Daniel Estrada Ocaña A01369854

Visualizing Data in Python

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 matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd


```

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/Sistemas/4to_semestre/semanaTec/TC1002S"

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

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

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


# Read the .csv file and store it as a pandas Data Frame

df = pd.read_csv(url )

```

Exploring the content of the data set

Get a general 'feel' of the data

```
1 df
```

	Ms_1	Ms_2	Ms_3	Ms_4	Type
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
...	...	...	...	...	...
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

150 rows × 5 columns

Next steps: View recommended plots

```
1 df.shape
```

```
(150, 5)
```

```
1 nrows = df.shape[0]
2 nrows
```

```
150
```

```
1 ncols = df.shape[1]
2 ncols
```

```
5
```

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.

```
1 # Number of times that each distinct value of a variable occurs in a data set
2 ms1 = df.value_counts()
3 ms1
```

```
Ms_1  Ms_2  Ms_3  Ms_4  Type
5.8   2.7   5.1   1.9   Iris-virginica     2
6.2   2.2   4.5   1.5   Iris-versicolor    1
      2.9   4.3   1.3   Iris-versicolor    1
      3.4   5.4   2.3   Iris-virginica     1
6.3   2.3   4.4   1.3   Iris-versicolor    1
                                          ..
5.4   3.9   1.3   0.4   Iris-setosa        1
            1.7   0.4   Iris-setosa        1
5.5   2.3   4.0   1.3   Iris-versicolor    1
      2.4   3.7   1.0   Iris-versicolor    1
7.9   3.8   6.4   2.0   Iris-virginica     1
Length: 149, dtype: int64
```

```
# Proportion of each distinct value of a variable occurs in a data set
propor = df.value_counts() / len(df)
propor
```

```
Ms_1  Ms_2  Ms_3  Ms_4  Type
5.8   2.7   5.1   1.9   Iris-virginica     0.013333
6.2   2.2   4.5   1.5   Iris-versicolor    0.006667
      2.9   4.3   1.3   Iris-versicolor    0.006667
      3.4   5.4   2.3   Iris-virginica     0.006667
6.3   2.3   4.4   1.3   Iris-versicolor    0.006667
                                             ...
5.4   3.9   1.3   0.4   Iris-setosa        0.006667
            1.7   0.4   Iris-setosa        0.006667
5.5   2.3   4.0   1.3   Iris-versicolor    0.006667
      2.4   3.7   1.0   Iris-versicolor    0.006667
7.9   3.8   6.4   2.0   Iris-virginica     0.006667
Length: 149, 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

obser = len(df)
# Total number of null observations

nobser = df.isnull().sum()

# Total number of counts (excluding missing values)
tot = obser-nobser

tot
```

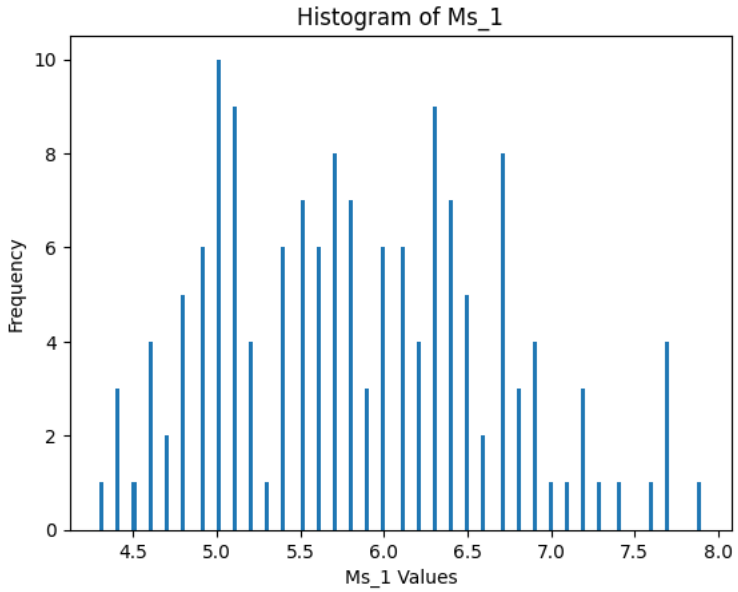
```
Ms_1    150
Ms_2    150
Ms_3    150
Ms_4    150
Type    150
dtype: int64
```

There are no missing values

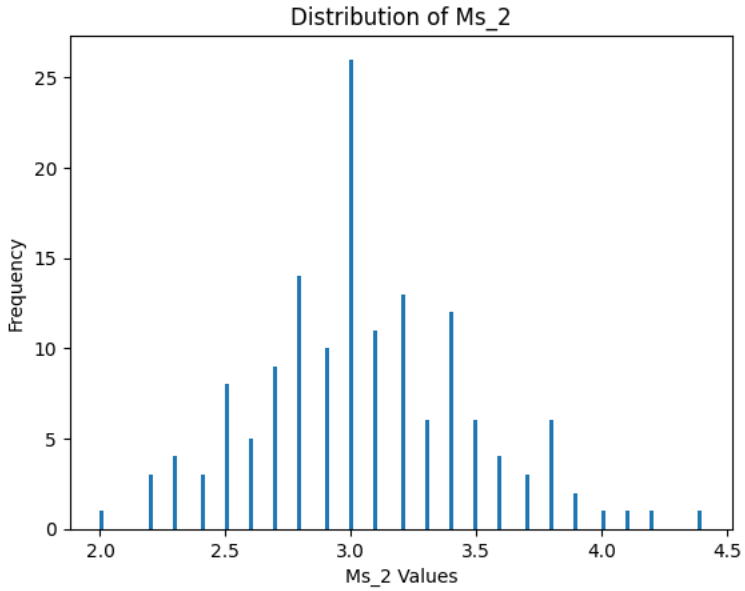
Histogram

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

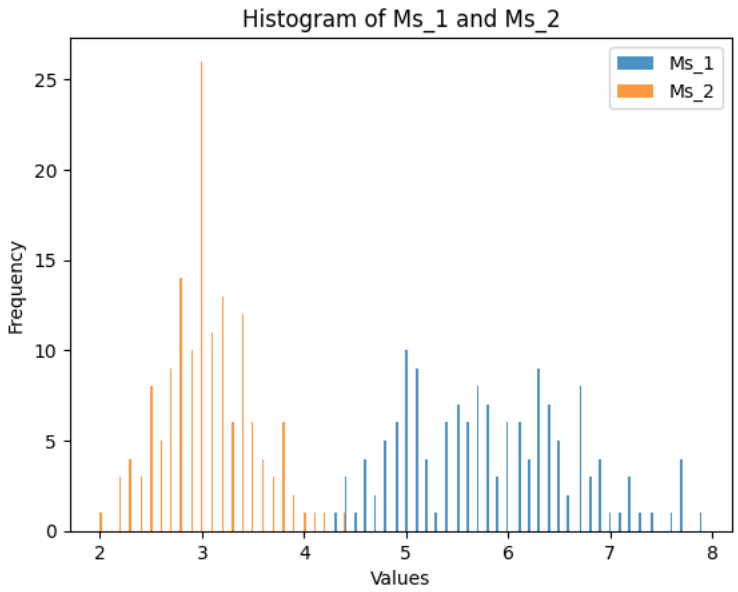
```
# Plot histogram of the Ms_1 only
plt.hist(df['Ms_1'], bins=len(df))  # Adjust the number of bins as needed
plt.title('Histogram of Ms_1')
plt.xlabel('Ms_1 Values')
plt.ylabel('Frequency')
plt.show()

```




```
# Plot distribution of the Ms_2 only
plt.hist(df['Ms_2'], bins=len(df))  # Adjust the number of bins as needed
plt.title('Distribution of Ms_2')
plt.xlabel('Ms_2 Values')
plt.ylabel('Frequency')
plt.show()

```



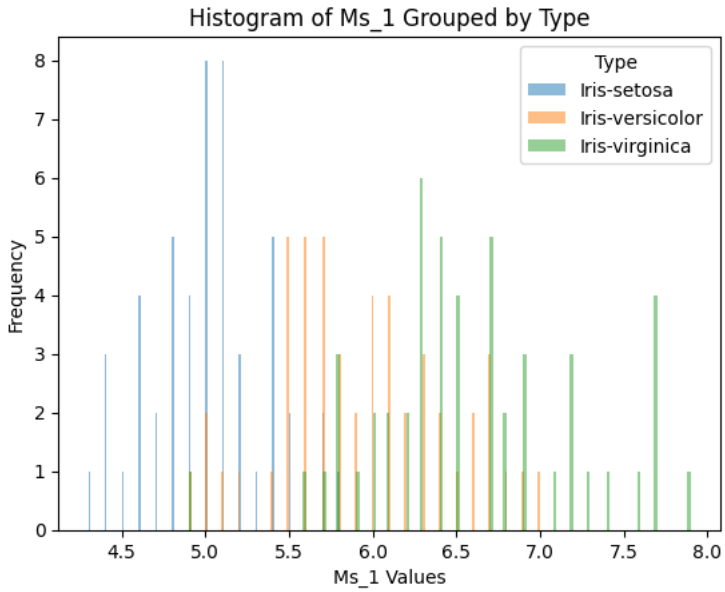

```
# Plot histogram of both the Ms_1 and the Ms_2
plt.hist(df['Ms_1'], bins=len(df), alpha=0.8, label='Ms_1')  # Adjust the number of bins as needed
plt.hist(df['Ms_2'], bins=len(df), alpha=0.8, label='Ms_2')  # Adjust the number of bins as needed
plt.title('Histogram of Ms_1 and Ms_2')
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.legend()
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.

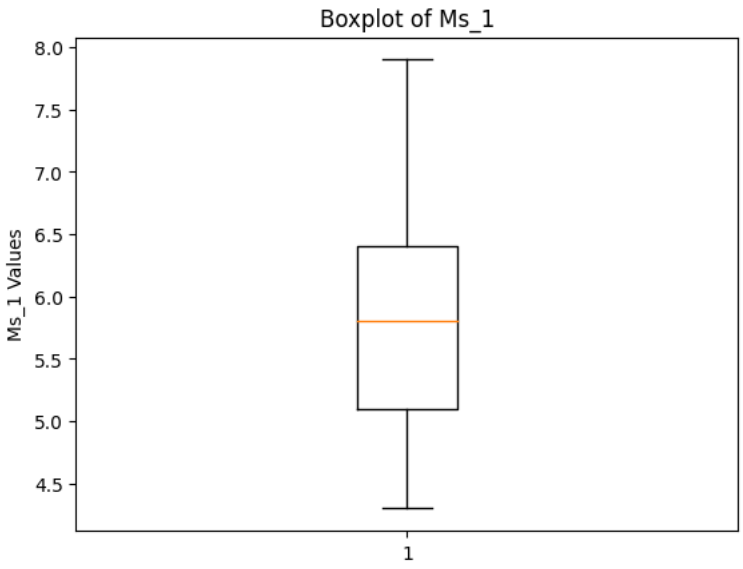
```
# Create histograms of the "Ms_1" grouped by "Type"

for type_name, group_data in df.groupby('Type'):
    plt.hist(group_data['Ms_1'], bins=len(df), alpha=0.5, label=type_name)

# Add labels and legend
plt.title('Histogram of Ms_1 Grouped by Type')
plt.xlabel('Ms_1 Values')
plt.ylabel('Frequency')
plt.legend(title='Type')
plt.show()

```



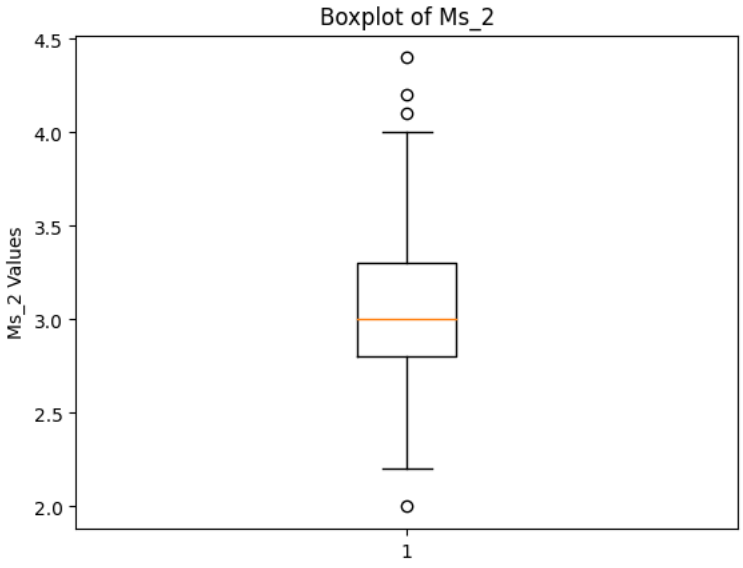
Boxplots

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

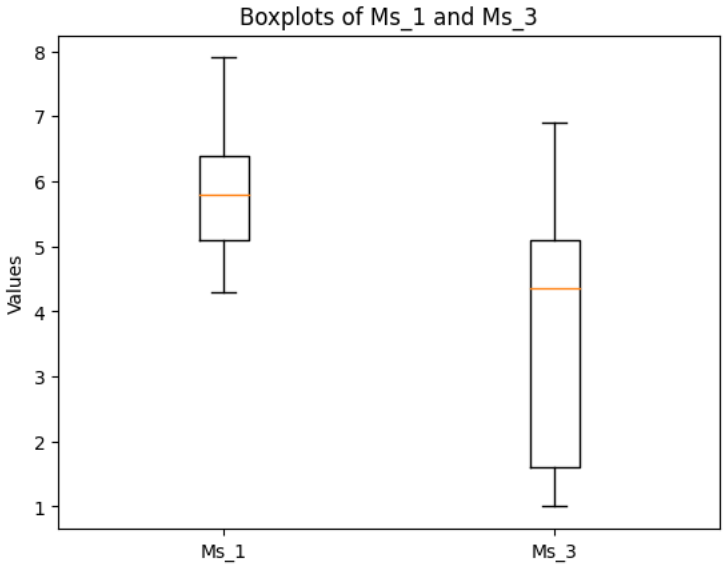
```
# Create the boxplot of the "Ms_1" amounts
plt.boxplot(df['Ms_1'])
plt.title('Boxplot of Ms_1')
plt.ylabel('Ms_1 Values')
plt.show()

```



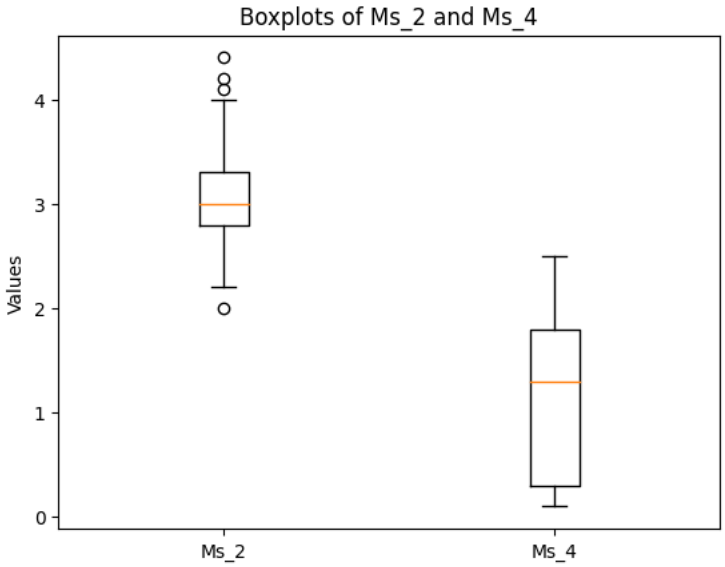

```
# Create the boxplot of the "tips" amounts

plt.boxplot(df['Ms_2'])
plt.title('Boxplot of Ms_2')
plt.ylabel('Ms_2 Values')
plt.show()
```




```
# Create the boxplots of the "Ms_1" and of the "Ms_3" amounts

plt.boxplot([df['Ms_1'], df['Ms_3']])
plt.title('Boxplots of Ms_1 and Ms_3')
plt.ylabel('Values')
plt.xticks([1, 2], ['Ms_1', 'Ms_3'])  # Assign labels to x-axis ticks
plt.show()
```



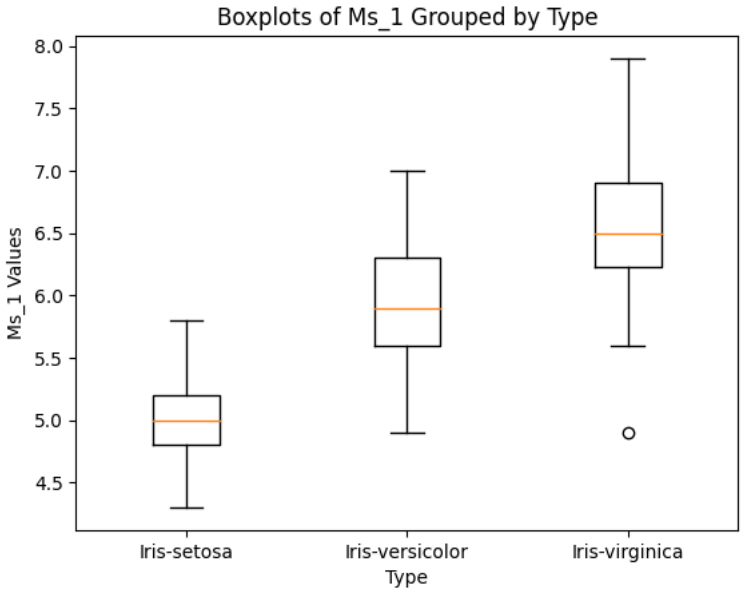

```
# Create the boxplots of the "Ms_2" and of the "Ms_4" amounts
plt.boxplot([df['Ms_2'], df['Ms_4']])
plt.title('Boxplots of Ms_2 and Ms_4')
plt.ylabel('Values')
plt.xticks([1, 2], ['Ms_2', 'Ms_4'])  # Assign labels to x-axis ticks
plt.show()

```



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.

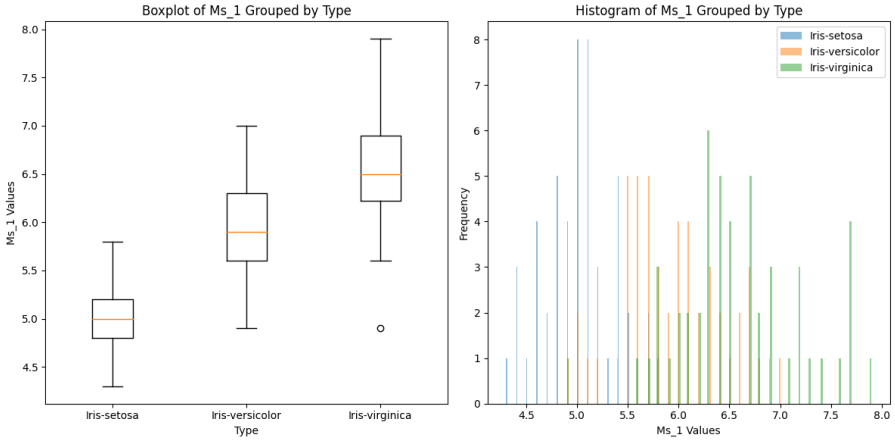
```
# Create side-by-side boxplots of the "Ms_1" grouped by "Type"

grouped_data = [group['Ms_1'] for name, group in df.groupby('Type')]

# Create side-by-side boxplots
plt.boxplot(grouped_data, labels=df['Type'].unique())
plt.title('Boxplots of Ms_1 Grouped by Type')
plt.xlabel('Type')
plt.ylabel('Ms_1 Values')
plt.show()
```



Histograms and boxplots plotted by groups

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

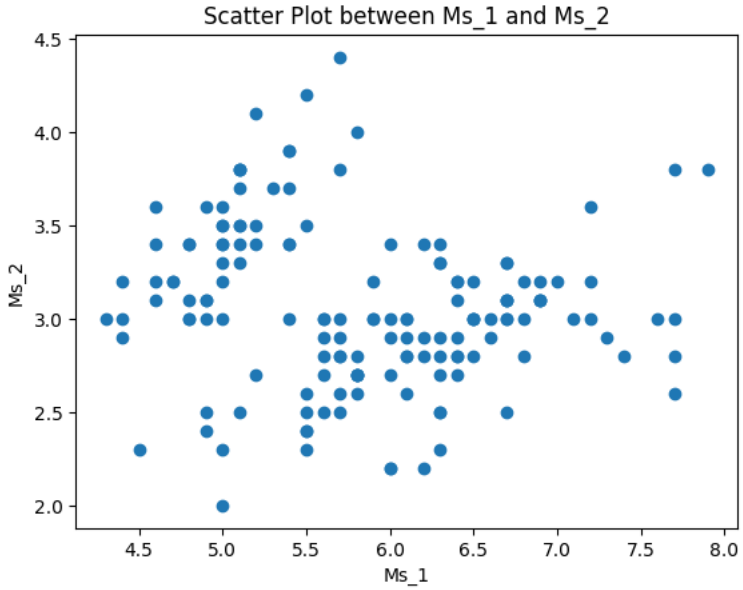
```
# Create a boxplot and histogram of the "Ms_1" grouped by "Type"
# Create subplots with 1 row and 2 columns
fig, axs = plt.subplots(1, 2, figsize=(12, 6))

# Group the data by 'Type'
grouped_data = [group['Ms_1'] for name, group in df.groupby('Type')]

# Create boxplot
axs[0].boxplot(grouped_data, labels=df['Type'].unique())
axs[0].set_title('Boxplot of Ms_1 Grouped by Type')
axs[0].set_xlabel('Type')
axs[0].set_ylabel('Ms_1 Values')

# Create histogram
for name, group in df.groupby('Type'):
    axs[1].hist(group['Ms_1'], bins=len(df), alpha=0.5, label=name)
axs[1].set_title('Histogram of Ms_1 Grouped by Type')
axs[1].set_xlabel('Ms_1 Values')
axs[1].set_ylabel('Frequency')
axs[1].legend()

plt.tight_layout()
plt.show()

```



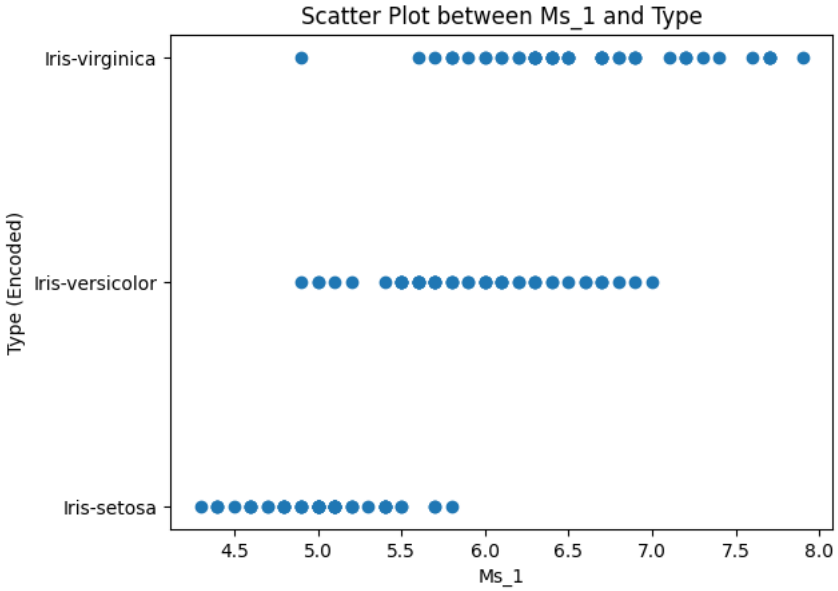
Scatter plot

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

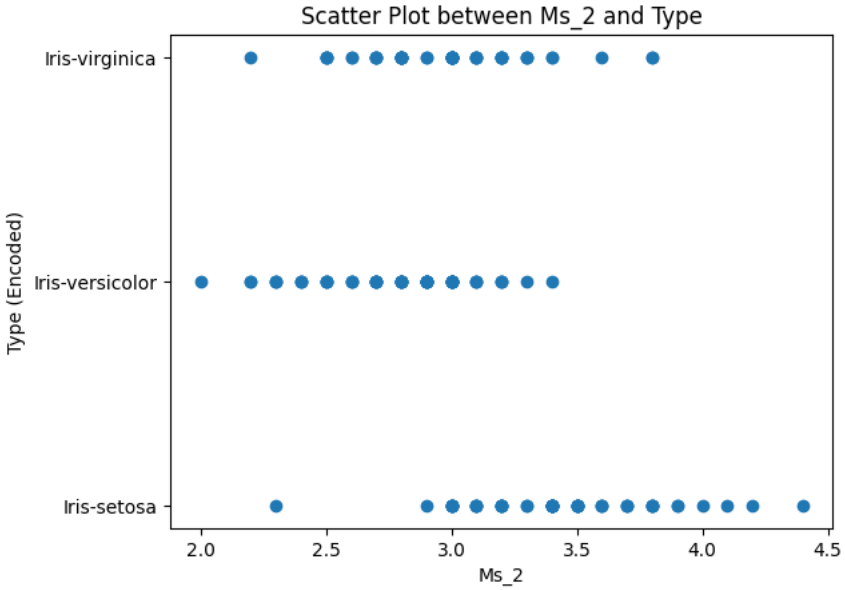
```
# scatter plot between two variables

plt.scatter(df['Ms_1'], df['Ms_2'])
plt.title('Scatter Plot between Ms_1 and Ms_2')
plt.xlabel('Ms_1')
plt.ylabel('Ms_2')
plt.show()
```



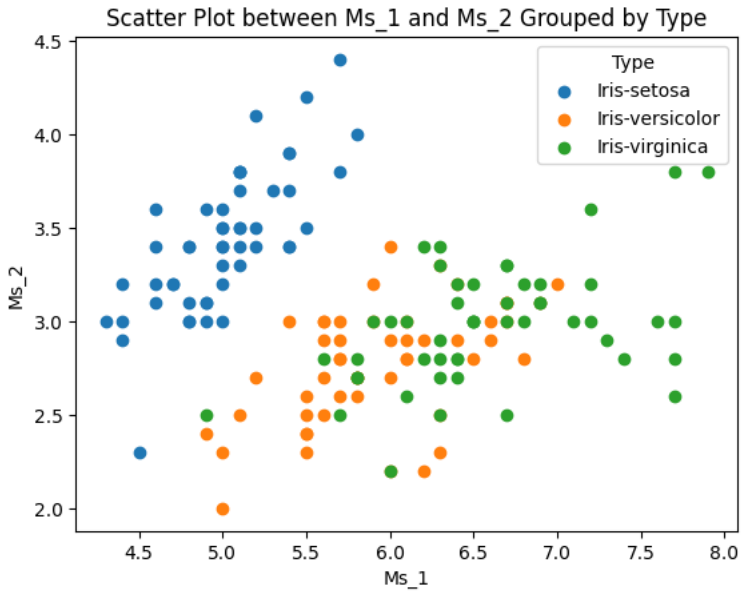
```
# scatter plot between two variables (one categorical)
df["Type encoded"] = df.Type.replace({"Iris-setosa": 1,"Iris-versicolor": 2, "Iris-virginica": 3})

plt.scatter(df['Ms_1'], df['Type encoded'])
plt.title('Scatter Plot between Ms_1 and Type')
plt.xlabel('Ms_1')
plt.ylabel('Type (Encoded)')
plt.yticks(df['Type encoded'].unique(), df['Type'].unique())
plt.show()

```



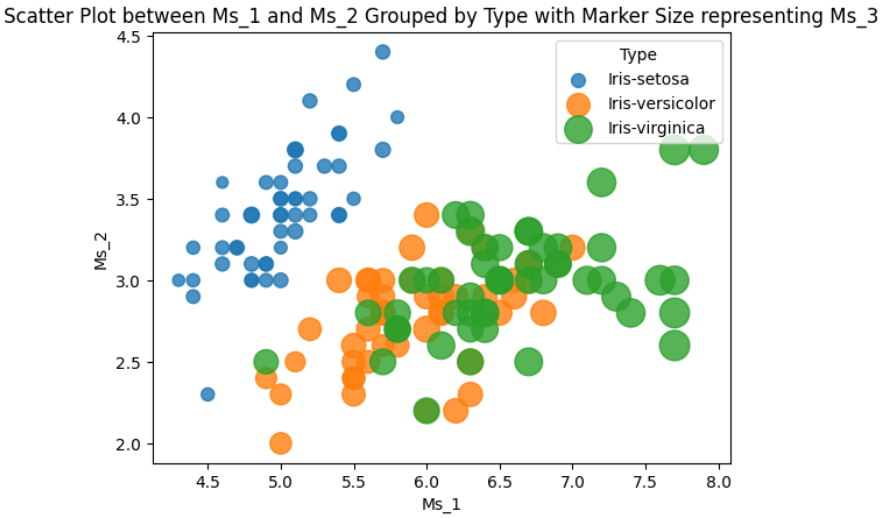

```
# scatter plot between two variables (one categorical)
df["Type encoded"] = df.Type.replace({"Iris-setosa": 1,"Iris-versicolor": 2, "Iris-virginica": 3})

plt.scatter(df['Ms_2'], df['Type encoded'])
plt.title('Scatter Plot between Ms_2 and Type')
plt.xlabel('Ms_2')
plt.ylabel('Type (Encoded)')
plt.yticks(df['Type encoded'].unique(), df['Type'].unique())
plt.show()

```



```
# scatter plot between two variables grouped according to a categorical variable

grouped_data = df.groupby('Type')


for name, group in grouped_data:
    plt.scatter(group['Ms_1'], group['Ms_2'], label=name)


plt.title('Scatter Plot between Ms_1 and Ms_2 Grouped by Type')
plt.xlabel('Ms_1')
plt.ylabel('Ms_2')
plt.legend(title='Type')
plt.show()
```




```
# scatter plot between two variables grouped according to a categorical variable and with size of markers


grouped_data = df.groupby('Type')


for name, group in grouped_data:
    plt.scatter(group['Ms_1'], group['Ms_2'], s=group['Ms_3']*50, alpha=0.8, label=name)


plt.title('Scatter Plot between Ms_1 and Ms_2 Grouped by Type with Marker Size representing Ms_3')
plt.xlabel('Ms_1')
plt.ylabel('Ms_2')
plt.legend(title='Type')
plt.show()
```



Final remarks

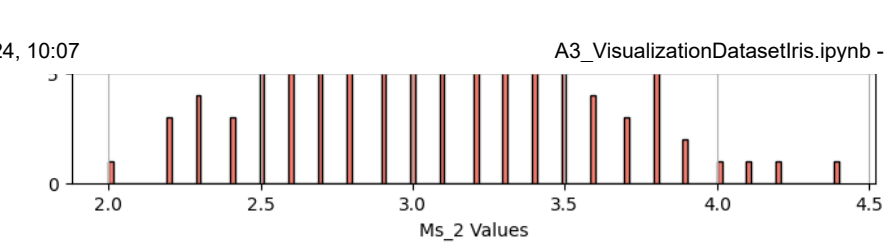
- Visualizing your data using **tables**, **histograms**, **boxplots**, **scatter plots** and other tools is essential to carry put analysis and extract conclusions
- There are several ways to do the same thing
- The **Seaborn** package provides visualisations tools that allow to explore data from a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

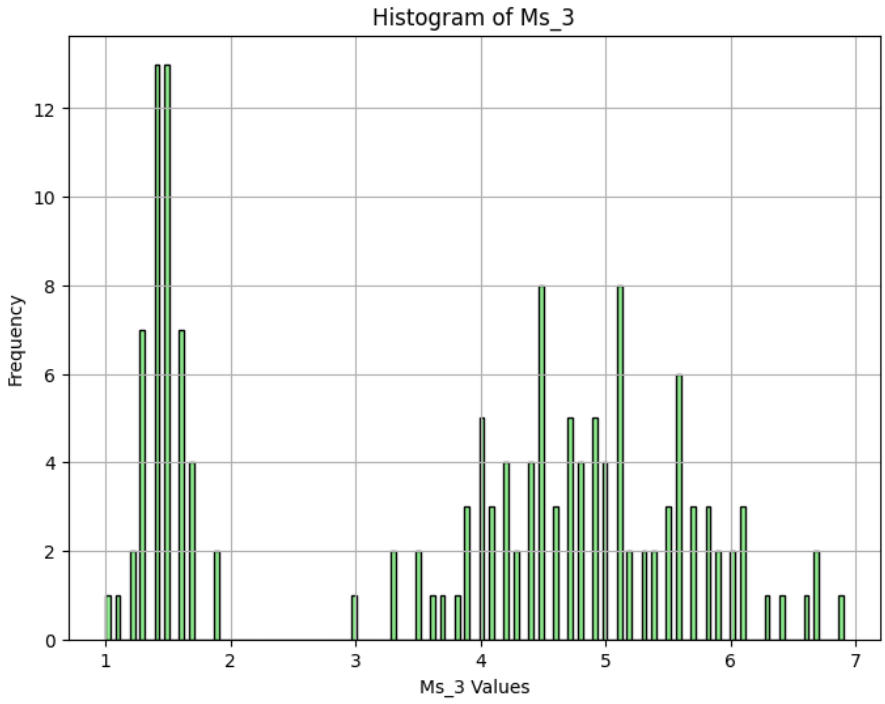
Activity: work with the iris dataset

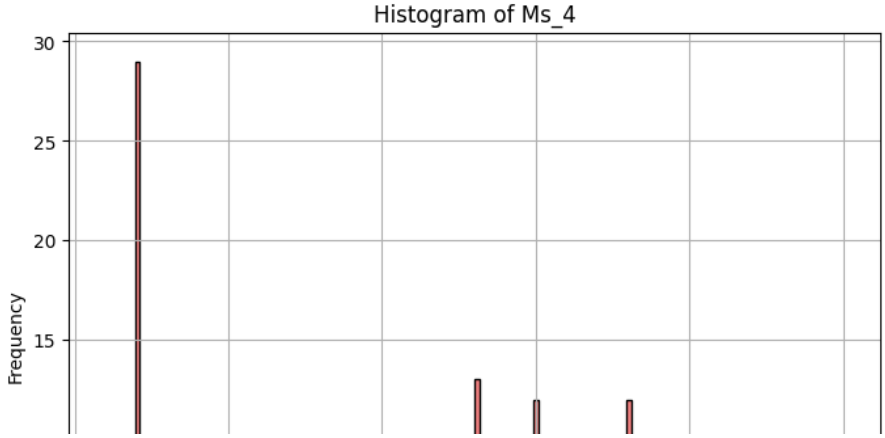
Repeat this tutorial with the iris data set and respond to the following inquiries

1. Plot the histograms for each of the four quantitative variables

```
# Histogram of Ms_1
plt.figure(figsize=(8, 6))
plt.hist(df['Ms_1'], bins=len(df), color='skyblue', edgecolor='black')
plt.title('Histogram of Ms_1')
plt.xlabel('Ms_1 Values')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# Histogram of Ms_2
plt.figure(figsize=(8, 6))
plt.hist(df['Ms_2'], bins=len(df), color='salmon', edgecolor='black')
plt.title('Histogram of Ms_2')
plt.xlabel('Ms_2 Values')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# Histogram of Ms_3
plt.figure(figsize=(8, 6))
plt.hist(df['Ms_3'], bins=len(df), color='lightgreen', edgecolor='black')
plt.title('Histogram of Ms_3')
plt.xlabel('Ms_3 Values')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# Histogram of Ms_4
plt.figure(figsize=(8, 6))
plt.hist(df['Ms_4'], bins=len(df), color='lightcoral', edgecolor='black')
plt.title('Histogram of Ms_4')
plt.xlabel('Ms_4 Values')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```


0
2.0
2.5
3.0
3.5
4.0
4.5
Ms_2 Values


Histogram of Ms_3
Frequency
0
2
4
6
8
10
12
1
2
3
4
5
6
7
Ms_3 Values


Histogram of Ms_4
Frequency
15
20
25
30

2. Plot the histograms for each of the quantitative variables

```
fig, axs = plt.subplots(2, 2, figsize=(12, 8))

# Flatten the axs array for easier iteration
axs = axs.flatten()

# Plot histograms for each variable
for i, column in enumerate(['Ms_1', 'Ms_2', 'Ms_3', 'Ms_4']):
    axs[i].hist(df[column], bins=len(df), color='skyblue', edgecolor='black')
    axs[i].set_title(f'Histogram of {column}')
    axs[i].set_xlabel(f'{column} Values')
    axs[i].set_ylabel('Frequency')

# Adjust layout
plt.tight_layout()
plt.show()
```