Visualizing Data in Python

Adela Solorio A01637205

When working with a new dataset, one of the most useful things to do is to begin to visualize the data. By using **tables**, **histograms**, **boxplots**, **scatter plots** and other visual tools, we can get a better idea of what the data may be trying to tell us, and we can gain insights into the data that we may have not discovered otherwise.

In this notebook will use the Seaborn data processing library, which is a higher-level interface to **Matplotlib** that can be used to simplify many visualization tasks

The **Seaborn** provides visualisations tools that will allow to explore data from a graphical perspective.

Acknowledgments

- Data from https://www.coursera.org/ from the course "Understanding and Visualizing Data with Python" by University of Michigan

Importing libraries

```
# Import the packages that we will be using
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Importing data

```
# Define where you are running the code: colab or local
RunInColab          = True     # (False: no  | True: yes)

# If running in colab:
if RunInColab:
    # Mount your google drive in google colab
    from google.colab import drive
    drive.mount('/content/drive')

    # Find location
    #!pwd
    #!ls
    #!ls "/content/drive/My Drive/Colab Notebooks/MachineLearningWithPython/"

    # Define path del proyecto
    Ruta            = "/content/drive/MyDrive/Colab Notebooks/a01637205/NotebooksProfessor/"

else:
    # Define path del proyecto
    Ruta            = ""
```

```
 Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
```

```
# url string that hosts our .csv file
url = Ruta + "datasets/iris/iris.csv"

# Read the .csv file and store it as a pandas Data Frame
df = pd.read_csv(url, header = None, names = ["sepal_length", "sepal_width", "petal_length", "petal_width", "class"])
```

Exploring the content of the data set

Get a general 'feel' of the data

```
df.shape
```

```
(150, 5)
```

```
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Next steps: View recommended plots

```
df.tail()
```

	sepal_length	sepal_width	petal_length	petal_width	class
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

```
df.columns
```

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class'], dtype='object')
```

```
df.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
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

Frequency tables

The `value_counts()` method can be used to determine the number of times that each distinct value of a variable occurs in a data set. In statistical terms, this is the "frequency distribution" of the variable. The `value_counts()` method produces a table with two columns. The first column contains all distinct observed values for the variable. The second column contains the number of times each of these values occurs. Note that the table returned by `value_counts()` is actually a **Pandas** data frame, so can be further processed using any Pandas methods for working with data frames.

```
# Number of times that each distinct value of a variable occurs in a data set
```

```
df.sepal_length.value_counts()
```

```
5.0    10
5.1     9
6.3     9
5.7     8
6.7     8
5.8     7
5.5     7
```

```
6.4     7
4.9     6
5.4     6
6.1     6
6.0     6
5.6     6
4.8     5
6.5     5
6.2     4
7.7     4
6.9     4
4.6     4
5.2     4
5.9     3
4.4     3
7.2     3
6.8     3
6.6     2
4.7     2
7.6     1
7.4     1
7.3     1
7.0     1
7.1     1
5.3     1
4.3     1
4.5     1
7.9     1
Name: sepal_length, dtype: int64
```

```
df.sepal_width.value_counts()
```

```
3.0    26
2.8    14
3.2    13
3.4    12
3.1    11
2.9    10
2.7     9
2.5     8
3.5     6
3.3     6
3.8     6
2.6     5
2.3     4
3.6     4
3.7     3
2.4     3
2.2     3
3.9     2
4.4     1
4.0     1
4.1     1
4.2     1
2.0     1
Name: sepal_width, dtype: int64
```

```
df.petal_length.value_counts()
```

```
1.4    13
1.5    13
5.1     8
4.5     8
1.6     7
1.3     7
5.6     6
4.7     5
4.9     5
4.0     5
4.2     4
5.0     4
4.4     4
4.8     4
1.7     4
3.9     3
4.6     3
5.7     3
4.1     3
5.5     3
6.1     3
5.8     3
3.3     2
```

```
5.4     2
6.7     2
5.3     2
5.9     2
6.0     2
1.2     2
4.3     2
1.9     2
3.5     2
5.2     2
3.0     1
1.1     1
3.7     1
3.8     1
6.6     1
6.3     1
1.0     1
6.9     1
3.6     1
6.4     1
Name: petal_length, dtype: int64
```

```
df.petal_width.value_counts()
```

```
0.2    29
1.3    13
1.8    12
1.5    12
1.4     8
2.3     8
1.0     7
0.4     7
0.3     7
2.1     6
2.0     6
0.1     5
1.2     5
1.9     5
1.6     4
2.5     3
2.2     3
2.4     3
1.1     3
1.7     2
0.6     1
0.5     1
Name: petal_width, dtype: int64
```

```
# Proportion of each distinct value of a variable occurs in a data set
```

```
sLength = df.sepal_length.value_counts()
porcentaje = (100*sLength)/sLength.sum()
porcentaje
```

```
5.0    6.666667
5.1    6.000000
6.3    6.000000
5.7    5.333333
6.7    5.333333
5.8    4.666667
5.5    4.666667
6.4    4.666667
4.9    4.000000
5.4    4.000000
6.1    4.000000
6.0    4.000000
5.6    4.000000
4.8    3.333333
6.5    3.333333
6.2    2.666667
7.7    2.666667
6.9    2.666667
4.6    2.666667
5.2    2.666667
5.9    2.000000
4.4    2.000000
7.2    2.000000
6.8    2.000000
6.6    1.333333
4.7    1.333333
7.6    0.666667
```

```
7.4    0.666667
7.3    0.666667
7.0    0.666667
7.1    0.666667
5.3    0.666667
4.3    0.666667
4.5    0.666667
7.9    0.666667
Name: sepal_length, dtype: float64
```

```
sWidth = df.sepal_width.value_counts()
porcentaje = (100*sWidth)/sWidth.sum()
porcentaje
```

```
3.0    17.333333
2.8     9.333333
3.2     8.666667
3.4     8.000000
3.1     7.333333
2.9     6.666667
2.7     6.000000
2.5     5.333333
3.5     4.000000
3.3     4.000000
3.8     4.000000
2.6     3.333333
2.3     2.666667
3.6     2.666667
3.7     2.000000
2.4     2.000000
2.2     2.000000
3.9     1.333333
4.4     0.666667
4.0     0.666667
4.1     0.666667
4.2     0.666667
2.0     0.666667
Name: sepal_width, dtype: float64
```

```
pLength = df.petal_length.value_counts()
porcentaje = (100*pLength)/pLength.sum()
porcentaje
```

```
1.4    8.666667
1.5    8.666667
5.1    5.333333
4.5    5.333333
1.6    4.666667
1.3    4.666667
5.6    4.000000
4.7    3.333333
4.9    3.333333
4.0    3.333333
4.2    2.666667
5.0    2.666667
4.4    2.666667
4.8    2.666667
1.7    2.666667
3.9    2.000000
4.6    2.000000
5.7    2.000000
4.1    2.000000
5.5    2.000000
6.1    2.000000
5.8    2.000000
3.3    1.333333
5.4    1.333333
6.7    1.333333
5.3    1.333333
5.9    1.333333
6.0    1.333333
1.2    1.333333
4.3    1.333333
1.9    1.333333
3.5    1.333333
5.2    1.333333
3.0    0.666667
1.1    0.666667
3.7    0.666667
3.8    0.666667
6.6    0.666667
6.3    0.666667
```

```
1.0    0.666667
6.9    0.666667
3.6    0.666667
6.4    0.666667
Name: petal_length, dtype: float64
```

```
pWidth = df.petal_width.value_counts()
porcentaje = (100*pWidth)/pWidth.sum()
porcentaje
```

```
0.2    19.333333
1.3     8.666667
1.8     8.000000
1.5     8.000000
1.4     5.333333
2.3     5.333333
1.0     4.666667
0.4     4.666667
0.3     4.666667
2.1     4.000000
2.0     4.000000
0.1     3.333333
1.2     3.333333
1.9     3.333333
1.6     2.666667
2.5     2.000000
2.2     2.000000
2.4     2.000000
1.1     2.000000
1.7     1.333333
0.6     0.666667
0.5     0.666667
Name: petal_width, dtype: float64
```

Note that the `value_counts()` method excludes missing values. We confirm this below by adding up observations to your data frame with some missing values and then computing `value_counts()` and comparing this to the total number of rows in the data set, which is 28. This tells us that there are 28 - (21+6) = 1 missing values for this variable (other variables may have different numbers of missing values).

```
# Total number of observations
print("Numero de observaciones: ", df.shape[0])
```

```
# Total number of null observations
print("Numero de observaciones nulas: ", df.isnull().sum())
```

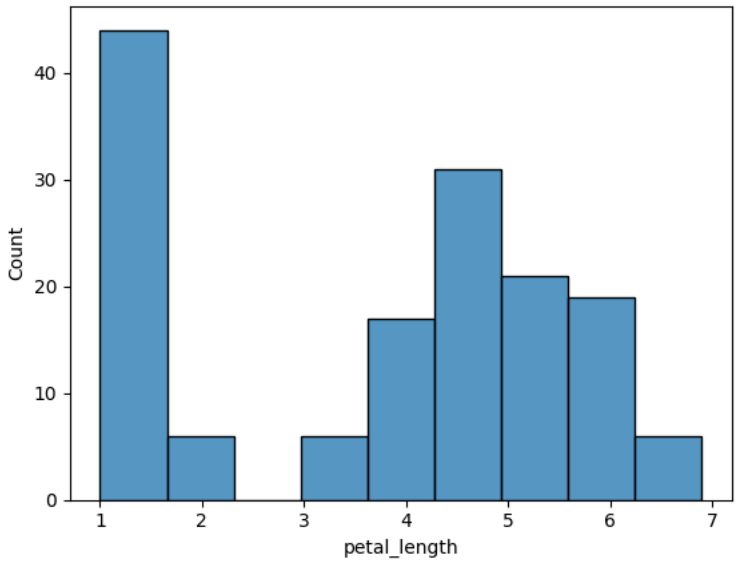
```
# Total number of counts (excluding missing values)
print("Numero de observaciones no nulas: ", df.notnull().sum())
```

```
Numero de observaciones:  150
Numero de observaciones nulas:  sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
class           0
dtype: int64
Numero de observaciones no nulas:  sepal_length    150
sepal_width     150
petal_length    150
petal_width     150
class           150
dtype: int64
```

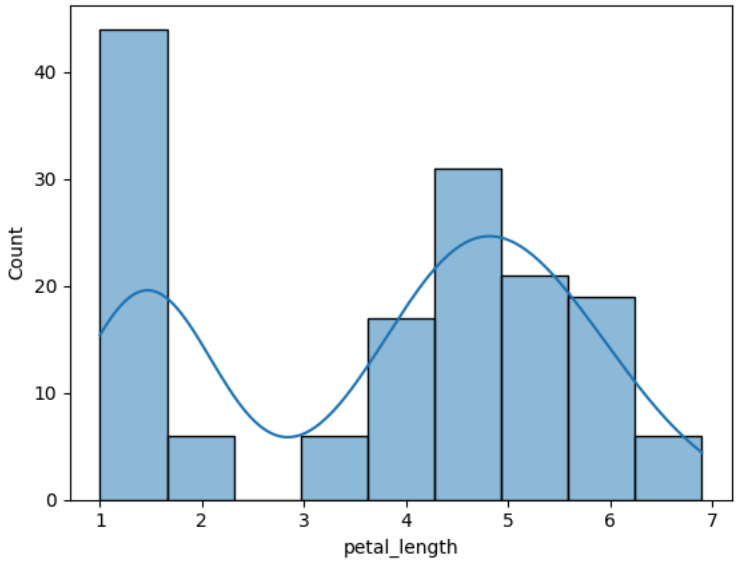
Histogram

It is often good to get a feel for the shape of the distribution of the data.

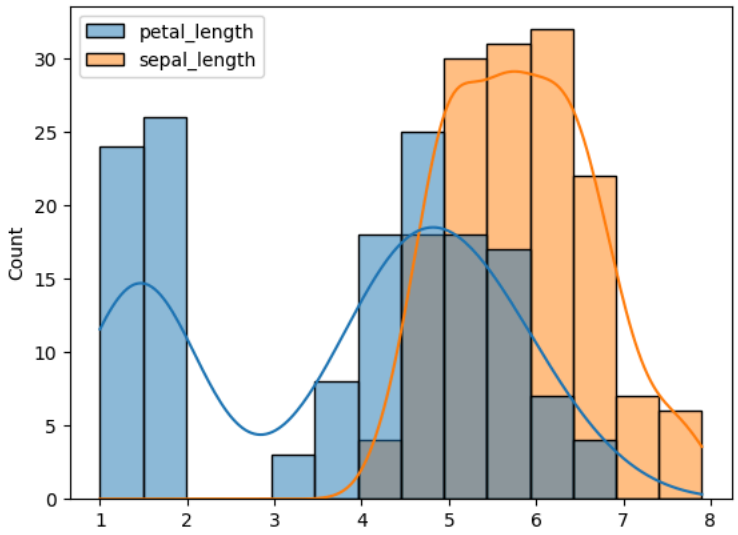
```
# Plot histogram of "petal_length"
sns.histplot(df.petal_length)
plt.show()
```




```
# Plot distribution of the tips only

sns.histplot(df.petal_length, kde=True)
plt.show()
```



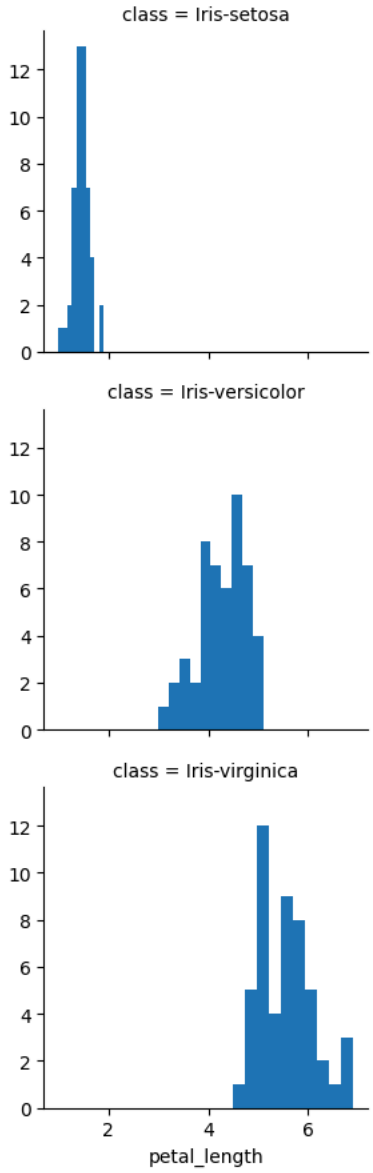

```
# Plot histogram of both the petal_length and the sepal_length
df2plot = df [["petal_length", "sepal_length"]]
sns.histplot(df2plot, kde=True)
plt.show()
```



Histograms plotted by groups

While looking at a single variable is interesting, it is often useful to see how a variable changes in response to another. Thus, we can create a histograms of one quantitative variable grouped by another categorical variables.

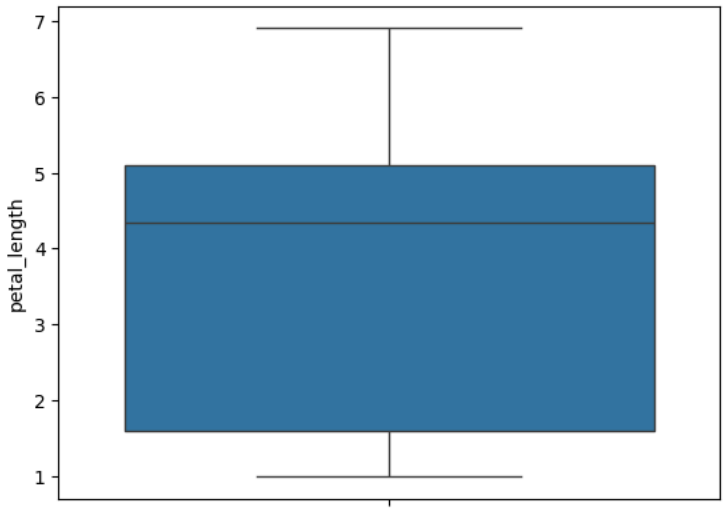
```
# Create histograms of the "petal_length" grouped by "class"
g = sns.FacetGrid(df, row="class")

g = g.map(plt.hist, "petal_length")

plt.show()
```



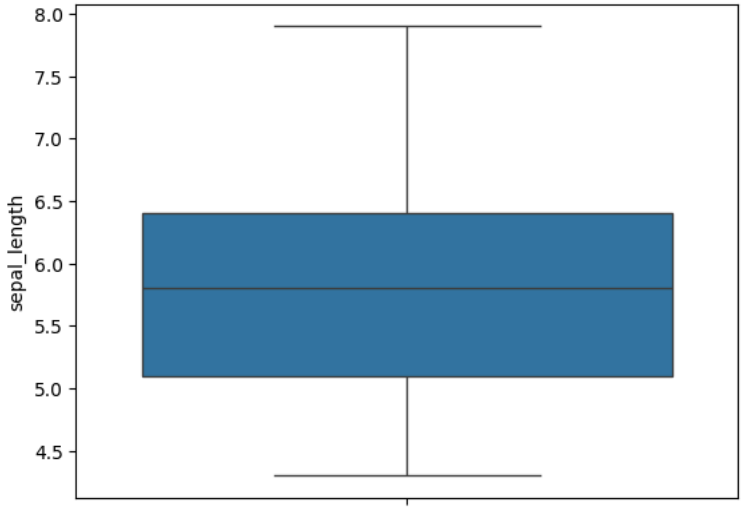
Boxplots

Boxplots do not show the shape of the distribution, but they can give us a better idea about the center and spread of the distribution as well as any potential outliers that may exist. Boxplots and Histograms often complement each other and help an analyst get more information about the data

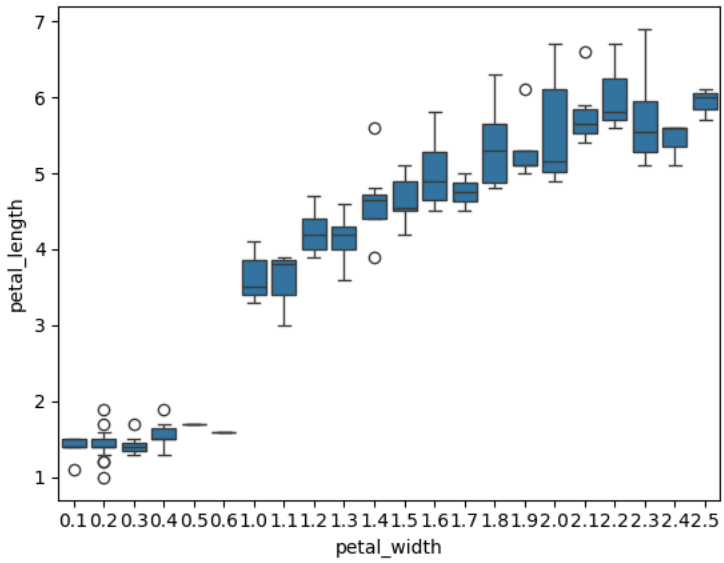
```
# Create the boxplot of the "petal_length" amounts
sns.boxplot(df["petal_length"])
plt.show()
```



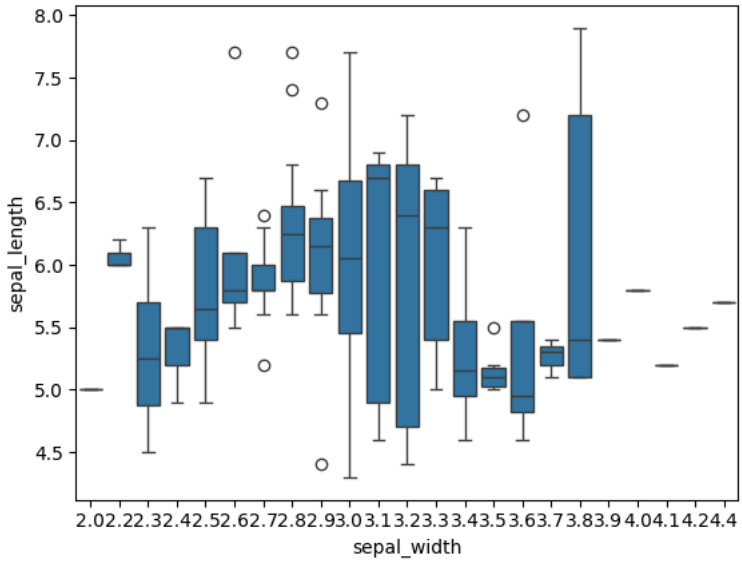

```
# Create the boxplot of the "sepal_length" amounts
sns.boxplot(df["sepal_length"])
plt.show()
```




```
# Create the boxplots of the "petal_width" and of the "petal_length" amounts
sns.boxplot(data = df, x = "petal_width", y = "petal_length")
plt.show()
```




```
# Create the boxplots of the "sepal_width" and of the "sepal_length" amounts
sns.boxplot(data = df, x = "sepal_width", y = "sepal_length")
plt.show()
```



Boxplots plotted by groups

While looking at a single variable is interesting, it is often useful to see how a variable changes in response to another. Thus, we can create a side-by-side boxplots of one quantitative variable grouped by another categorical variables.

```
# Create side-by-side boxplots of the "Height" grouped by "Gender"
```

Histograms and boxplots plotted by groups

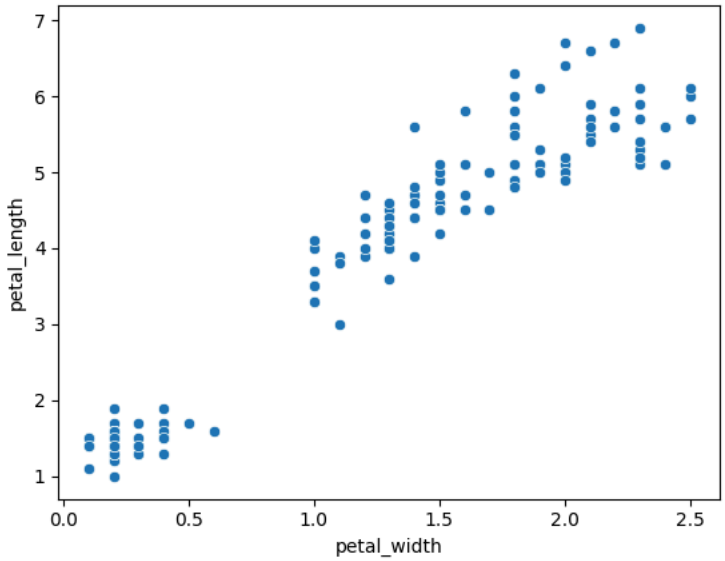
We cal also create both boxplots and histograms of one quantitative variable grouped by another categorical variables

[ ] ↳ *1 celda oculta*

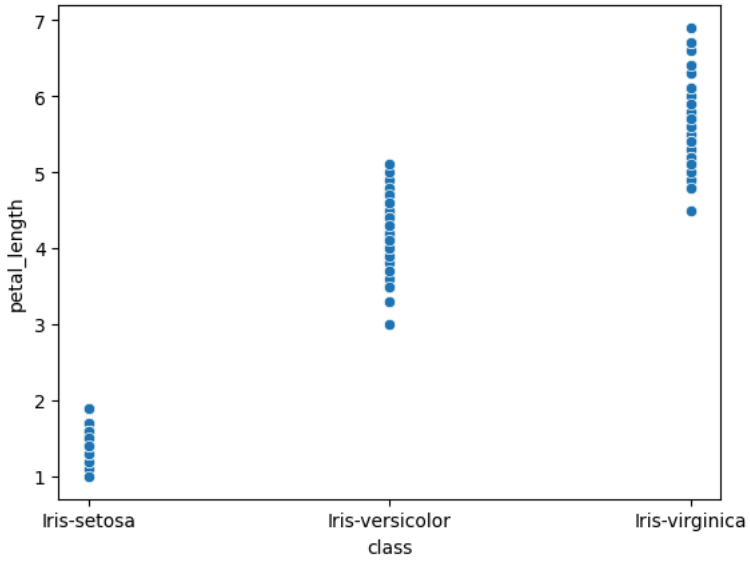
Scatter plot

Plot values of one variable versus another variable to see how they are correlated

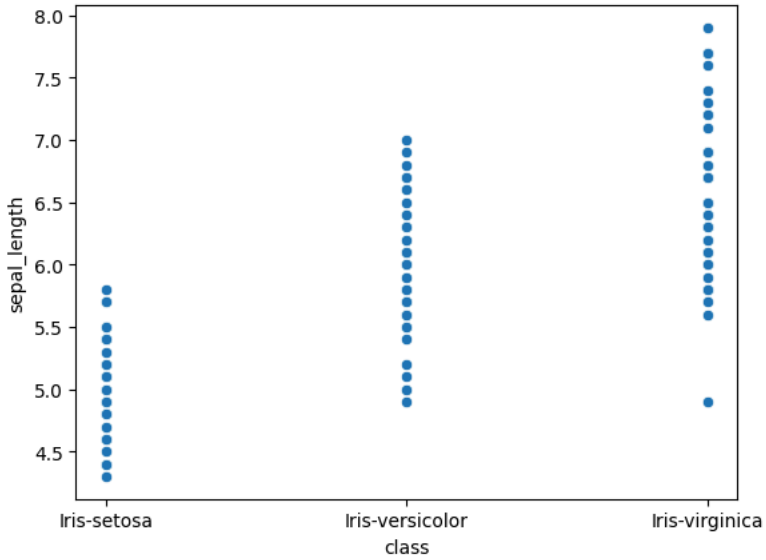
```
# scatter plot between two variables
sns.scatterplot(data=df, y="petal_length", x="petal_width")
plt.show()
```



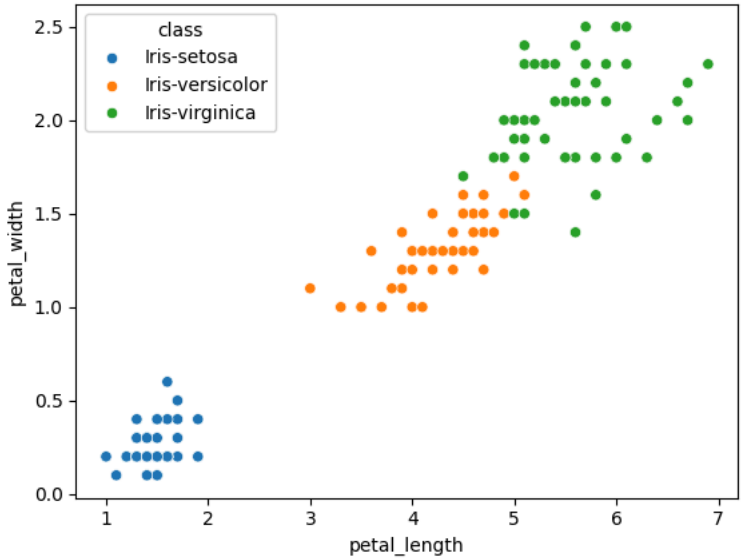

```
# scatter plot between two variables (one categorical)
sns.scatterplot(data=df, y="petal_length", x="class")
plt.show()
```



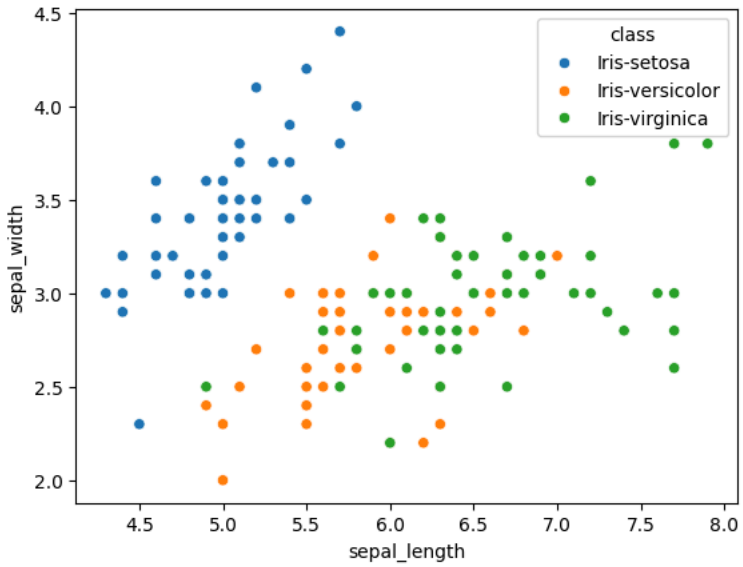

```
# scatter plot between two variables (one categorical)
sns.scatterplot(data=df, y="sepal_length", x="class")
plt.show()
```



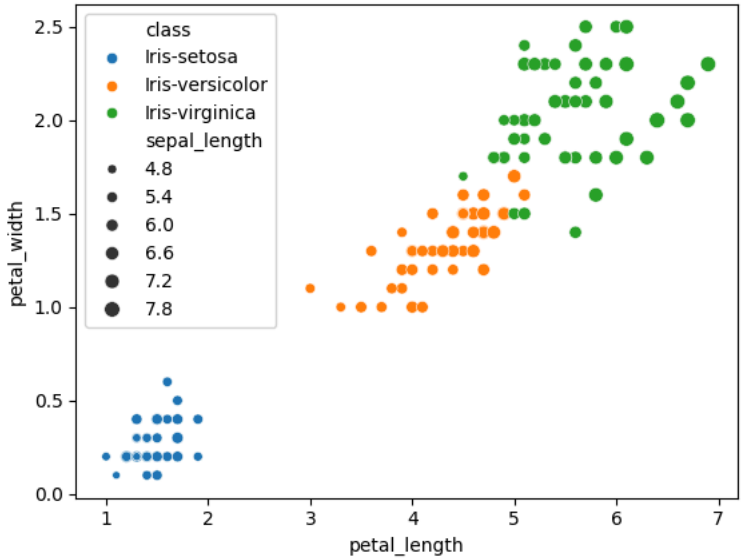

```
# scatter plot between two variables grouped according to a categorical variable
sns.scatterplot(data=df, x="petal_length", y="petal_width", hue="class")
plt.show()
```




```
# scatter plot between two variables grouped according to a categorical variable
sns.scatterplot(data=df, x="sepal_length", y="sepal_width", hue="class")
plt.show()
```




```
# scatter plot between two variables grouped according to a categorical variable and with size of markers
sns.scatterplot(data=df, x="petal_length", y="petal_width", hue="class", size="sepal_length")
plt.show()
```



Final remarks

- Visualizing your data using **tables**, **histograms**, **boxplots**, **scatter plots** and other tools is essential to carry put analysis and extract conclusions
- There are several ways to do the same thing
- The **Seaborn** package provides visualisations tools that allow to explore data from a graphical perspective

Activity: work with the iris dataset

Repeat this tutorial with the iris data set and respond to the following inquiries

1. Plot the histograms for each of the four quantitative variables
2. Plot the histograms for each of the quantitative variables
3. Plot the boxplots for each of the quantitative variables

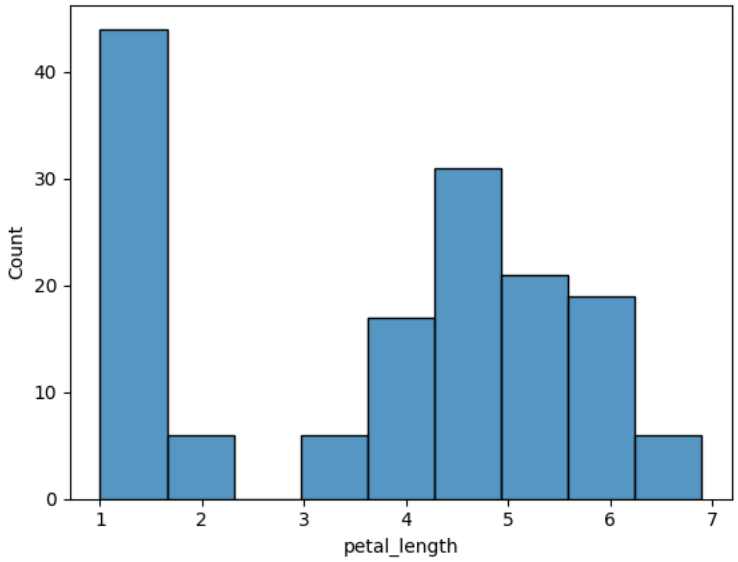
4. Plot the boxplots of the petal width grouped by type of flower

5. Plot the boxplots of the setal length grouped by type of flower

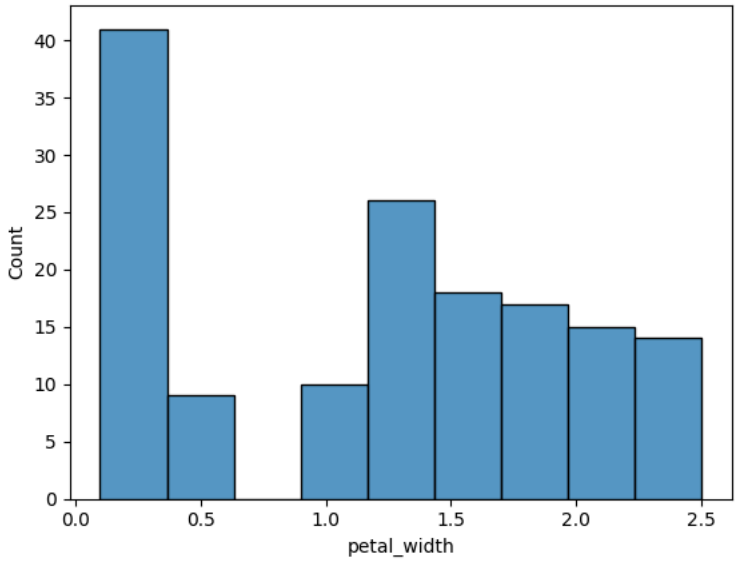
6. Provide a description (explaination from your observations) of each of the quantitative variables

Parte 1: histograma para cada variable cuantitativa

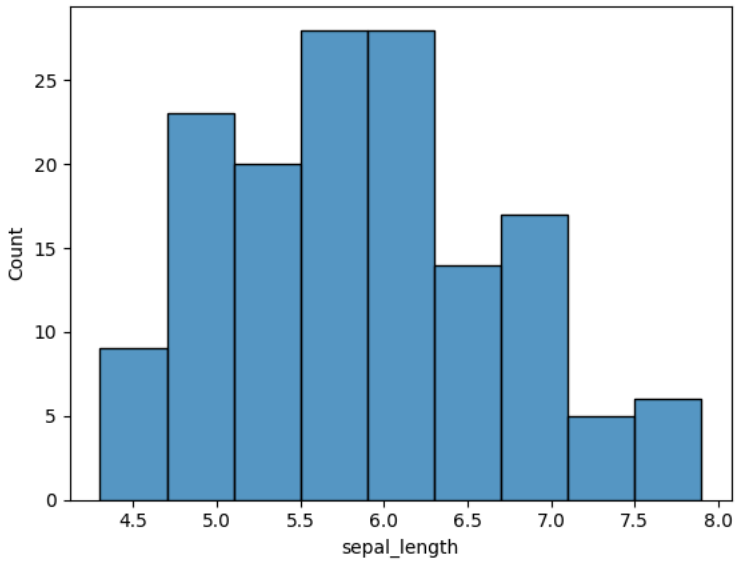
```
# Plot histogram de "petal_length"
sns.histplot(df.petal_length)
plt.show()
```



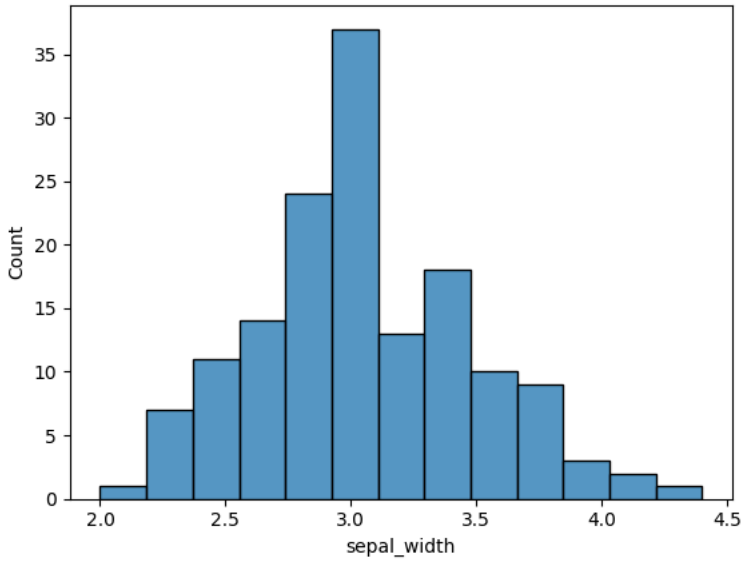

```
# Plot histogram de "petal_width"
sns.histplot(df.petal_width)
plt.show()
```




```
# Plot histogram de "sepal_length"
sns.histplot(df.sepal_length)
plt.show()
```

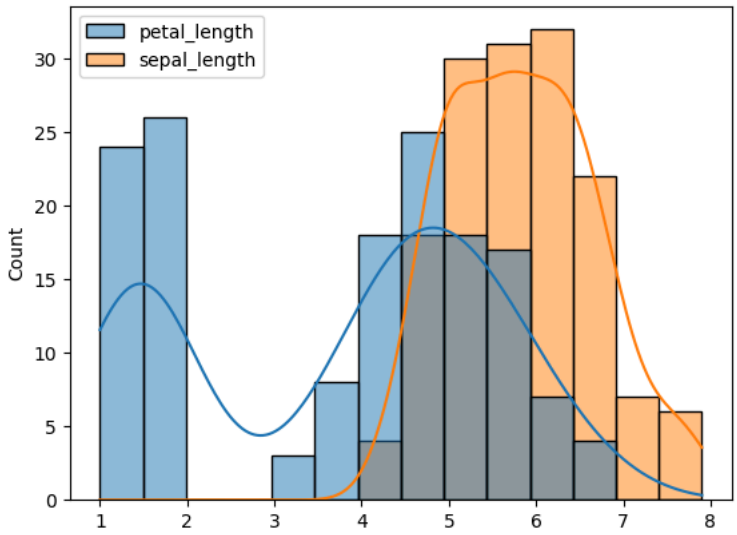



```
# Plot histogram de "sepal_width"
sns.histplot(df.sepal_width)
plt.show()
```

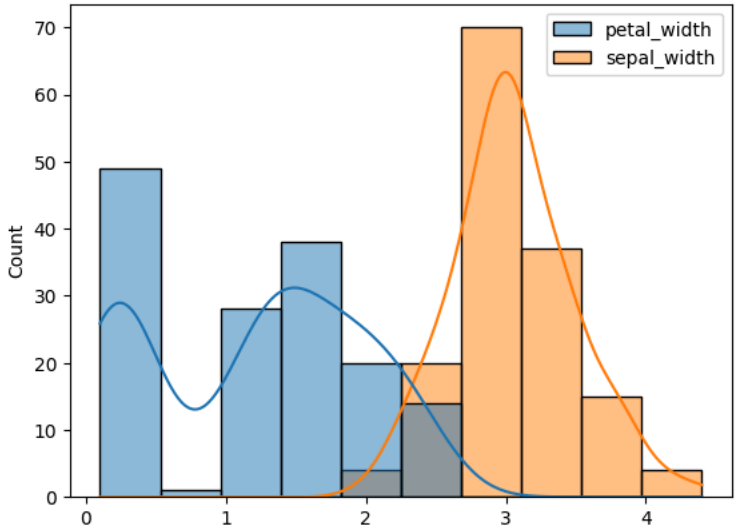


Parte 2: histograma en comparación de las variables cuantitativas

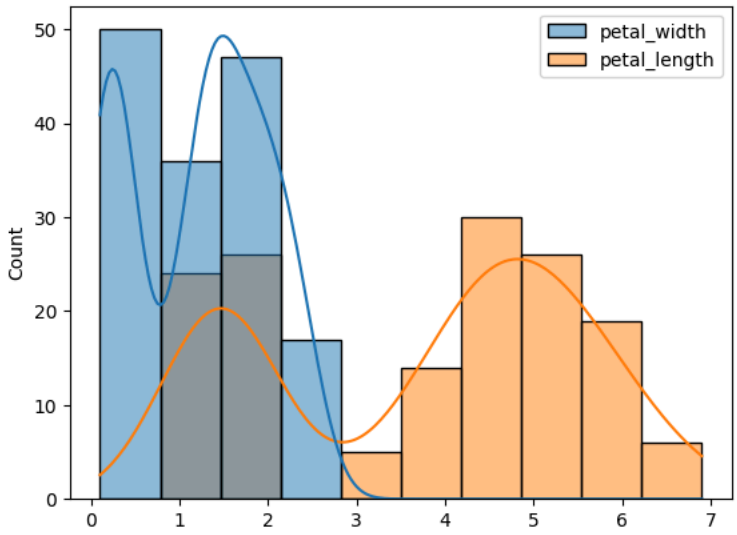
```
#petal_length y sepal_length
df2plot = df [["petal_length", "sepal_length"]]
sns.histplot(df2plot, kde=True)
plt.show()
```



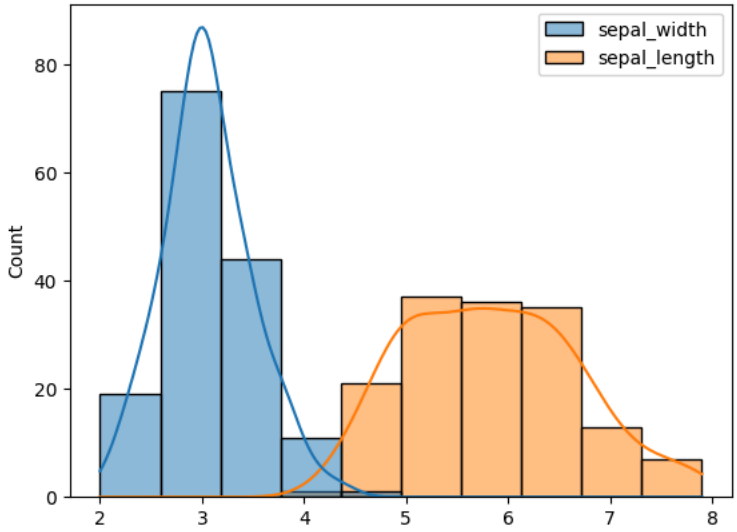

```
#petal_width y sepal_width
df2plot = df [["petal_width", "sepal_width"]]
sns.histplot(df2plot, kde=True)
plt.show()
```




```
#petal_width y petal_length
df2plot = df [["petal_width", "petal_length"]]
sns.histplot(df2plot, kde=True)
plt.show()
```




```
#sepal_width y sepal_length
df2plot = df [["sepal_width", "sepal_length"]]
sns.histplot(df2plot, kde=True)
plt.show()
```



Parte 3: boxplot para cada variable cuantitativa

```
#Boxplot para petal_length
sns.boxplot(df["petal_length"])
plt.show()
```


```
#Boxplot para petal_width
sns.boxplot(df["petal_width"])
plt.show()
```