new data structures to Python - DataFrame and Series, inspired by R dataframes.

Scikit-Learn

- It is an external Python library that provides a wide variety of supervised and unsupervised algorithms.

- It includes functions for data preprocessing, dimensionality reduction, model selection and cross-validation, essential for data preparation and model evaluation.

- It works well with other Python libraries, such as NumPy and SciPy, and supports many data formats, allowing seamless integration into the data science workflow.

The Iris Dataset

- We will work with a well-known dataset in data analysis called Iris.
- The dataset is composed of 150 observations of iris plant flowers.
- There are three types of iris flower classes: virginica, setosa and versicolor.
- There are 50 observations of each.
- The variables or attributes measured for each flower are:
  - Flower type as a categorical variable.
  - Petal length and width in cm as numerical variables.
  - The length and width of the sepal in cm as numerical variables.

The Iris Dataset

Virginica

Setosa

Versicolor

- The dataset is available in Scikit-Learn, a Python Machine Learning library:

```
> 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')
```

Categorical Data in Pandas

- `Categoricals` are a type of pandas data that correspond to categorical variables in statistics.
- A categorical variable assumes a limited and usually fixed number of possible values (categories; levels in R).
- Examples are gender, social class, blood type, country affiliation, time of observation, or rating by Likert scales.

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

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

Summary Statistics

- Summary statistics are values that explain properties of the data.
- Some of these properties include: frequencies, measures of central tendency and dispersion.
- Examples:
  - Central tendency: mean, median, mode.
  - Dispersion: they measure the variability of the data, such as standard deviation, range, etc..
- Most summary statistics can be calculated by making a single pass through the data.

Frequency and Mode

- The frequency of an attribute value is the percentage of times it is observed.
- In pandas we can count the frequencies of occurrence of each distinct value within a column using a function *value_counts*:

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

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

- Exercise: Calculate the percentage frequencies of the previous result.

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

Frequency and Mode (2)

- The mode of an attribute is the most frequently observed value.
- In pandas, there is a specific function to calculate the mode:

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

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

- We generally use frequency and mode to study categorical variables.

Measures of Central Tendency

- These measures attempt to summarize the observed values into a single value associated with the value located in the center.
- The mean is the most common measure of central tendency for a numerical variable.
- If we have m observations it is calculated as the arithmetic mean or average.

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

Measures of Central Tendency (2)

- The biggest problem with the mean is that it is a measure that is very sensitive to outliers.
- Example: We take a random vector of mean 20 and then add a random element that comes from a much larger mean distribution.
- We see that the mean is strongly affected by noise:

```
> 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)
```

Measures of Central Tendency (3)

- We can robust the mean by eliminating a fraction of the extreme values using the **trimmed mean**.
- In python we can use the scipy library that implements trim_mean
- The second parameter defines the fraction of extreme elements to discard.
- Example: We discard 10% of the extreme values in the previous example:

```
> 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)
```

Measures of Central Tendency (4)

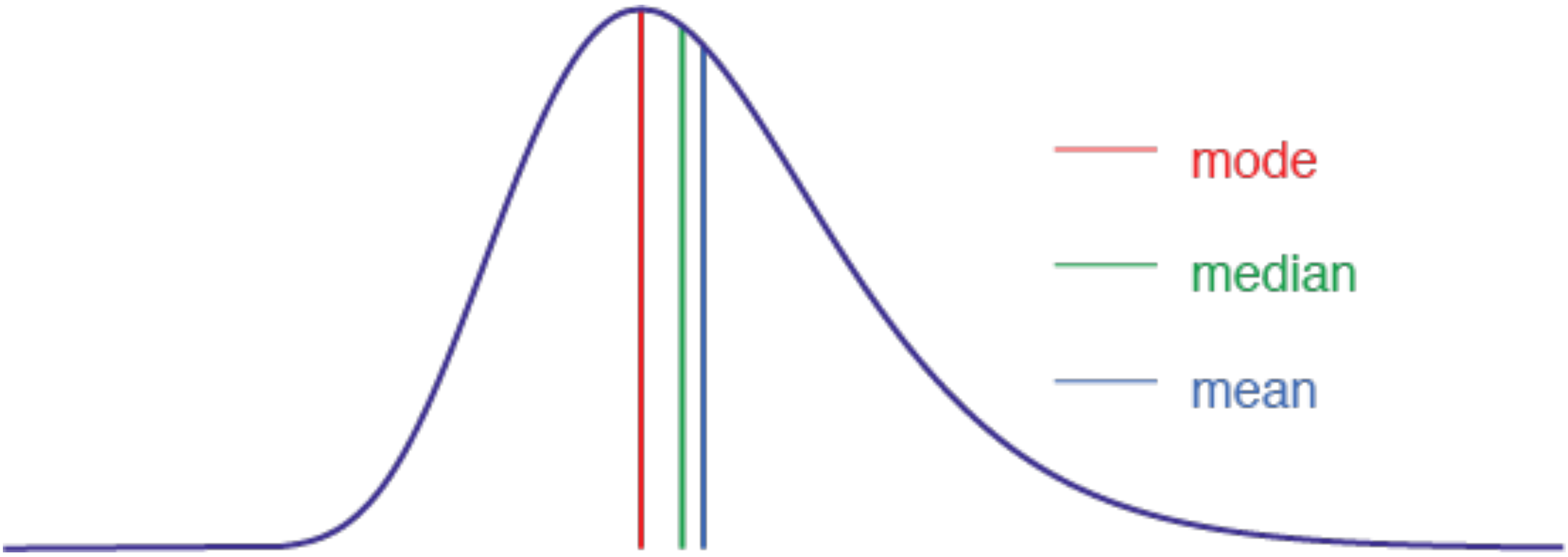
- The median represents the central position of the variable that separates the lower half and the upper half of the observations.
- Intuitively, it consists of the value where for one half of the observations all values

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

- For the above example, we see that the median is more robust to noise than the mean:

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

 (26.803434597319807, 29.500884175255894)
```

Comparison between mode, median and mean



Percentiles or Quantiles

- The k-th percentile of a numerical variable is a value such that k% of the observations are below the percentile and (100 - k) % are above this value.

- In statistics, quantiles are generally used, which are equivalent to percentiles expressed in fractions instead of percentages.

- In pandas they are calculated with numpy's quantile command:

```
> 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 or Quantiles (2)

* It is also very common to speak of quartiles which are three specific percentile

—> The first quartile Q1 (lower quartile) is the percentile with k = 25.

—> The second quartile Q2 is with k = 50 which is equivalent to the median.

—> The third quartile Q3 (upper quartile) is with k = 75.

The minimum, the three quartiles and the maximum.

```
> 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

Summarizing a 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

Exercise

- Using the `groupby` command analyze different summary statistics for the three Iris species for the four variables.

- Do you notice any differences in the different species?

```
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()
```

Answer:

```
> 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

Measures of Dispersion

- These measures tell us how different or similar the observations tend to be with respect to a particular value.
- Usually this particular value refers to some measure of central tendency.
- The range is the difference between the maximum and minimum value:

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

3.6

Measures of Dispersion (2)

- The standard deviation is the square root of the variance that measures the average squared differences of the observations with respect to the mean.

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

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

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

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

Measures of Dispersion (3)

- Like the mean, the standard deviation is sensitive to outliers.
- The most robust measures are usually based on the median.
- Let m(x) be a measure of central tendency of x (usually the median), the **average absolute deviation** (AAD) is defined as:

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

Measures of Dispersion (4)

- Exercise: program the average absolute deviation in python as a function called aad.
- The function receives a vector x and a central mean function fun.
- The absolute value is calculated with the abs command.
- Solution:

```
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)
```

Measures of Dispersion (5)

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

- Let b be a scale constant, the median absolute deviation is defined as median absolute deviation:

- In python it is calculated with the function scipy.stats.median_abs_deviation (not to be confused with the pandas function mad, which calculates mean absolute deviation, and which is also deprecated)

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

- Finally, the interquartile range (IQR) is defined as the difference between the third and the first quartile (Q3 - Q1).

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

Summary Statistics Multivariates

- To compare how one variable varies with respect to another, we use multivariate measures.

- The covariance cov(x, y) measures the degree of joint linear variation of a pair of variables x, y :

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

- Where cov(x, x) = var(x)
- In pandas it is calculated with the cov command:

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

-0.042434

Summary Statistics Multivariates (2)

- For datasets of several numerical columns one can calculate a covariance matrix.
- Each cell i,j of this matrix contains the covariance between attributes i and j.
- This matrix is symmetric.
- In pandas this matrix is obtained by applying the cov command to a data.frame of numeric variables:

```
> 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

Summary Statistics Multivariates (3)

- If two variables are independent of each other, their covariance is zero.
- To have a measure of relationship that does not depend on the scale of each variable, we use **linear correlation**.
- Linear correlation or **Pearson's** correlation coefficient r(x, y) is defined as:

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

- Linear correlation varies between -1 to 1.
- A value close to 1 indicates that as one variable grows the other also grows in a linear proportion.
- A value close to -1 indicates an inverse relationship (one variable grows and the other decreases).

Summary Statistics Multivariates (4)

- A correlation close to 0 does not imply that there cannot be a non-linear relationship between the variables.
- The linear correlation is calculated in pandas with the command corr.
- We can use it analogously to the covariance to obtain a correlation matrix.

```
> 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

Contingency Tables

- To analyze the relationship between variables of a categorical nature we use contingency tables.

- The table is filled with the co-occurrence frequencies of all pairs of values between two categorical variables.
- In pandas they are created with the `crosstab` command on attributes defined as 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

Contingency Tables (2)

Let's see now for the iris dataset:

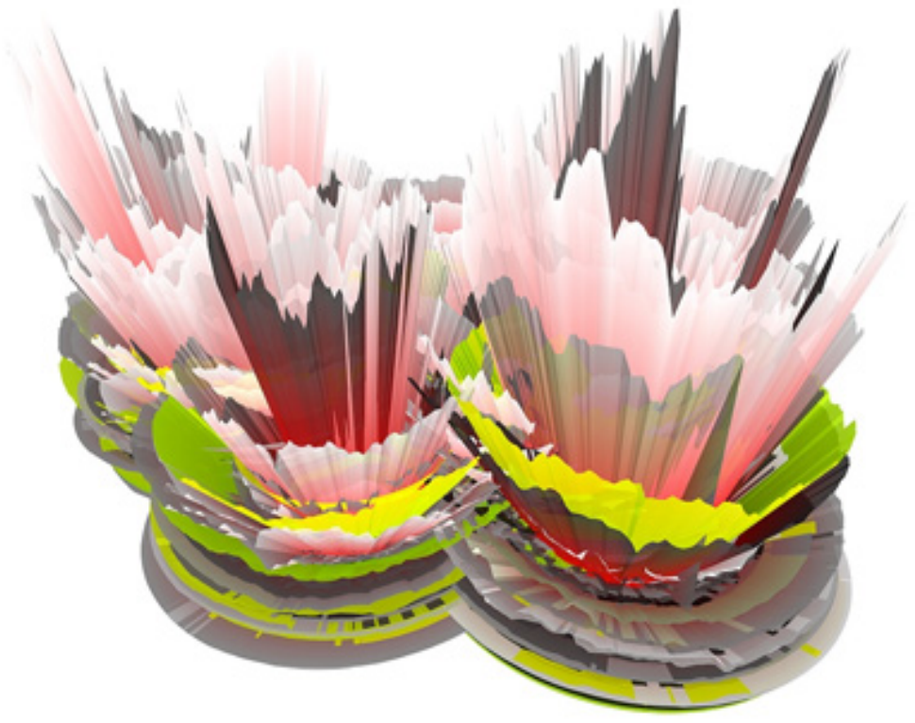
```
# Para el propósito de este ejemplo, vamos a categorizar la 'sepal length (cm)'
# en 'corto', 'medio', 'largo' usando pd.cut para crear categorías basadas en cuantiles
iris_df['sepal_length_cat'] = pd.cut(iris_df['sepal length (cm)'], 3, labels=["corto", "medio", "largo"])

# 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

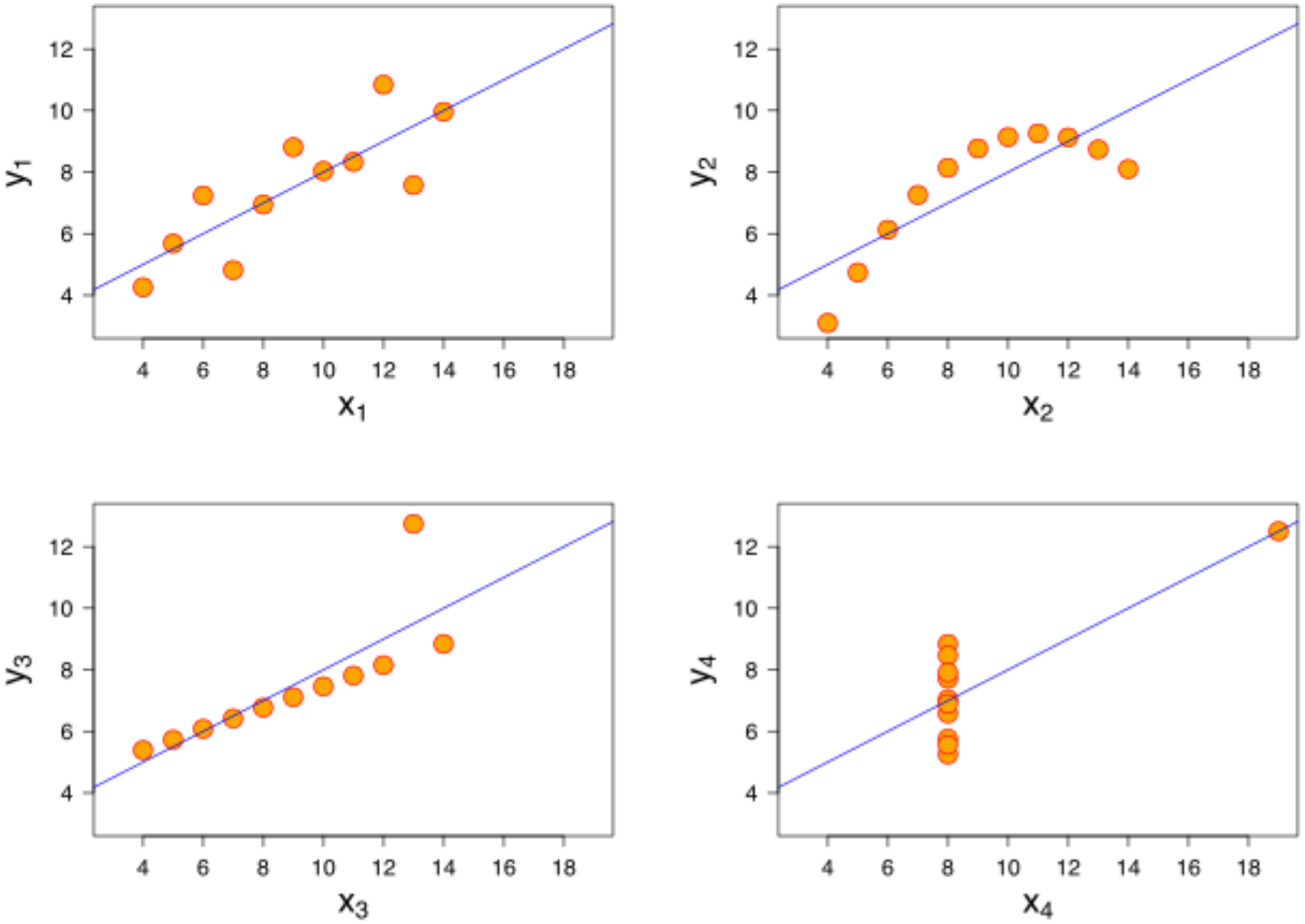
Data Visualization

- Data visualization is the transformation of a dataset into a visual format that allows people to identify the characteristics and relationships between its elements (columns and files).

- Visualization allows people to recognize patterns or trends based on their judgment or knowledge of the application domain.

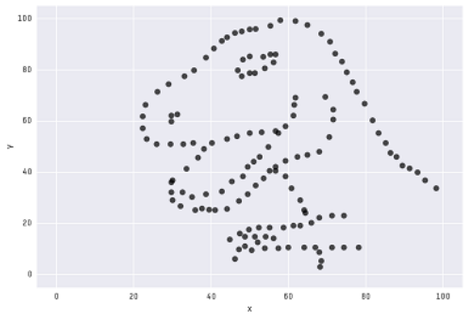


Data Visualization

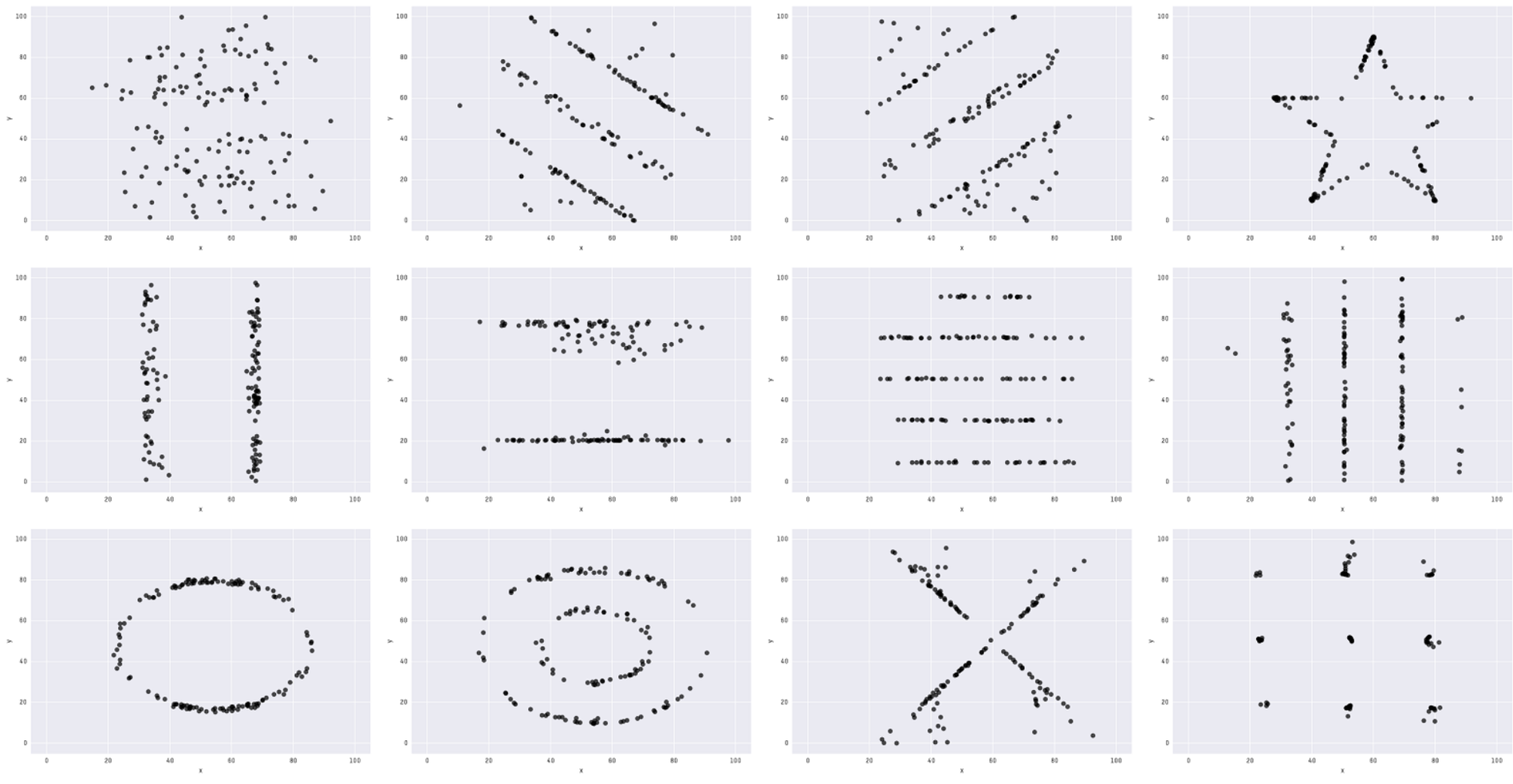


Source: [Anscombe's quartet]

Data Visualization



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



Source: [Same Stats, Different Graphs]

Matplotlib

- **Matplotlib** is a Python graphics library that allows the creation of a wide range of static, animated and interactive graphs and figures.

- It integrates well with pandas and NumPy, making it easy to visualize data from DataFrames and arrays, respectively.

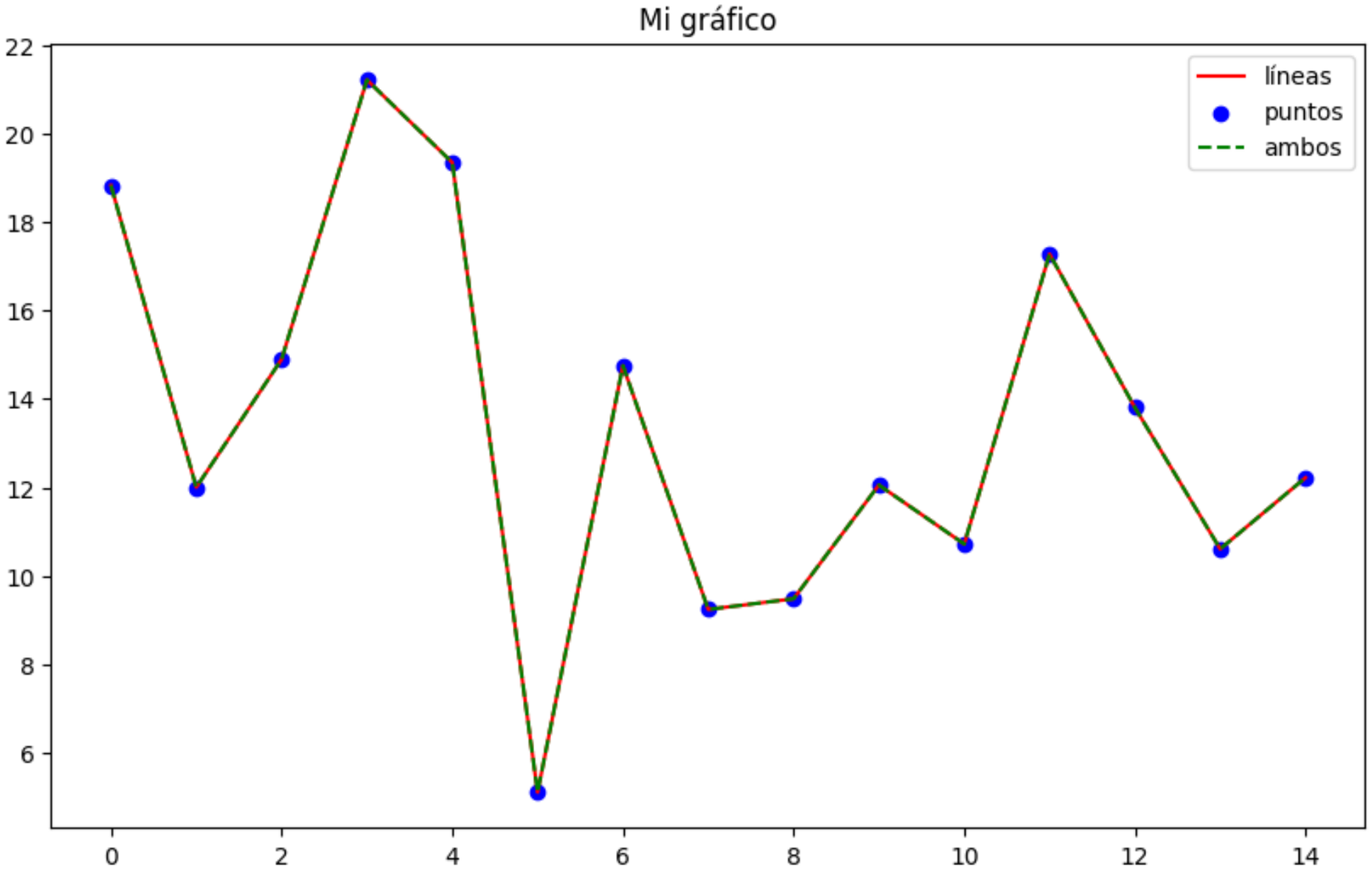
- It contains modules and functions for plotting a variety of graphs, from histograms and scatter plots to complex 3D plots.

- However, for more elaborate graphics there are other libraries such as **plotly** and **seaborn**.

Grafics in Python

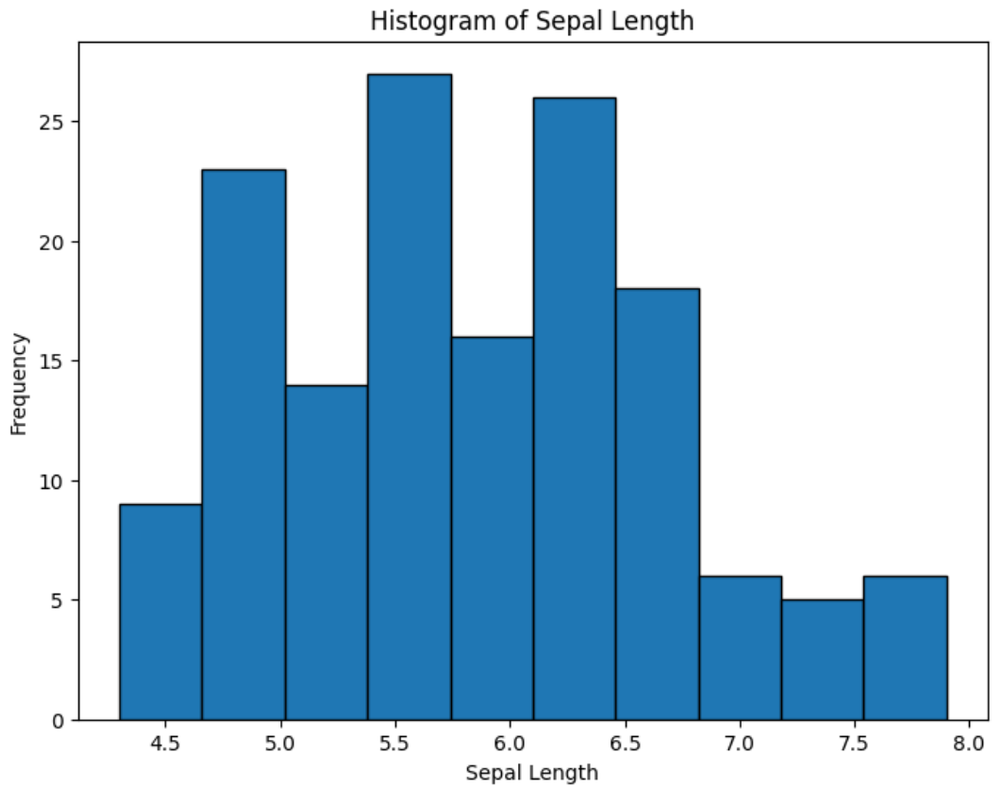
- In matplotlib for plotting we use the pyplot class.
- Pyplot allows different visualizations.
- Let's see an example:

```
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

Histograms

- They show the distribution of the values of a variable.
- The values of the elements are divided into bins and bar charts are created for each garbage can.
- The height of each bar indicates the number of elements or frequency of the garbage can.
- In pyplot they are created with the `hist` command.



Histograms (2)

Code:

```
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()
```

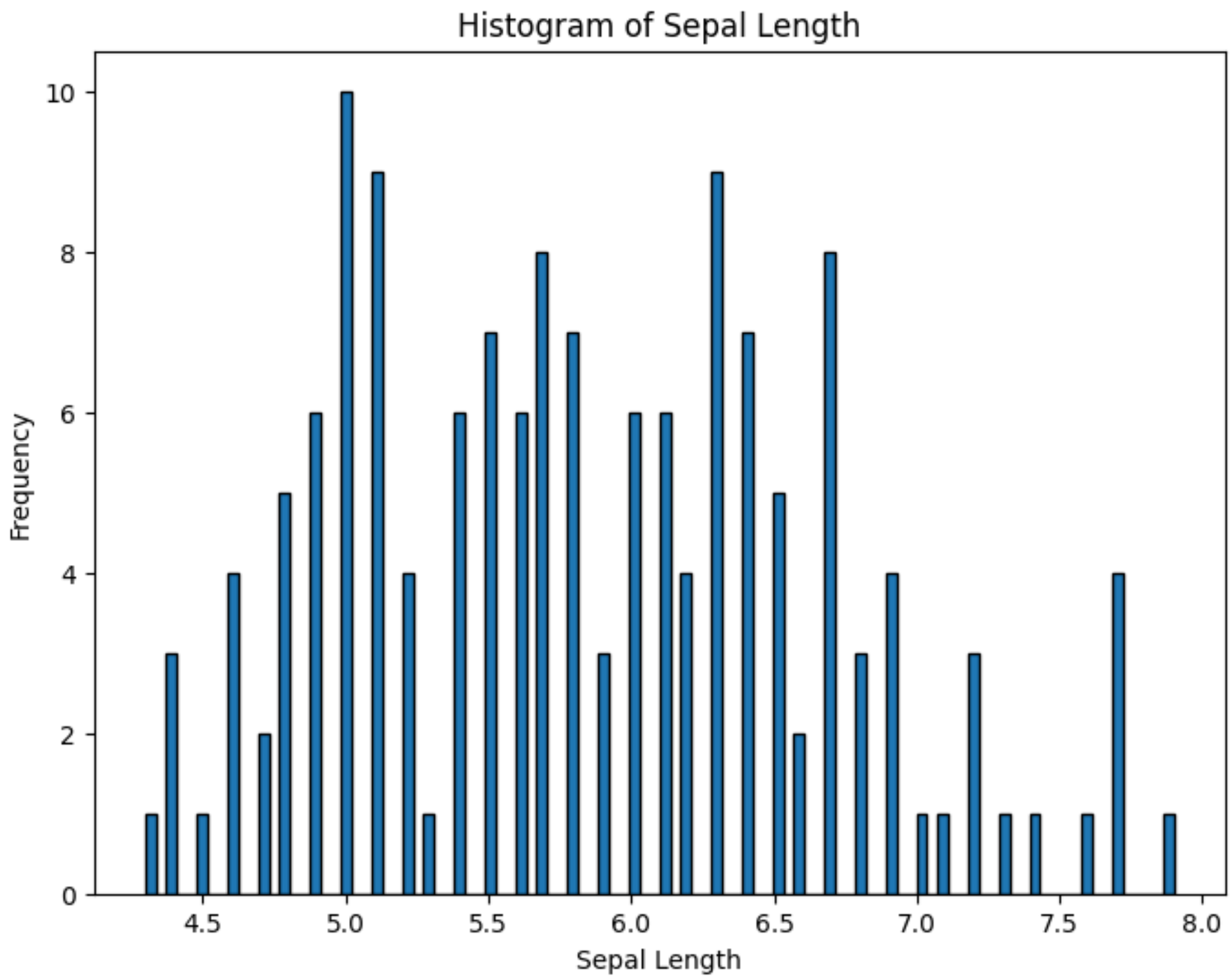
Histograms (3)

- The shape of the histogram depends on the number of garbage cans.
- In pyplot you can define this number with the parameter `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()
```

Histograms (4)



Density

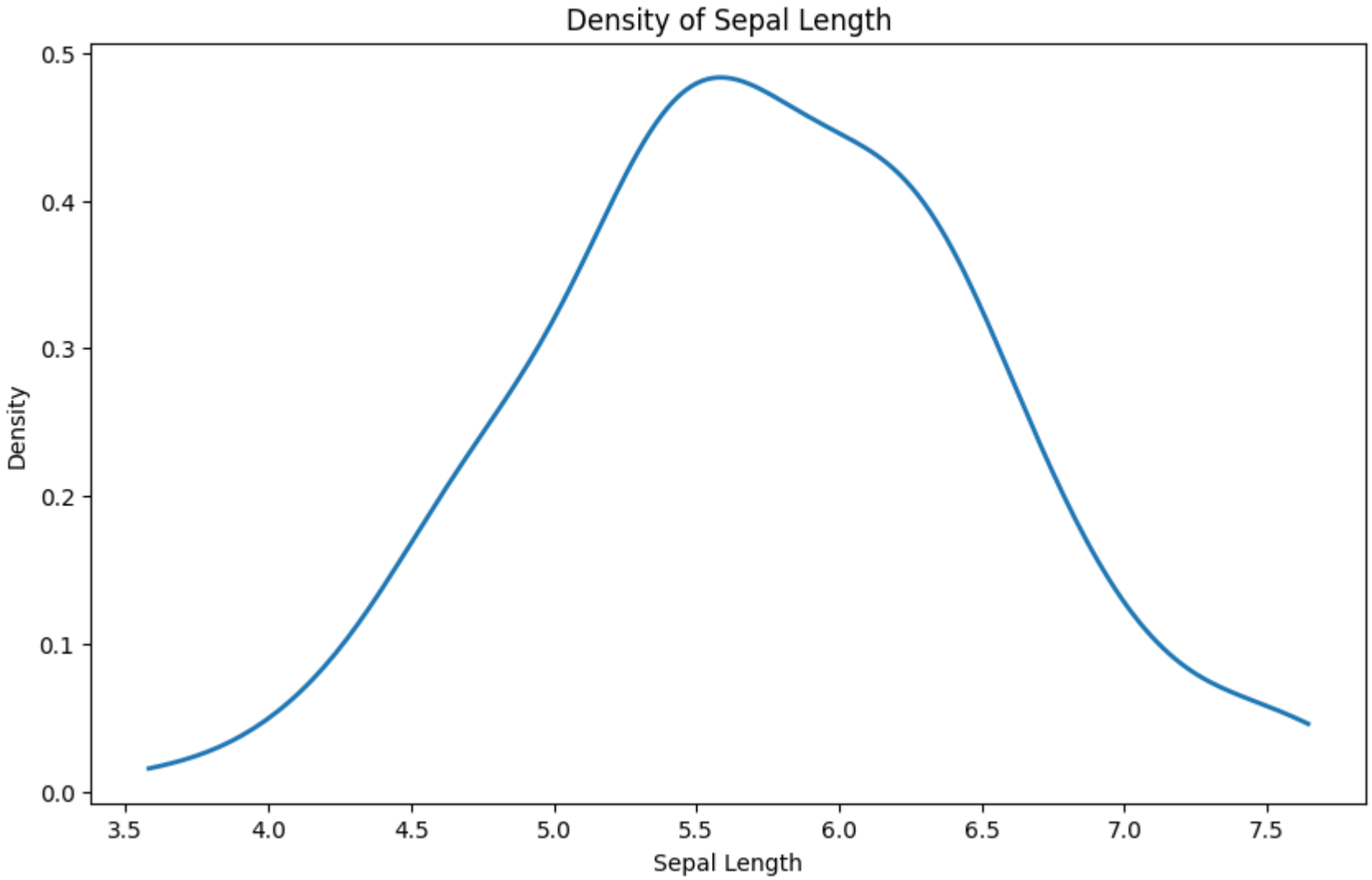
- Another way to visualize how the data are distributed is by estimating a density.

- They are calculated using non-parametric statistical techniques called **kernel density estimation**.

- The density is a smoothed version of the histogram and allows us to determine more clearly whether the observed data behave like a known density e.g. normal.

Density (1)

- In python are created with scipy's **`gaussian_kde`** command and then displayed with the plot command.



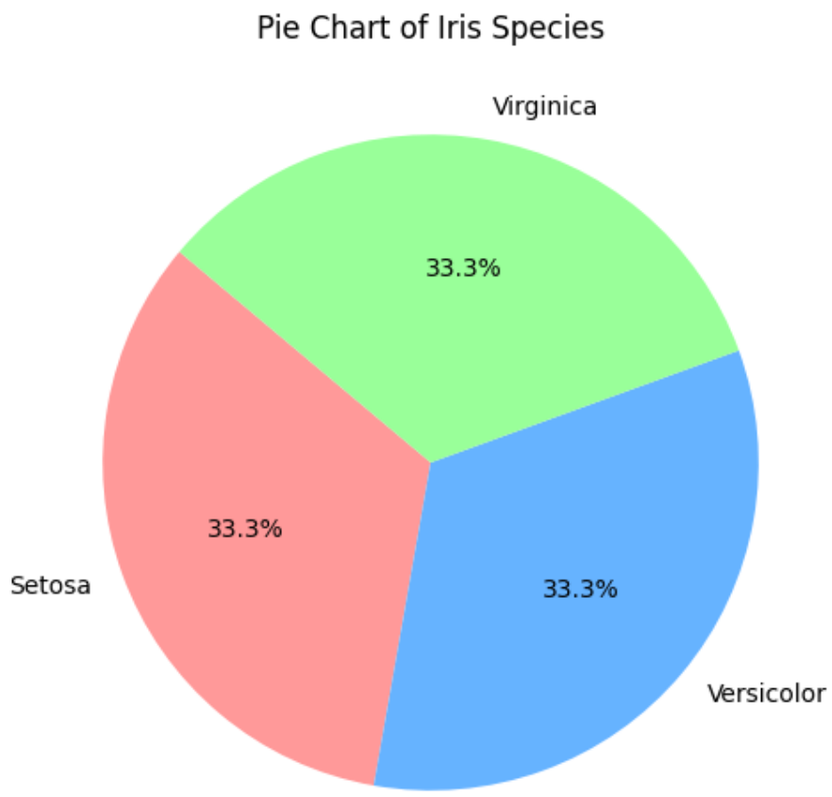
Density (2)

Code:

```
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')
```

Pie Charts

- Pie charts represent the frequency of elements in a circle.
- Each element has a share proportional to its relative frequency.
- They are generally used for categorical variables.
- Their use is not highly recommended, as they can provide misleading information.



Pie Charts (2)

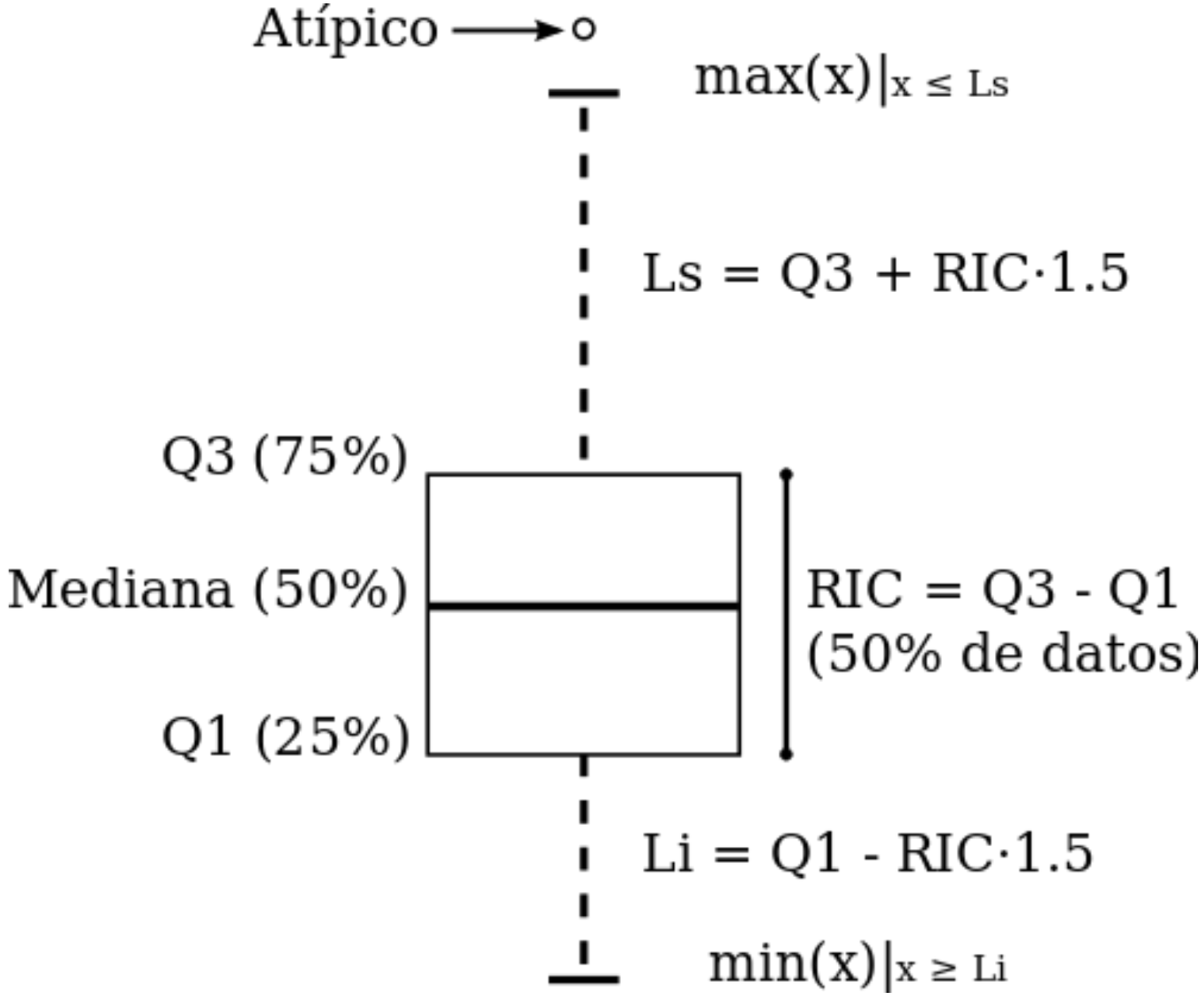
Code:

```
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

- Boxplots are constructed from the percentiles.
- A rectangle is constructed using the first and third quartiles (Q1 and Q3).
- The height of the rectangle is the interquartile range RIC (Q3 - Q1).
- The median is a line that divides the rectangle.
- Each end of the rectangle is extended with a line or arms of length Q1 - 1.5$*$RIC for the lower line and Q3 + 1.5$*$RIC for the upper line.
- Values more extreme than the length of the arms are considered outliers.
- The boxplot gives us information about the symmetry of the data distribution.
- If the median is not in the center of the rectangle, the distribution is not symmetric.
- They are useful to see the presence of outliers.

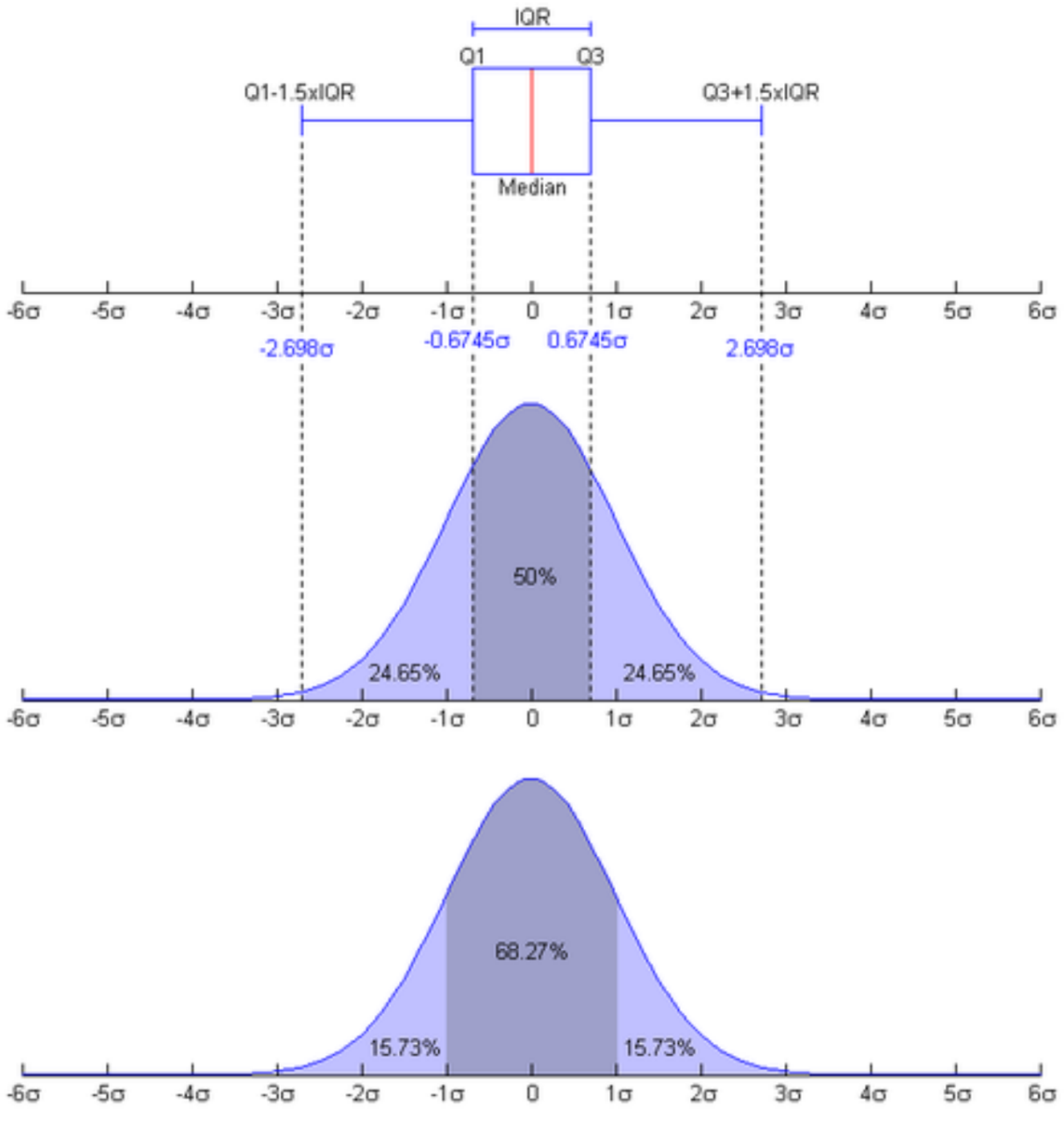
Boxplots (2)



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

Boxplots (3)

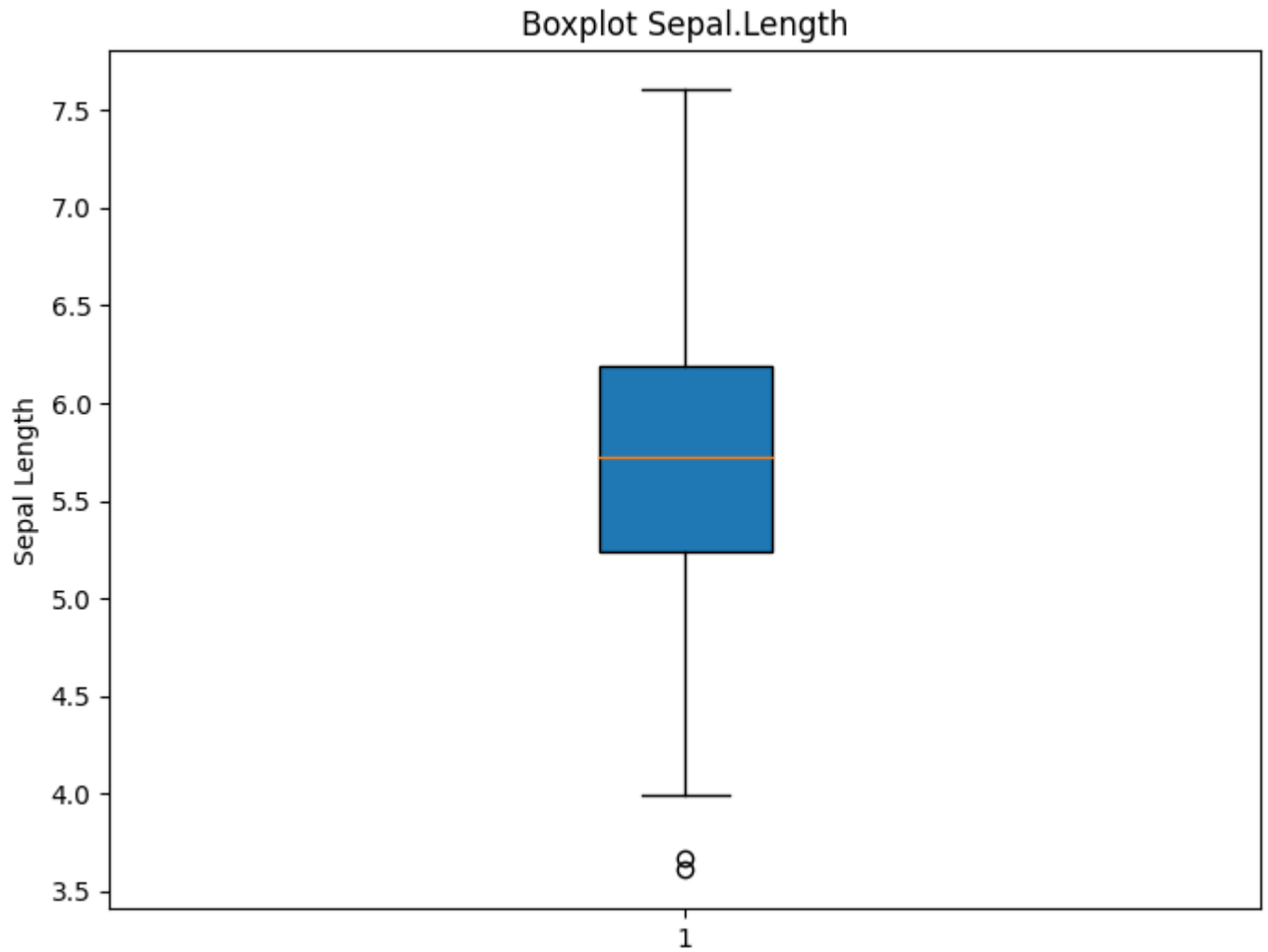
The length of the arms as well as the criteria for identifying outliers is based on the behavior of a normal.



Boxplots (4)

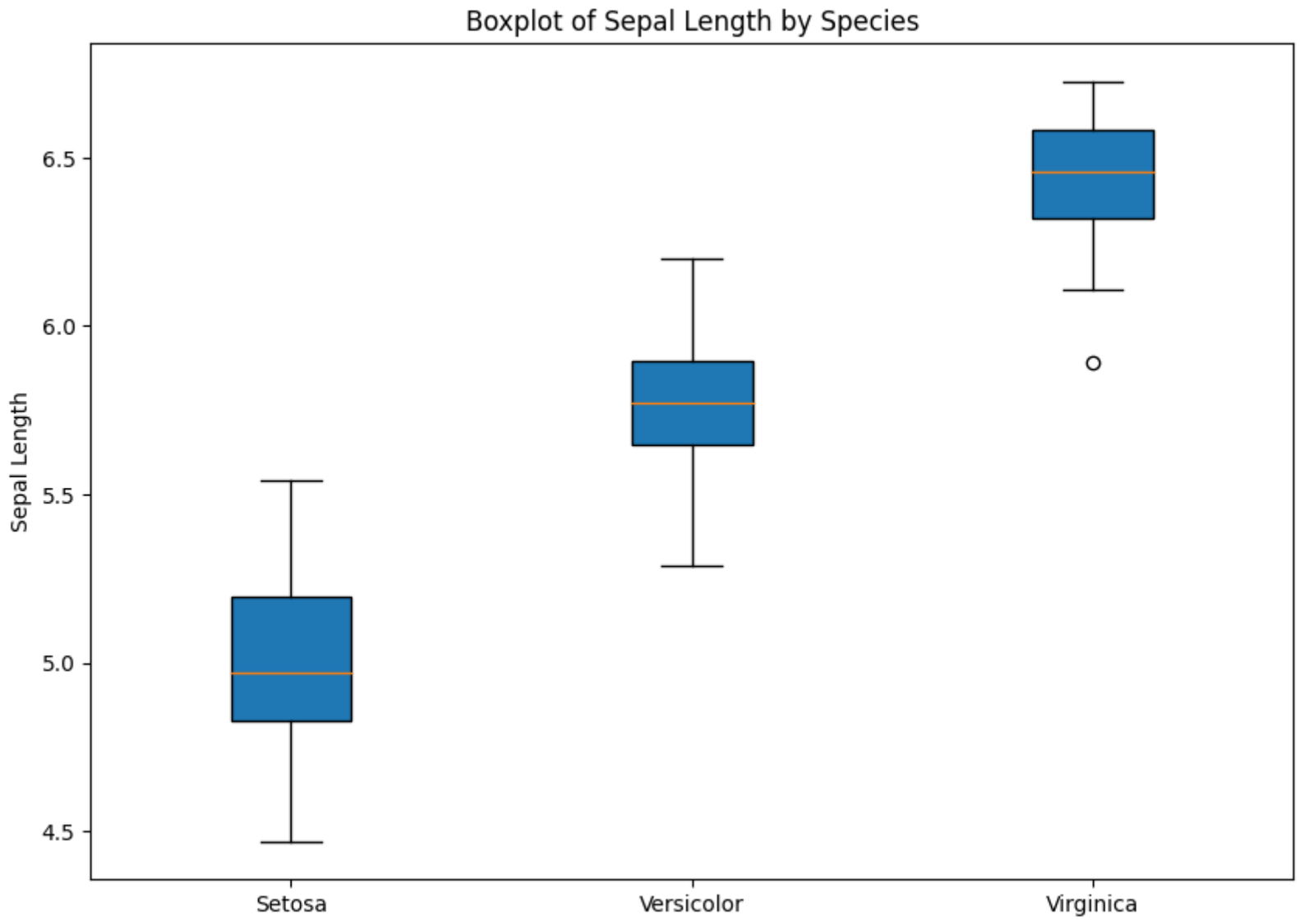
- In matplotlib boxplots are plotted with the `boxplot` function:

```
# 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)

- If we have a categorical variable, it is useful to create a boxplot for each value of this variable with respect to another numerical variable:



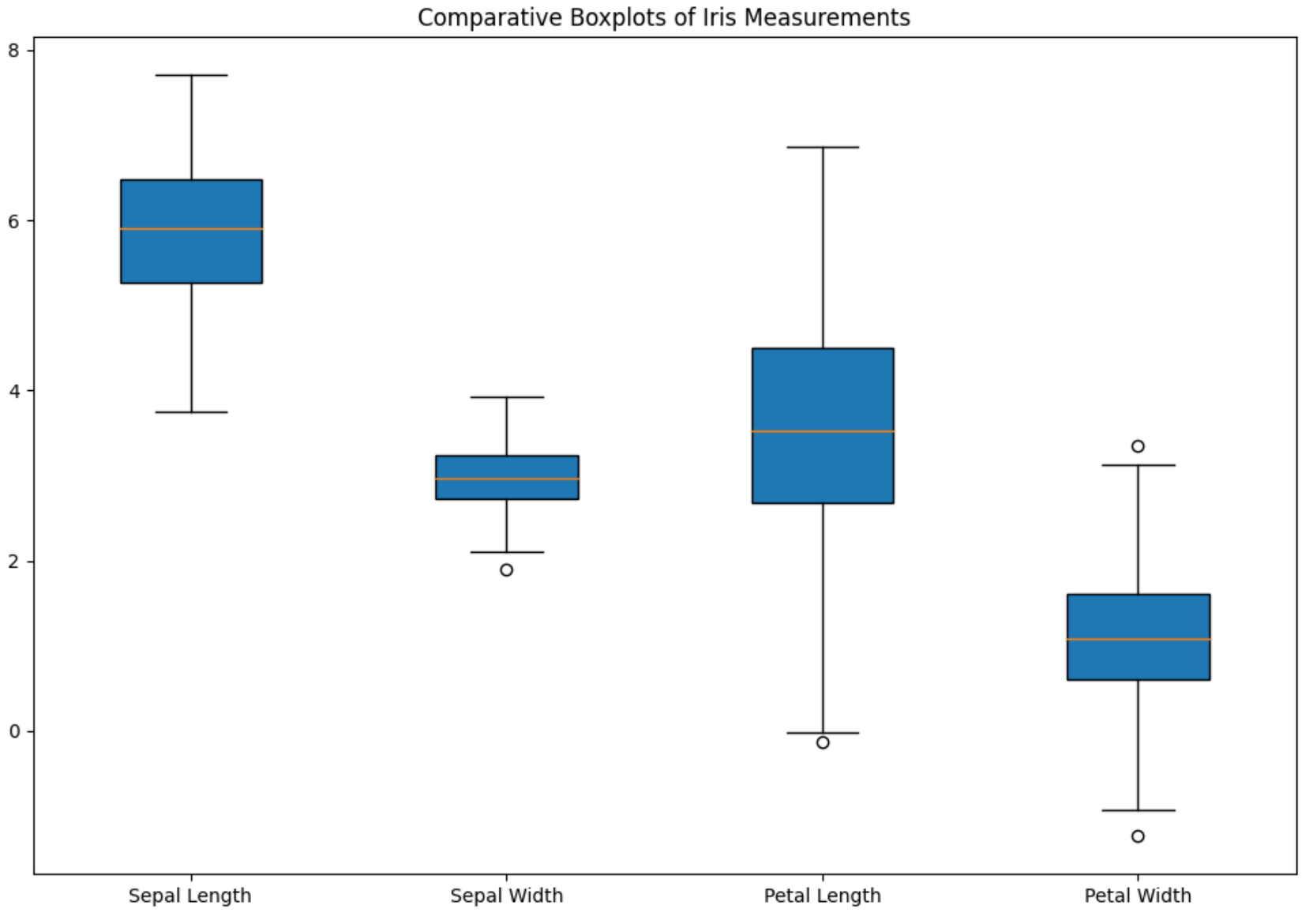
Boxplots (6)

Code:

```
# 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)

- We can also compare several boxplots in the same graph:



Boxplots (8)

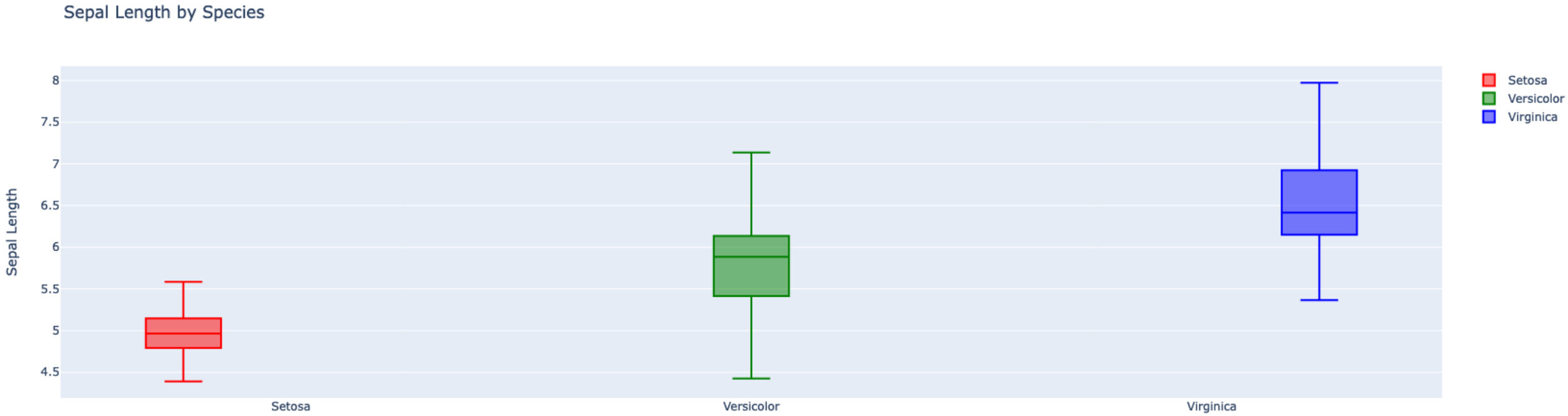
Code:

```
# 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)

- Now using `plotly`:



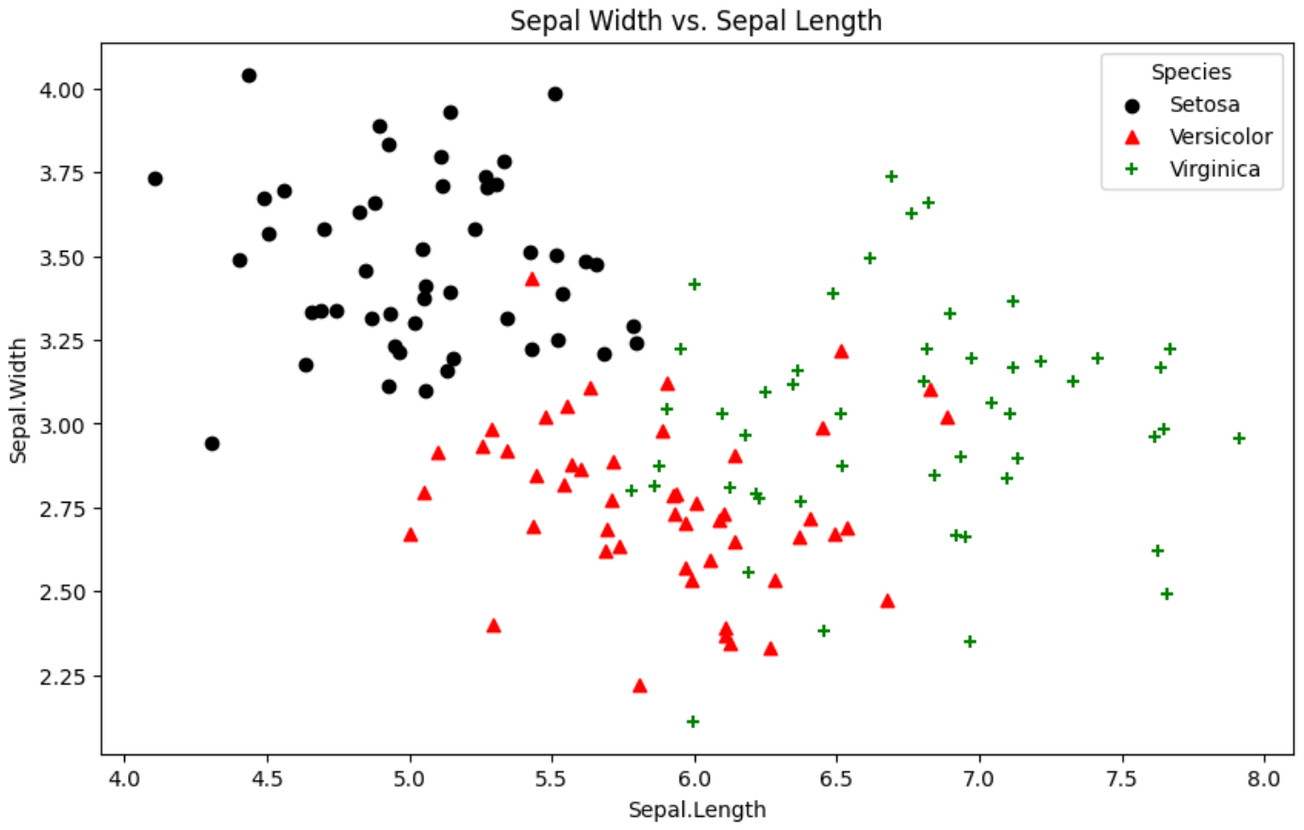
Boxplots (10)

Code:

```
# 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
)
```

Scatter Plots

- Scatter plots use Cartesian coordinates to show the values of two numerical variables of the same length.
- The values of the attributes determine the position of the elements.
- Other attributes can be used to define the size, shape or color of objects.
- In `matplotlib` we can plot a scatterplot of two numeric variables using the command scatter(x,y), which would be y vs x.



Scatter Plots (2)

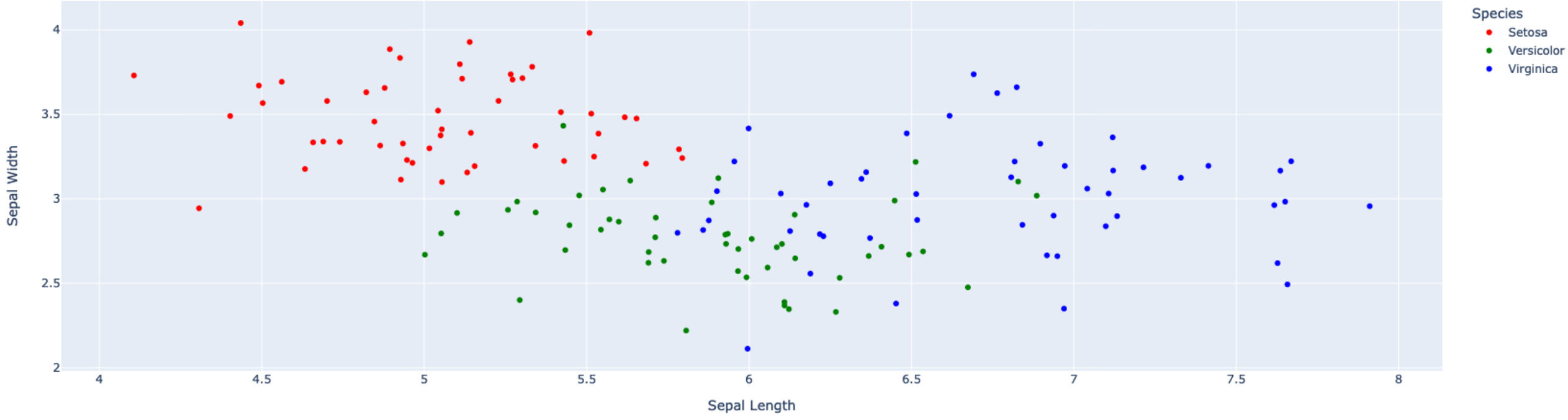
Code:

```
# 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')
```

Scatter Plots (3)

Now using plotly:

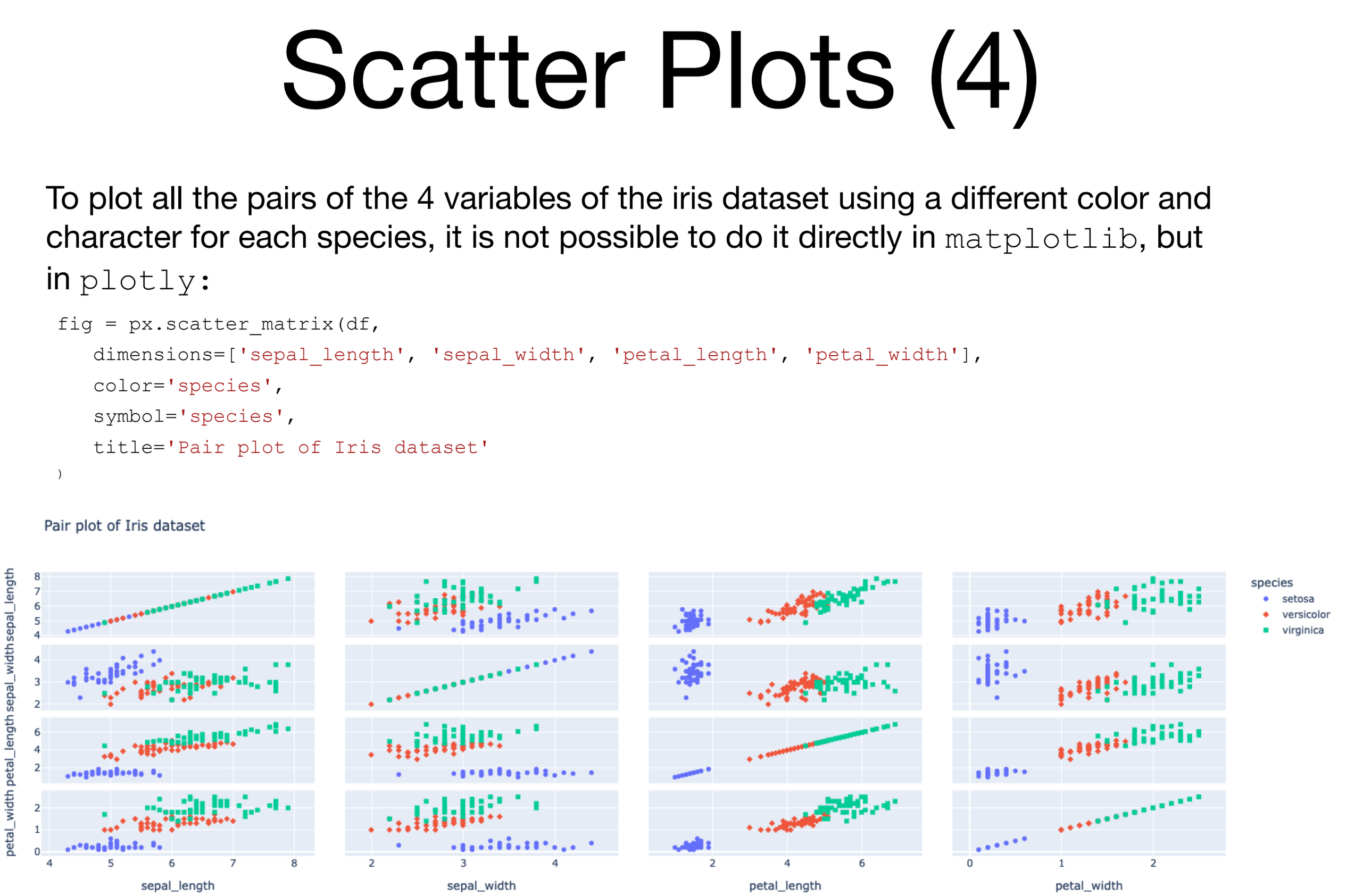
```
fig = px.scatter(iris_df, x='sepal length (cm)', y='sepal width (cm)',
color='Species', color_discrete_map={'Setosa': 'red', 'Versicolor': 'green',
'Virginica': 'blue'})
```



Scatter Plots (4)

To plot all the pairs of the 4 variables of the iris dataset using a different color and character for each species, it is not possible to do it directly in `matplotlib`, but in `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'
)
```

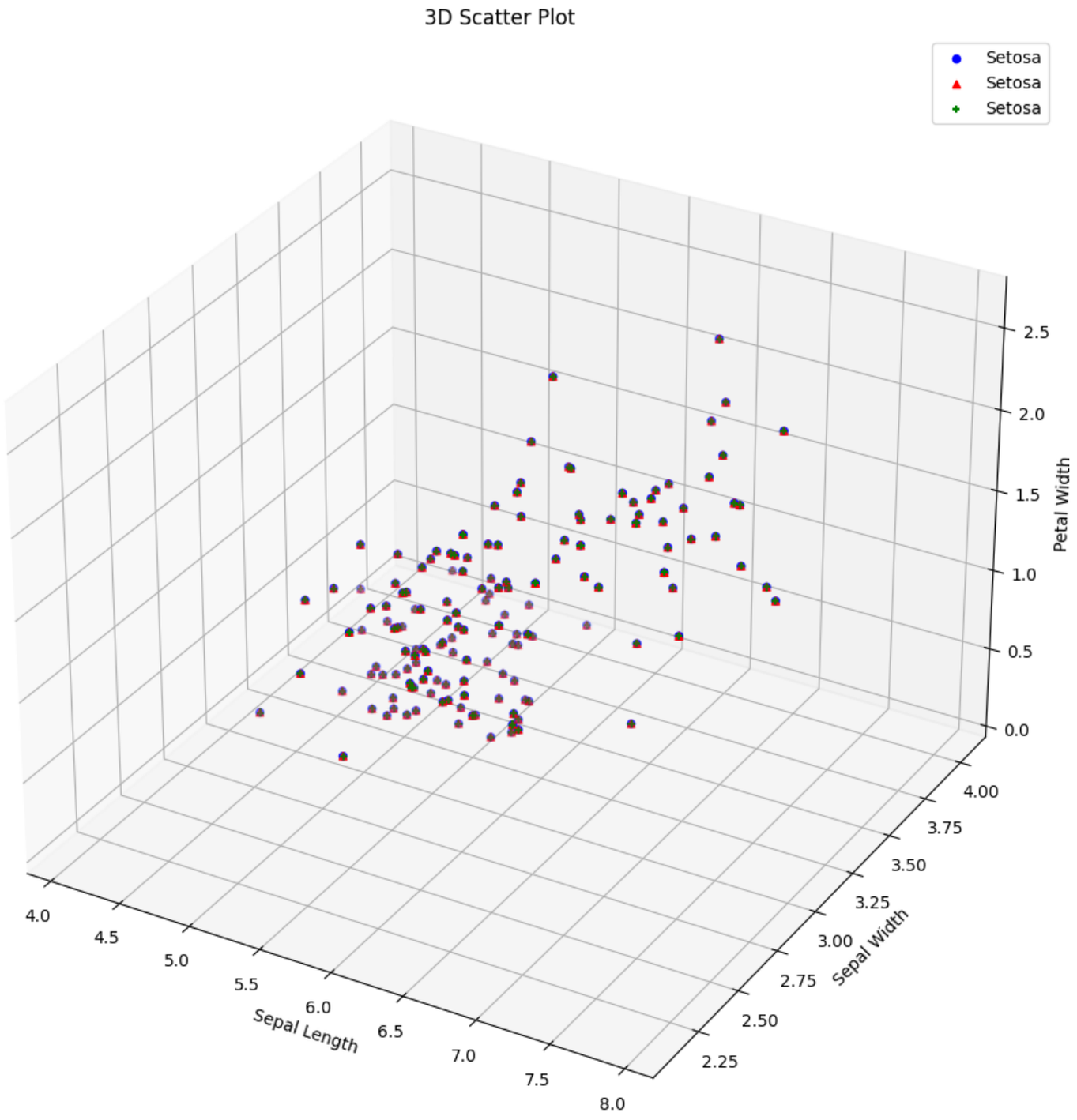


Scatter Plots (5)

- You can also create scatterplots in three dimensions using matplotlib.
- For this, import the submodule `mpl_toolkits.mplot3d`
- A 3d scatterplot for petal width, sepal length and sepal width:

```
# 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')
```

Scatter Plots (6)



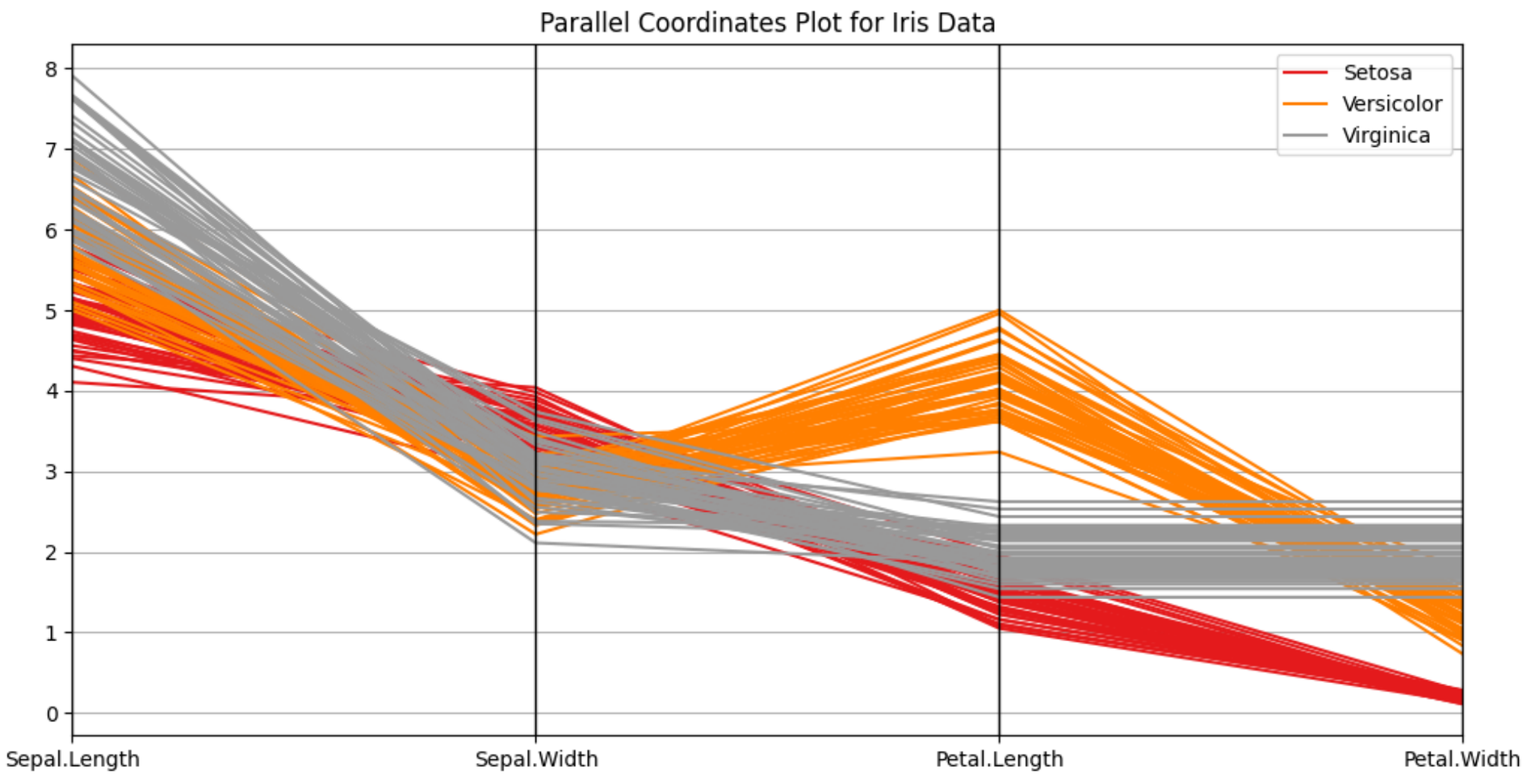
Parallel Coordinates Plots

- Parallel coordinate plots offer an alternative for visualizing multidimensional data.
- Instead of using perpendicular axes (x-y-z) we use several axes parallel to each other.
- Each attribute is represented by one of the parallel axes with its respective values.
- The values of the different attributes are scaled so that each axis has the same height.
- Each observation represents a line connecting the different axes according to their values.
- In this way, objects similar to each other tend to be grouped on lines with similar trajectory.
- In many cases it is necessary to rearrange the axes in order to visualize a pattern.

Parallel Coordinates Plots (2)

- In Python we can create parallel coordinate plots with the `parallel_coordinates` command of the `pandas` library and using `matplotlib` to visualize with `pyplot`.

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

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



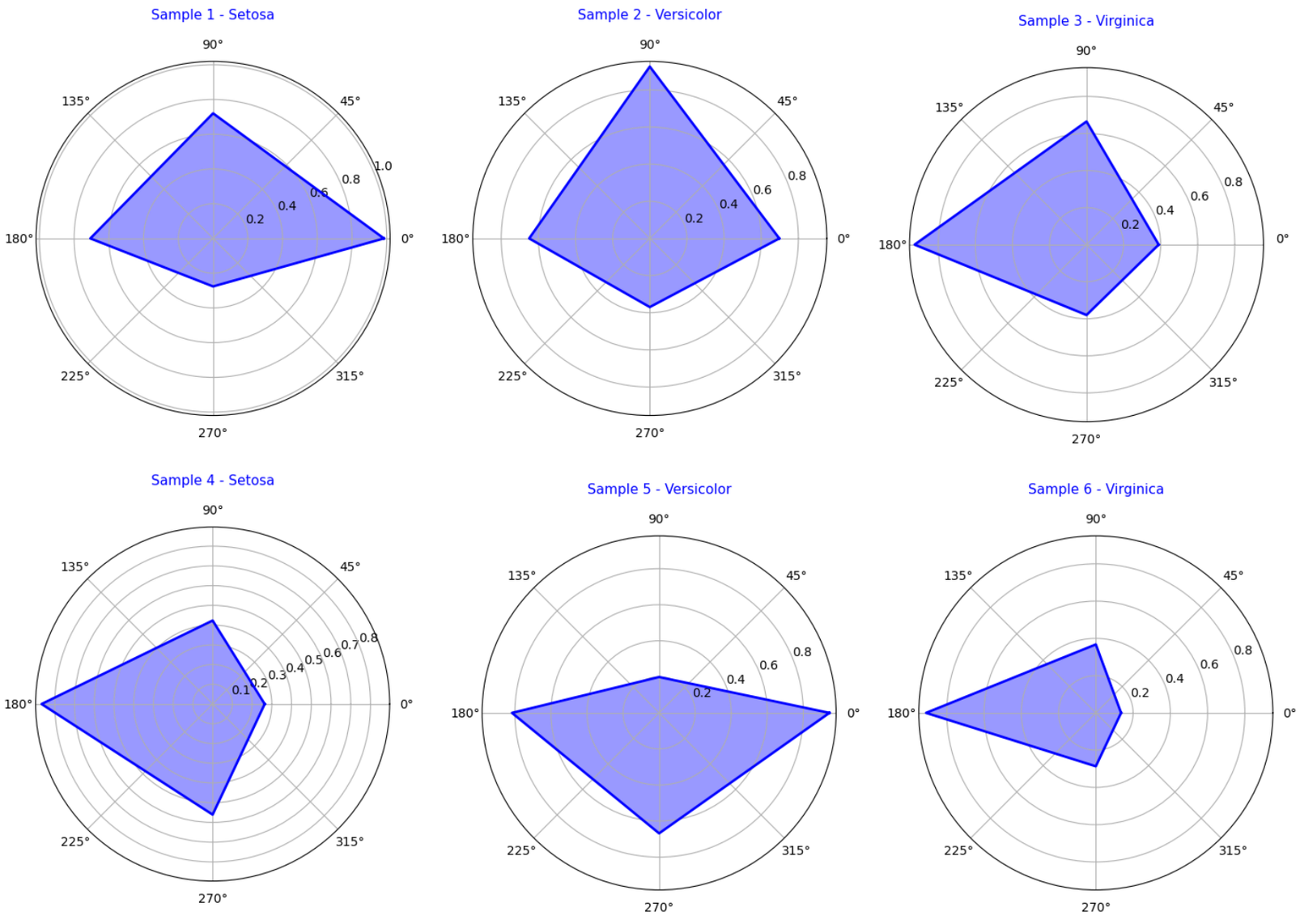
Radar Charts

- Star charts or radial charts represent each example as a star.
- Each variable is represented as an axis starting from the center of the star and extending outward in different directions, like clockwise.
- The value of each variable is represented by a line or segment connecting the center of the star to the corresponding point on the axis.
- The size of each line or segment relative to the center of the star reflects the rescaled value of the variable.
- If a line is longer, it indicates a higher value, while a shorter line indicates a lower value.
- By joining all the points corresponding to each variable, a polygon is formed that represents the profile or characteristics of the object or example being represented.
- Used to compare objects or detect outliers.

Radar Charts

- These graphs are useful for comparing objects or detecting outliers because they allow you to quickly visualize how values are distributed across multiple variables.

Example:



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')
```

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

www.dcc.uchile.cl

/ DCCUCHILE