Course: Artificial Intelligence and Machine Learning

Code: 20CS51I, WEEK - 4

SESSION – 3 Exploratory data analysis

- Overview

- EDA goals and benefits

Univariate data analysis

- Characterizing data with descriptive statistics

- Univariate distribution plots

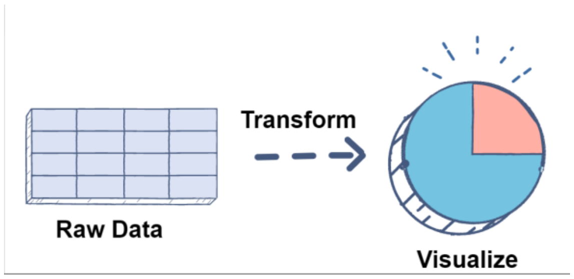
- Univariate comparison plots

- Univariate composition plots

Exploratory data analysis

- overview

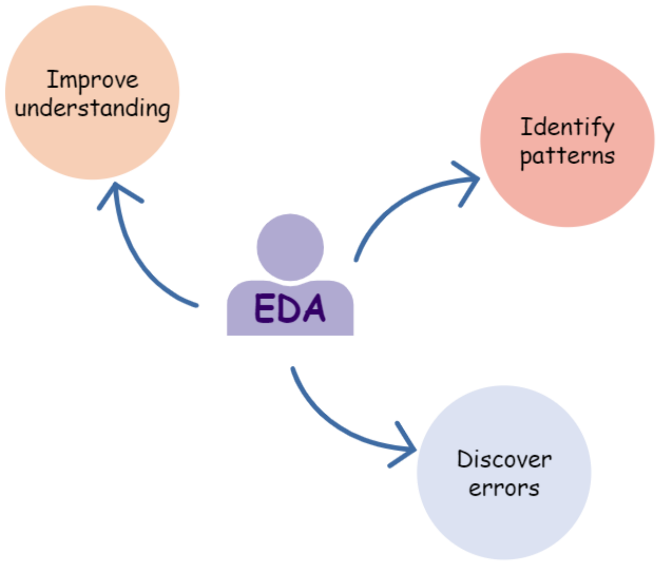
Exploratory Data Analysis (EDA) is a way to investigate datasets and find preliminary information, insights, or uncover underlying patterns in the data. Instead of making assumptions, data can be processed in a systematic method to gain insights and make informed decisions.



Why Exploratory Data Analysis?

Some advantages of Exploratory Data Analysis include:

- Improve understanding of variables by extracting averages, mean, minimum, and maximum values, etc.
- Discover errors, outliers, and missing values in the data.
- Identify patterns by visualizing data in graphs such as box plots, scatter plots, and histograms.
- Hence, the main goal is to understand the data better and use tools effectively to gain valuable insights or draw conclusions.



The Advantages of Exploratory Data Analysis

Statistics

The first step in data analysis is to observe the statistical values of the data to decide if it needs to be preprocessed in order to make it more consistent

Describe

The describe() method of a pandas data frame gives us important statistics of the data like min, max, mean, standard deviation, and quartiles. For example, we want to verify the minimum and maximum values in our data. This can be done by invoking the describe() method:

Data cleaning

Removing nulls: In order to identify the number of nulls within each column, we can invoke the isnull() method on each column of the pandas data frame. If null values are found within a column, they can be replaced with the column mean using the fillna() method:

Data visualizations

As human beings, it is difficult to visualize statistical values. As an alternative, visualizations can be utilized in order to better understand the data and detect patterns.

- EDA goals and benefits

Goals

The goal of EDA is to allow data scientists to get deep insight into a data set and at the same time provide specific outcomes that a data scientist would want to extract from the data set. It includes:

- List of outliers
- Estimates for parameters
- Uncertainties about those estimates
- List of all important factors
- Conclusions or assumptions as to whether certain individual factors are statistically essential
- Optimal settings
- A good predictive model

The following are some benefits of an EDA:

Detecting missing or inaccurate data

A data clean-up in the early stages of Exploratory Data Analysis may help you discover any faults in the dataset during the analysis. It will alert you if you need to modify the data or collect new data entirely before continuing with the deep analysis.

Testing your hypothesis

It is critical to ensure that any assumptions or hypotheses you are working on can withstand inspection. It will assist you in determining if you are inferring the correct results based on your knowledge of the facts. If not, you know your assumptions are incorrect or you're asking the wrong questions about the dataset.

Developing the most effective model

During the analysis, any unnecessary information must be removed. This is due to the fact that extraneous data might either distort your results or just hide crucial insights with unneeded noise.

Error detection

Exploratory Data Analysis assists in determining whether data may result in inevitable mistakes in your subsequent analysis. Knowing which facts will have an influence on your results can assist you to avoid accepting erroneous conclusions or mistakenly identifying an outcome.

Assisting in choosing the right tool

Exploratory Data Analysis will assist you in determining which approaches and statistical models will assist you in extracting the information you want from your dataset. Intuition and reflection are essential abilities for doing exploratory data analysis. While EDA may entail the execution of predefined tasks, it is the interpretation of the outcomes of these activities that is the true talent.

Exploratory Data Analysis greatly helps data scientists guarantee that the results they create are legitimate and appropriate to any targeted business outcomes and goals. When EDA is finished and insights are obtained, its characteristics can be used for more complex data analysis or modeling, including machine learning. Aspiring data analysts might consider taking a complete curriculum in data analytics to gain critical skills relating to tools, methodologies, strategies, and frequently used computer languages for exploratory data analysis.

Univariate data analysis

Data visualization is the process of representing data using visual elements like charts, graphs, etc. that helps in deriving meaningful insights from the data. It is aimed at revealing the information behind the data and further aids the viewer in seeing the structure in the data.

Data visualization will make the scientific findings accessible to anyone with minimal exposure in data science and helps one to communicate the information easily. It is to be understood that the visualization technique one employs for a particular data set depends on the individual's taste and preference. Let's first understand univariate analysis in python and how to perform univariate analysis in python.

Need for visualizing data :

- Understand the trends and patterns of data
- Analyze the frequency and other such characteristics of data
- Know the distribution of the variables in the data.
- Visualize the relationship that may exist between different variables

The number of variables of interest featured by the data classifies it as univariate, bivariate, or multivariate. For eg., If the data features only one variable of interest then it is a uni-variate data. Further, based on the characteristics of data, it can be classified as categorical/discrete and continuous data.

In this article, the main focus is on univariate data visualization(data is visualized in one-dimension). For the purpose of illustration, the 'iris' data set is considered. The iris data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. The different variables involved in the data set are Sepal Length, Sepal Width, Petal Length, Petal width which is continuous and Variety which is a categorical variable. Though the data set is multivariate in nature, for univariate analysis, we consider one variable of interest at a time.

Univariate Analysis

- Analysis of a single variable in your data
- A variable can be a column in your dataset
- We can use various graphs like

- Histograms, Barcharts, Density plots, Distribution plots, and so on
- We can use pandas plotting functionality, seaborn library for plotting data.

These plots help in understanding the location/position of observations in the data variable, its distribution, and dispersion. Uni-variate plots are of two types: 1)Enumerative plots and 2)Summary plots These plots enumerate/show every observation in data and provide information about the distribution of the observations on a single data variable.

- Characterizing data with descriptive statistics

Types of variables

- Categorical variable:
    - These have values in the form of string or text
    - These even include values like dates, gender and so on
- Numerical variable
    - These have numerical values like age, stock value, and so on
    - Basically any value that can be measured

Categorical values can be further divided into two types:

Nominal variables:

Simply put, these are values which have no order amongst them, that is no value is greater than the other or can be arranged in any order with respect to one another.

For example weather, country, and so on.

Ordinal variable:

These categorical variables can be arranged in some order with respect to one another

For example, rank in a competition, educational level and so on.

Numerical variable can be further divided into two types:

Continuous variable:

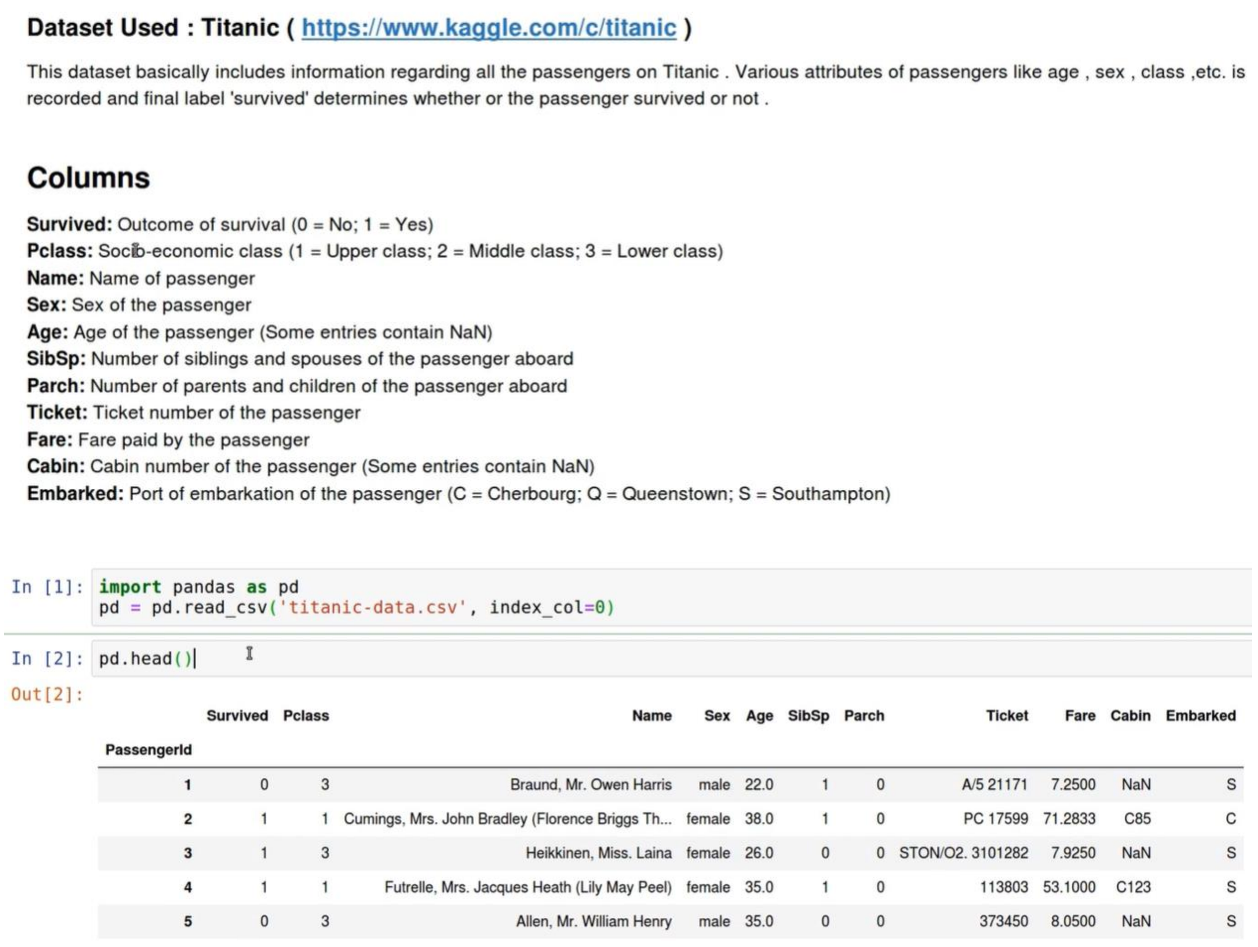
These have infinite values

For example, height, weight, age, and so on

Discrete variable:

These have finite values

For example the number of gold medalswon by an athlete etc



Dataset Used : Titanic (https://www.kaggle.com/c/titanic)

This dataset basically includes information regarding all the passengers on Titanic . Various attributes of passengers like age , sex , class ,etc. is recorded and final label 'survived' determines whether or the passenger survived or not .

Columns

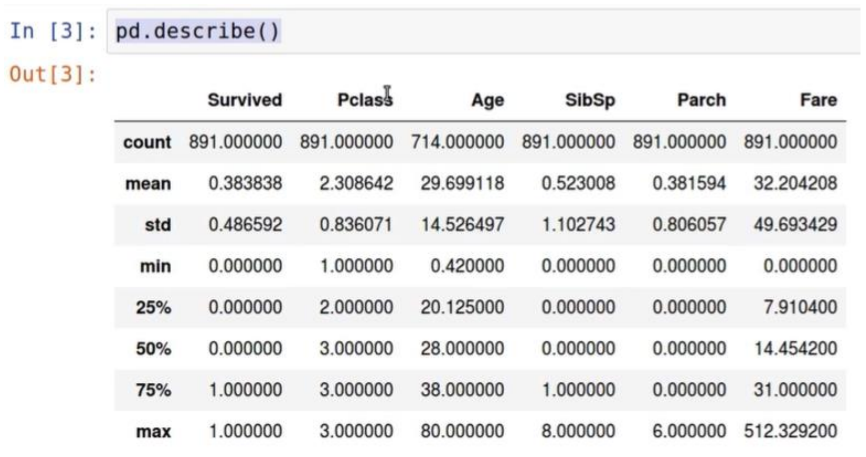
Survived: Outcome of survival (0 = No; 1 = Yes)
Pclass: Socio-economic class (1 = Upper class; 2 = Middle class; 3 = Lower class)
Name: Name of passenger
Sex: Sex of the passenger
Age: Age of the passenger (Some entries contain NaN)
SibSp: Number of siblings and spouses of the passenger aboard
Parch: Number of parents and children of the passenger aboard
Ticket: Ticket number of the passenger
Fare: Fare paid by the passenger
Cabin: Cabin number of the passenger (Some entries contain NaN)
Embarked: Port of embarkation of the passenger (C = Cherbourg; Q = Queenstown; S = Southampton)

```
In [1]: import pandas as pd
        pd = pd.read_csv('titanic-data.csv', index_col=0)
```

```
In [2]: pd.head()
```

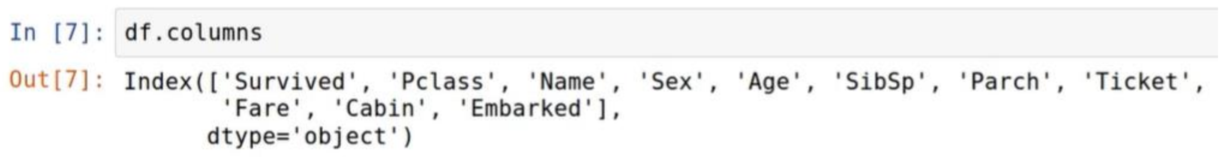
Out[2]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S



```
In [3]: pd.describe()
Out[3]:
```

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200



```
In [7]: df.columns
Out[7]: Index(['Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket',
               'Fare', 'Cabin', 'Embarked'],
              dtype='object')
```

Categorical Variables are name, Cabin, Sex, Embarked, Survived, Pclass

Nominal Variable : Sex

Ordinal Variable : Pclass

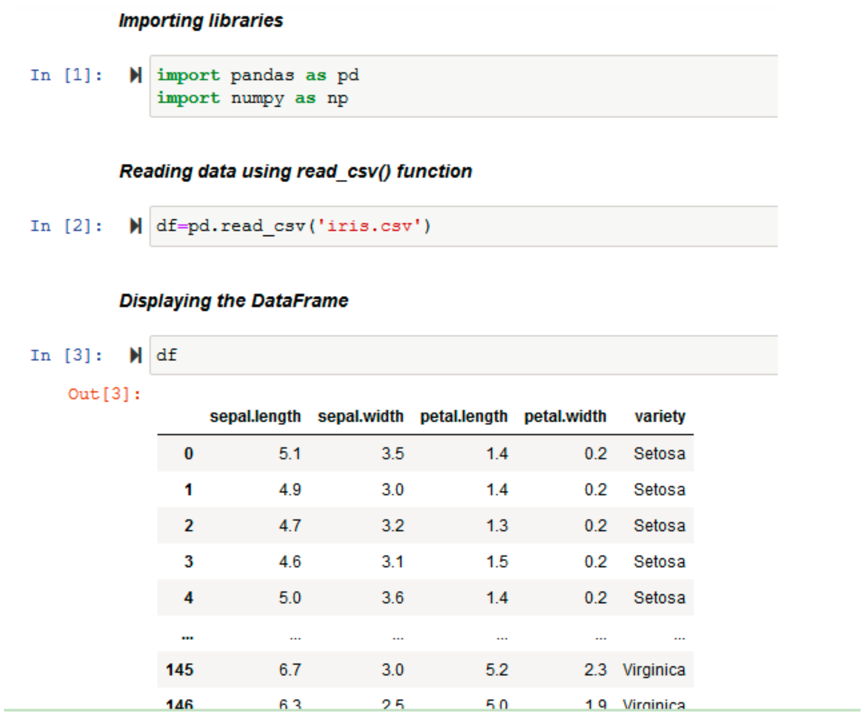
Numerical Variables are age, fare, parch, SibSp

Age and Fare are Continuous variables

Parch and SibSp are Discrete Variables

Descriptive Statistics in data science -with illustrations in ‘python’
https://medium.com/analytics-vidhya/descriptive-statistics-in-data-science-with-illustrations-in-python-efd5ccc152f1

Univariate distribution plots



Importing libraries

```
In [1]:  import pandas as pd
         import numpy as np
```

Reading data using read_csv() function

```
In [2]:  df=pd.read_csv('iris.csv')
```

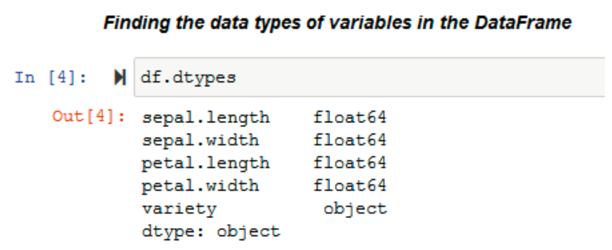
Displaying the DataFrame

```
In [3]:  df
Out[3]:
```

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Virginica
146	6.3	2.5	5.0	1.9	Virginica

The data set originally in .csv format is loaded into the DataFrame df using the *pd.read_csv( )* function of pandas . Then, it displays the DataFrame df.

Before analyzing any data set, inspect the data types of the data variables. Then, one can decide on the right methods for univariate data visualization.



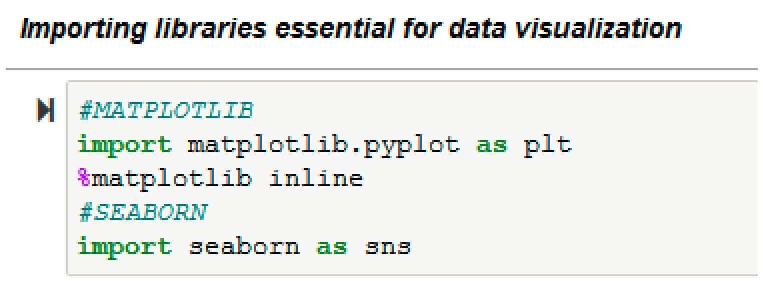
Finding the data types of variables in the DataFrame

```
In [4]:  df.dtypes

Out[4]:  sepal.length    float64
         sepal.width     float64
         petal.length    float64
         petal.width     float64
         variety          object
         dtype: object
```

The *.dtypes* property is used to know the data types of the variables in the data set. Pandas stores these variables in different formats according to their type. Pandas stores *categorical variables as 'object'* and, on the other hand, *continuous variables are stored as int or float*. The methods used for visualization of univariate data also depends on the types of data variables.

In this article, we visualize the iris data using the libraries: matplotlib and seaborn. We use Matplotlib library to draw basic plots. Seaborn library is based on the matplotlib library and it provides a wide variety of visualization techniques for univariate data.



Importing libraries essential for data visualization

```
#MATPLOTLIB
import matplotlib.pyplot as plt
%matplotlib inline
#SEABORN
import seaborn as sns
```

Visualizing Univariate Continuous Data :

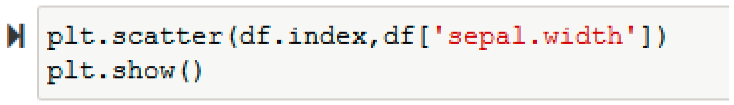
Univariate data visualization plots help us comprehend the enumerative properties as well as a descriptive summary of the particular data variable. These plots help in understanding the **location/position** of observations in the data variable, its **distribution**, and **dispersion**. Uni-variate plots are of two types: 1)Enumerative plots and 2)Summary plots

Univariate enumerative Plots :

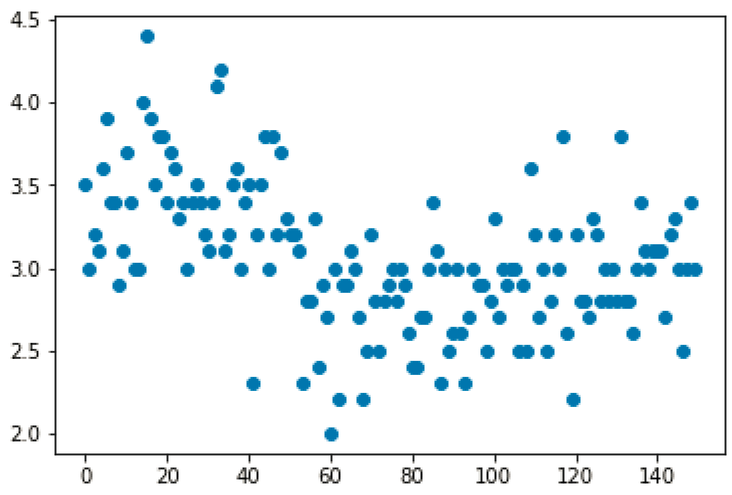
These plots enumerate/show every observation in data and provide information about the distribution of the observations on a single data variable. We now look at different enumerative plots.

1. Univariate Scatter Plot :

This plots different observations/values of the same variable corresponding to the index/observation number. Consider plotting of the variable ‘sepal length(cm)’ :



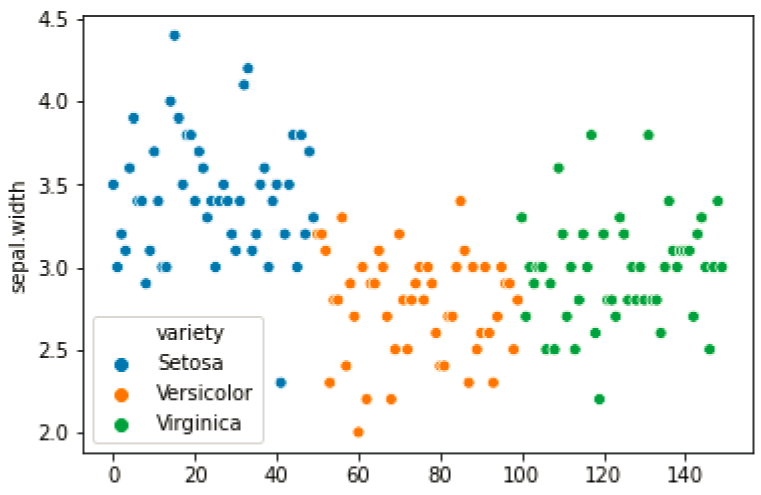
```
plt.scatter(df.index,df['sepal.width'])
plt.show()
```



Use the *plt.scatter()* function of matplotlib to plot a univariate scatter diagram. The scatter() function requires two parameters to plot. So, in this example, we plot the variable 'sepal.width' against the corresponding observation number that is stored as the index of the data frame (df.index).

Then visualize the same plot by considering its variety using the *sns.scatterplot()* function of the seaborn library.

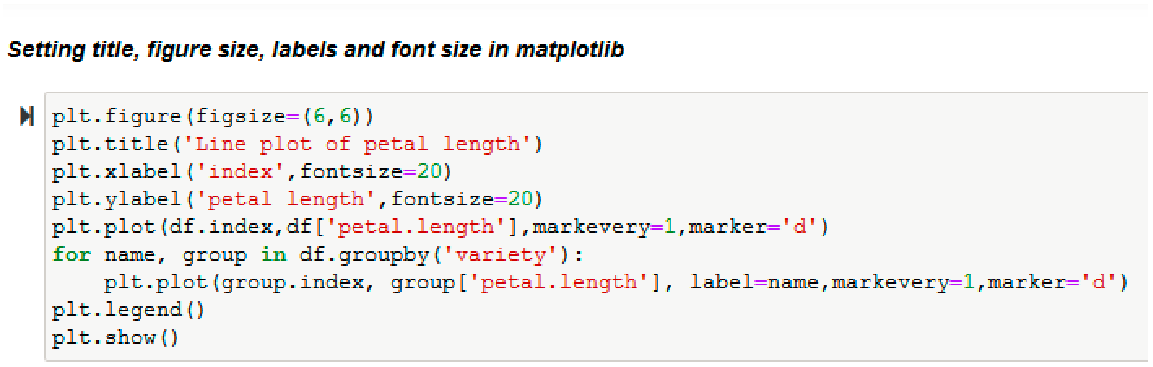
```
sns.scatterplot(x=df.index,y=df['sepal.width'],hue=df['variety'])
```



One of the interesting features in seaborn is the 'hue' parameter. In seaborn, the hue parameter determines which column in the data frame should be used for color encoding. This helps to differentiate between the data values according to the categories they belong to. The hue parameter takes the grouping variable as it's input using which it will produce points with different colors. The variable passed onto 'hue' can be either categorical or numeric, although color mapping will behave differently in the latter case.

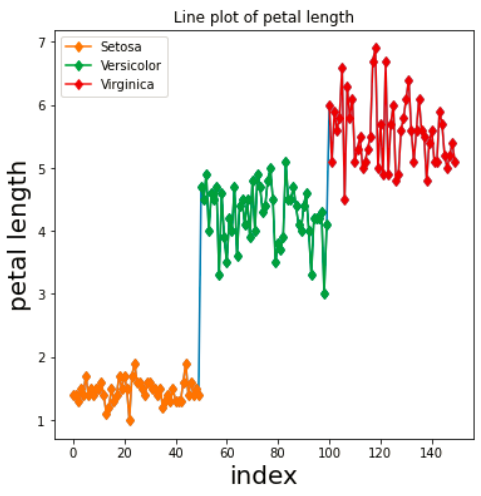
2. Line Plot (with markers) :

A line plot visualizes data by connecting the data points via line segments. It is similar to a scatter plot except that the measurement points are ordered (typically by their x-axis value) and joined with straight line segments.



Setting title, figure size, labels and font size in matplotlib

```
plt.figure(figsize=(6,6))
plt.title('Line plot of petal length')
plt.xlabel('index',fontsize=20)
plt.ylabel('petal length',fontsize=20)
plt.plot(df.index,df['petal.length'],markevery=1,marker='d')
for name, group in df.groupby('variety'):
    plt.plot(group.index, group['petal.length'], label=name,markevery=1,marker='d')
plt.legend()
plt.show()
```

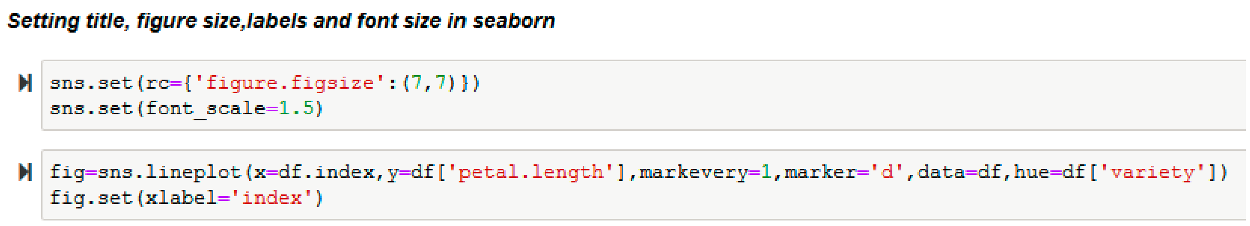


The matplotlib *plt.plot()* function by default plots the data using a line plot. One can use the *groupby()* function of pandas which helps in plotting such a graph.

— Explanation of the functions used :

- *plt.figure(figsize=())* : To set the size of figure
- *plt.title()* : To set title
- *plt.xlabel() / plt.ylabel()* : To set labels on X-axis/Y-axis
- *df.groupby( )* : To group the rows of the data frame according to the parameter passed onto the function
- The groupby() function returns the data frames grouped by the criterion variable passed and the criterion variable.
- The *for loop* is used to plot each data point according to its variety.
- *plt.legend()*: Adds a legend to the graph (Legend describes the different elements seen in the graph).
- *plt.show()* : to show the plot.

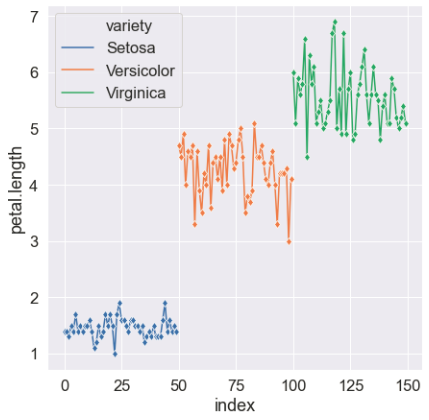
The 'markevery' parameter of the function *plt.plot()* is assigned to '1' which means it will plot every 1st marker starting from the first data point. There are various marker styles which we can pass as a parameter to the function. The *sns.lineplot()* function can also visualize the line plot.



Setting title, figure size,labels and font size in seaborn

```
sns.set(rc={'figure.figsize':(7,7)})
sns.set(font_scale=1.5)
```

```
fig=sns.lineplot(x=df.index,y=df['petal.length'],markevery=1,marker='d',data=df,hue=df['variety'])
fig.set(xlabel='index')
```



In seaborn, the labels on axes are automatically set based on the columns that are passed for plotting. However if one desires to change it, it is possible too using the set() function.

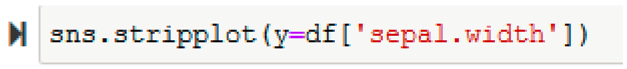
Univariate comparison plots

There are often cases where in one would want to explore how the distribution of a single continuous variable is affected by a second categorical variable. The seaborn library provides a variety of plots that help perform such types of comparisons between uni-variate distributions. Three such plots are Strip plot, Swarm plot (under enumerative plots), and Violin plot (under summary plots). The hue parameter mentioned in above plots is also for similar use.

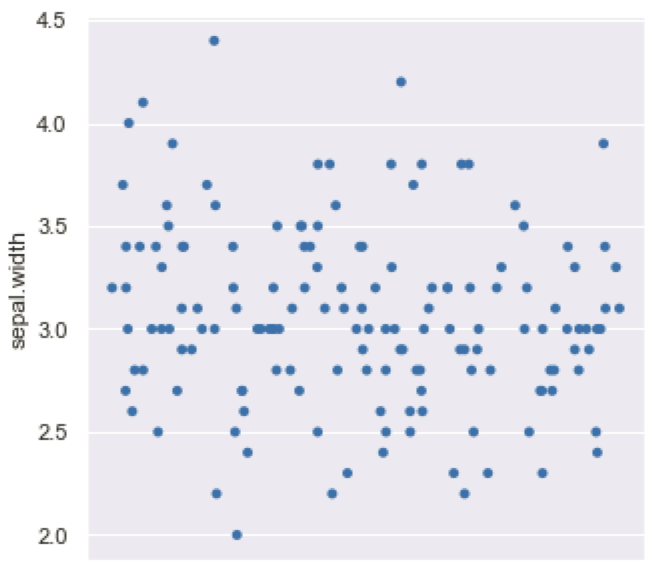
3. STRIP PLOT :

The strip plot is similar to a scatter plot. It is often used along with other kinds of plots for better analysis. It is used to visualize the distribution of data points of the variable.

The *sns.striplot ( )* function is used to plot a strip-plot :

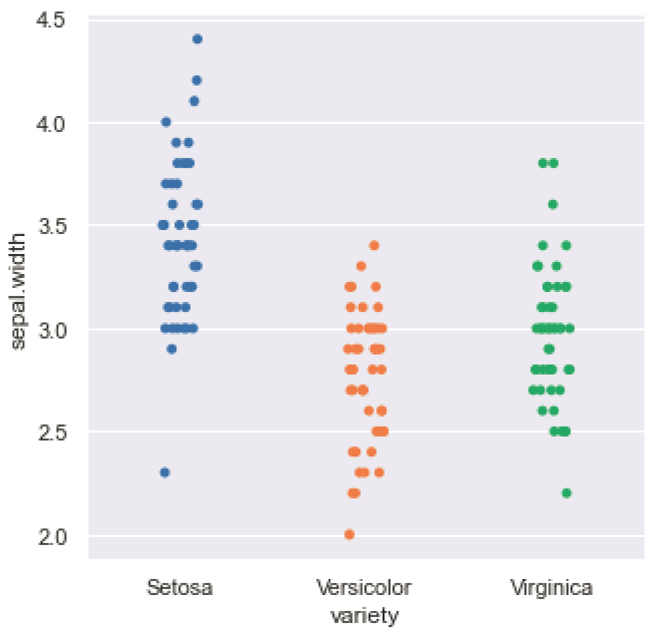


```
sns.stripplot(y=df['sepal.width'])
```



It also helps to plot the distribution of variables for each category as individual data points. By default, the function creates a vertical strip plot where the distributions of the continuous data points are plotted along the Y-axis and the categories are spaced out along the X-axis. In the above plot, categories are not considered. Considering the categories helps in better visualization as seen in the below plot.

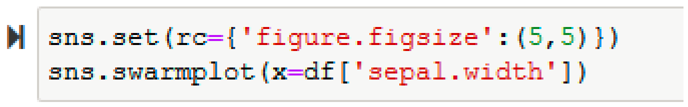
```
sns.stripplot(x=df['variety'],y=df['sepal.width'])
```



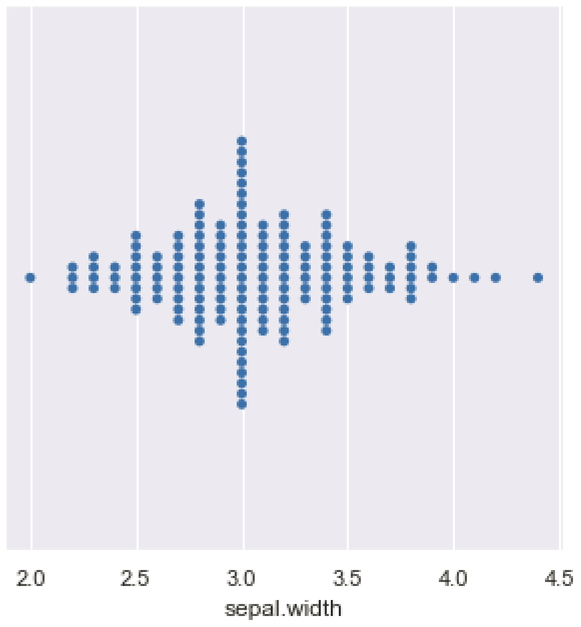
4. SWARM PLOT :

The swarm-plot, similar to a strip-plot, provides a visualization technique for univariate data to view the spread of values in a continuous variable. The only difference between the strip-plot and the swarm-plot is that the swarm-plot spreads out the data points of the variable automatically to avoid overlap and hence provides a better visual overview of the data.

The *sns.swarmplot( )* function is used to plot a swarm-plot :

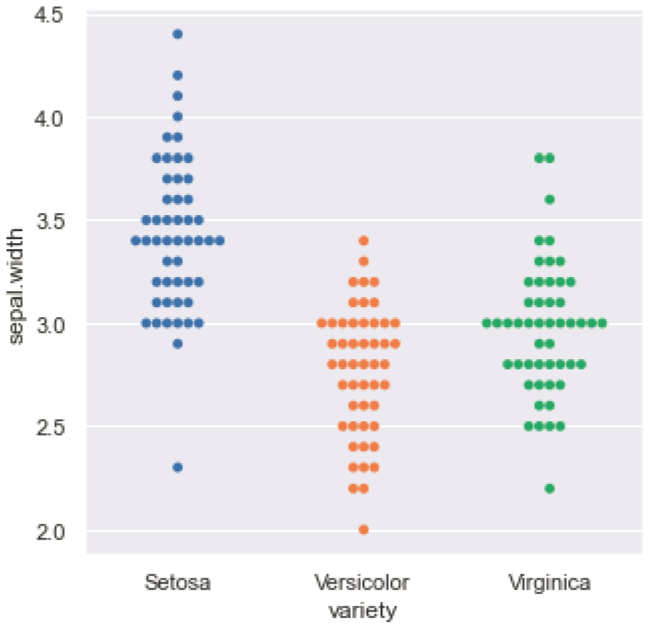


```
sns.set(rc={'figure.figsize':(5,5)})
sns.swarmplot(x=df['sepal.width'])
```



Distribution of the variable 'sepal.width' according to the categories :

```
sns.swarmplot(x=df['variety'],y=df['sepal.width'])
```



Univariate composition/**summary** plots

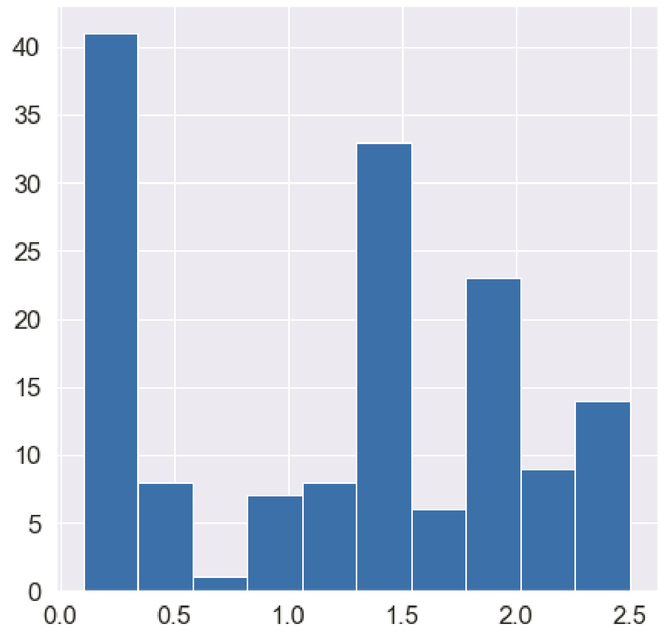
These plots give a more concise description of the location, dispersion, and distribution of a variable than an enumerative plot. It is not feasible to retrieve every individual data value in a summary plot, but it helps in efficiently representing the whole data from which better conclusions can be made on the entire data set.

5. Histograms :

Histograms are similar to bar charts which display the counts or relative frequencies of values falling in different class intervals or ranges. A histogram displays the shape and spread of continuous sample data. It also helps us understand the skewness and kurtosis of the distribution of the data.

Plotting histogram using the matplotlib *plt.hist()* function

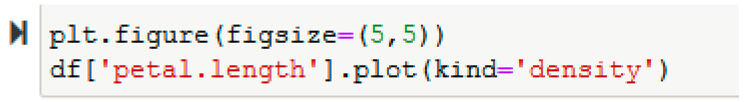
```
In [12]:  plt.hist(df['petal.width'])
Out[12]: (array([41.,  8.,  1.,  7.,  8., 33.,  6., 23.,  9., 14.]),
          array([0.1 , 0.34, 0.58, 0.82, 1.06, 1.3 , 1.54, 1.78, 2.02, 2.26, 2.5 ]),
          <a list of 10 Patch objects>)
```



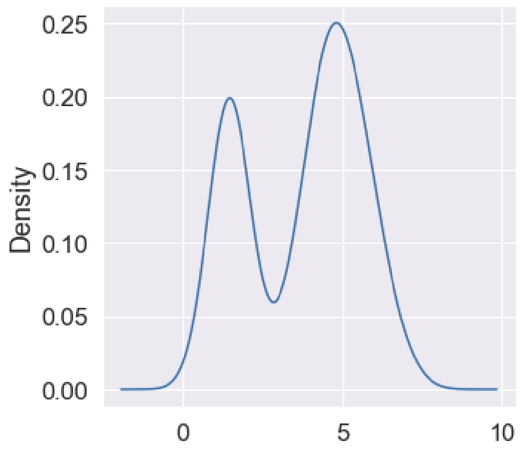
6. Density Plots :

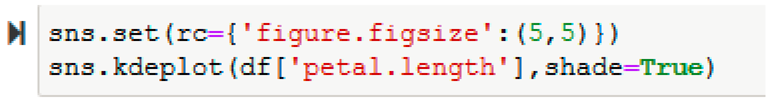
A density plot is like a smoother version of a histogram. Generally, the kernel density estimate is used in density plots to show the probability density function of the variable. A continuous curve, which is the kernel is drawn to generate a smooth density estimation for the whole data.

Plotting density plot of the variable 'petal.length' :

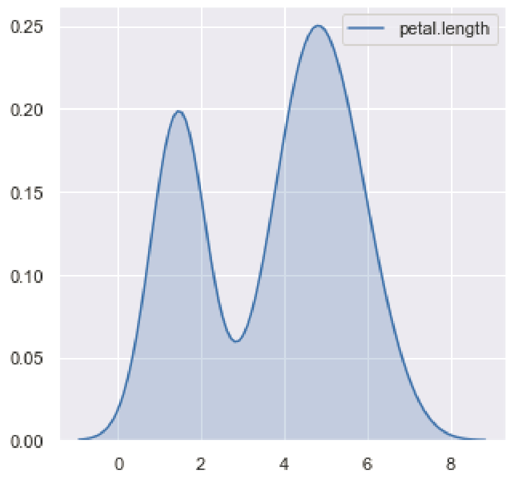


```
plt.figure(figsize=(5,5))
df['petal.length'].plot(kind='density')
```





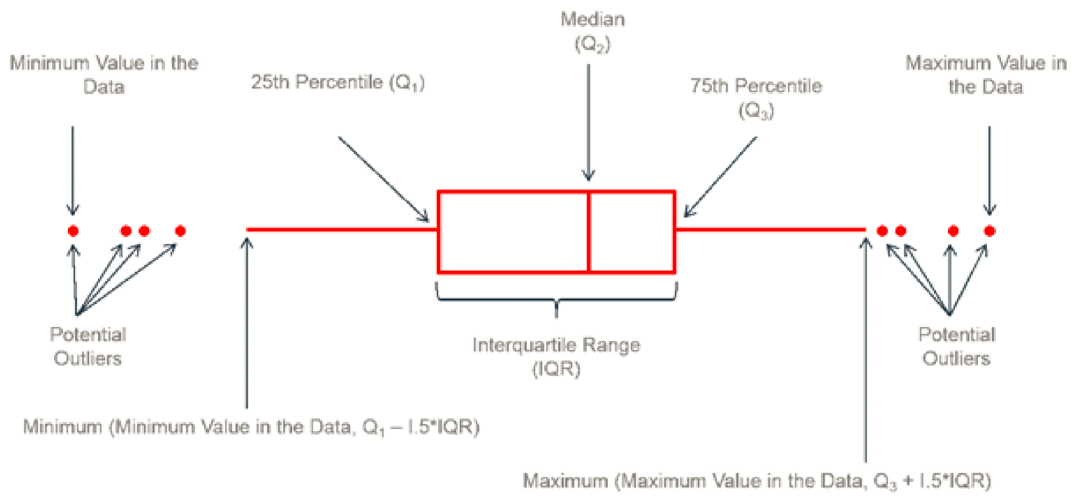
```
sns.set(rc={'figure.figsize':(5,5)})
sns.kdeplot(df['petal.length'],shade=True)
```



we use the pandas *df.plot()* function (built over matplotlib) or the seaborn library's *sns.kdeplot()* function to plot a density plot .

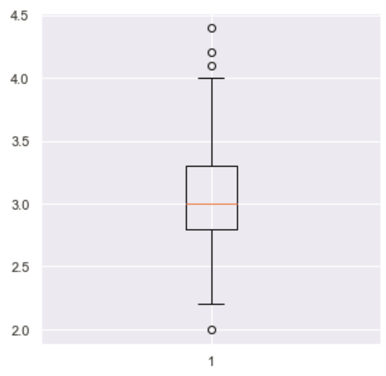
7. Box Plots :

A box-plot is a very useful and standardized way of displaying the distribution of data based on a five-number summary (minimum, first quartile, second quartile(median), third quartile, maximum). It helps in understanding these parameters of the distribution of data and is extremely helpful in detecting outliers.


Median
(Q_2)
Minimum Value in the Data
25th Percentile (Q_1)
75th Percentile (Q_3)
Maximum Value in the Data
Potential Outliers
Interquartile Range (IQR)
Potential Outliers
Minimum (Minimum Value in the Data, Q_1 – 1.5*IQR)
Maximum (Maximum Value in the Data, Q_3 + 1.5*IQR)

Plotting box plot of variable 'sepal.width' :

```
plt.boxplot(df['sepal.width'])
```

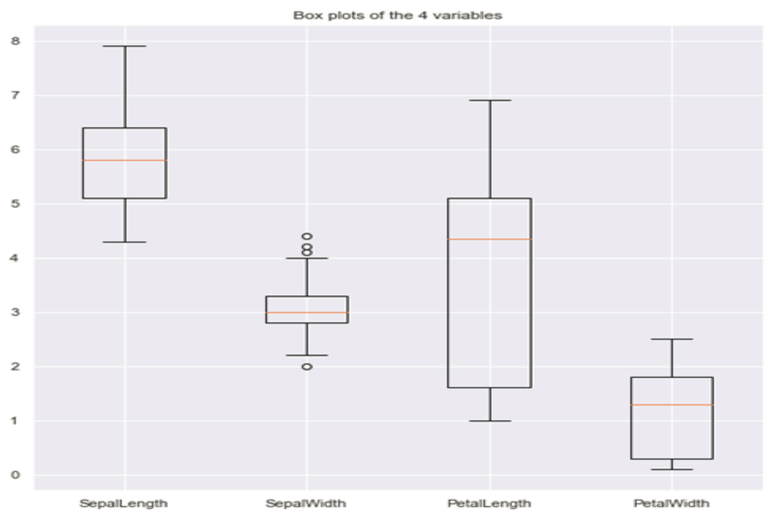


Since the box plot is for continuous variables, firstly create a data frame without the column 'variety'. Then drop the column from the DataFrame using the *drop( )* function and specify axis=1 to indicate it.

Removing the column with categorical variables

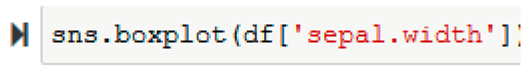
```
dfM=df.drop('variety',axis=1)
```

```
plt.figure(figsize=(9,9))
#Set Title
plt.title('Box plots of the 4 variables')
plt.boxplot(dfM.values,labels=['SepalLength','SepalWidth','PetalLength','PetalWidth'])
```

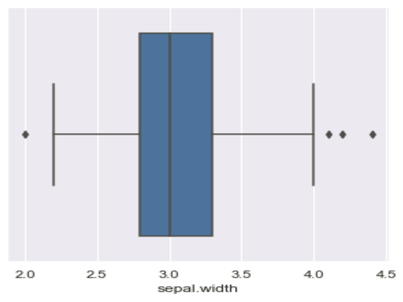


In matplotlib, mention the labels separately to display it in the output.

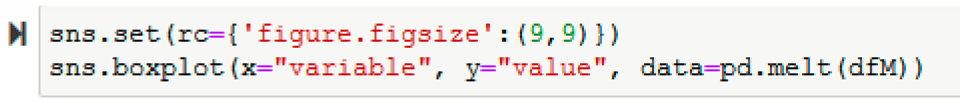
The plotting box plot in seaborn :



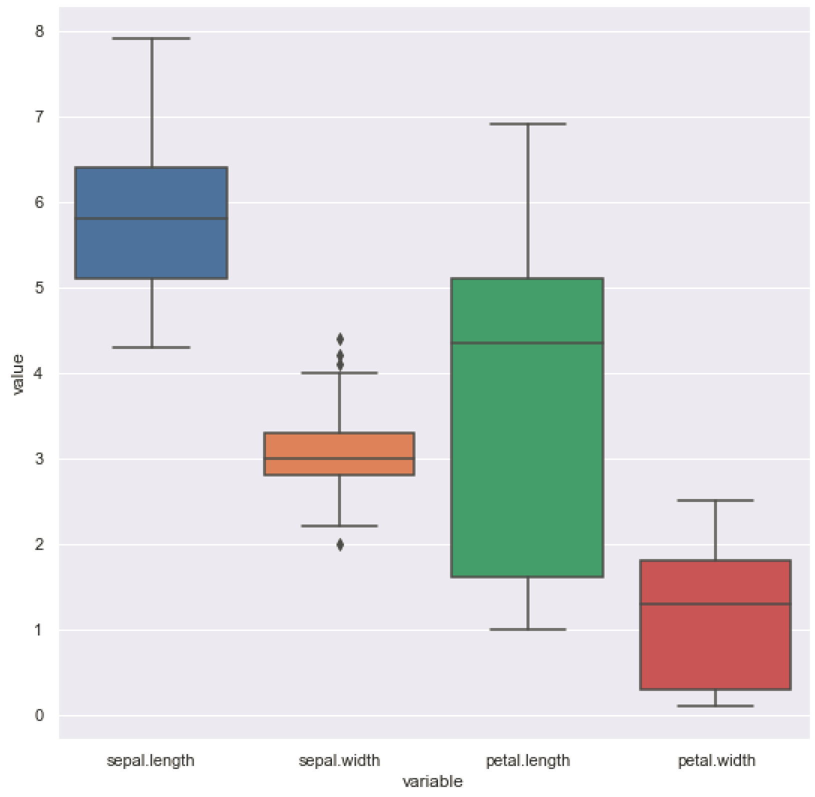
```
sns.boxplot(df['sepal.width']]
```



Plotting the box plots of all variables in one frame :



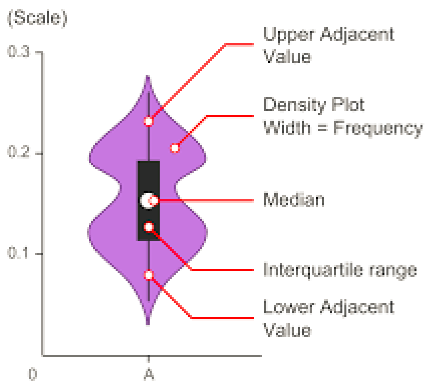
```
sns.set(rc={'figure.figsize':(9,9)})
sns.boxplot(x="variable", y="value", data=pd.melt(dfM))
```



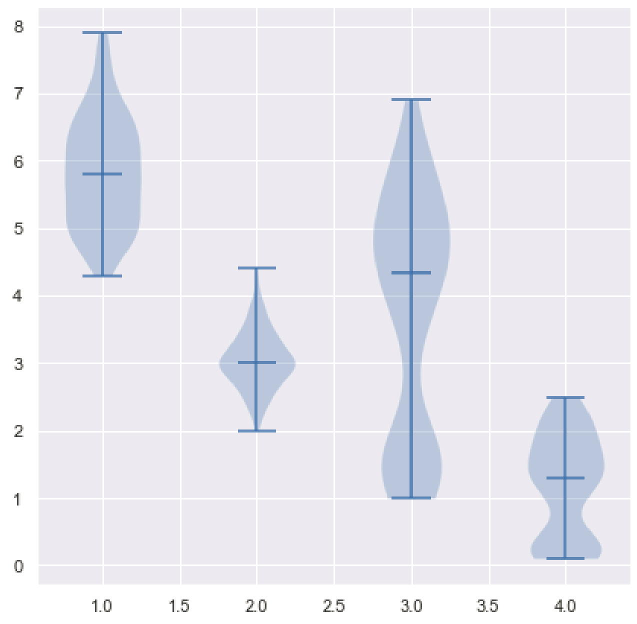
Apply the pandas function *pd.melt()* on the modified data frame which is then passed onto the *sns.boxplot()* function.

8. Violin Plots :

The Violin plot is very much similar to a box plot, with the addition of a rotated kernel density plot on each side. It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared.

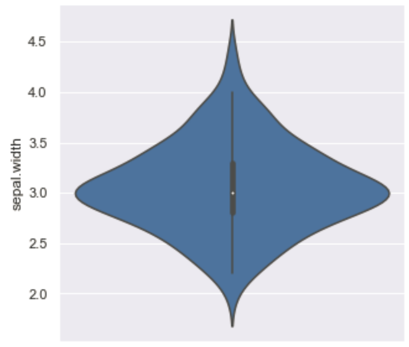


```
plt.figure(figsize=(7,7))
plt.violinplot(dfM.values,showmedians=True)
```

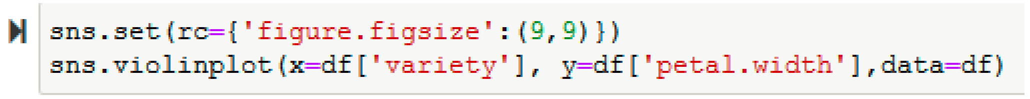


We use *plt.violinplot( )* function. The Boolean parameter 'showmedians' is set to True, due to which the medians are marked for every variable. The violin plot helps to understand the estimated density of the variable.

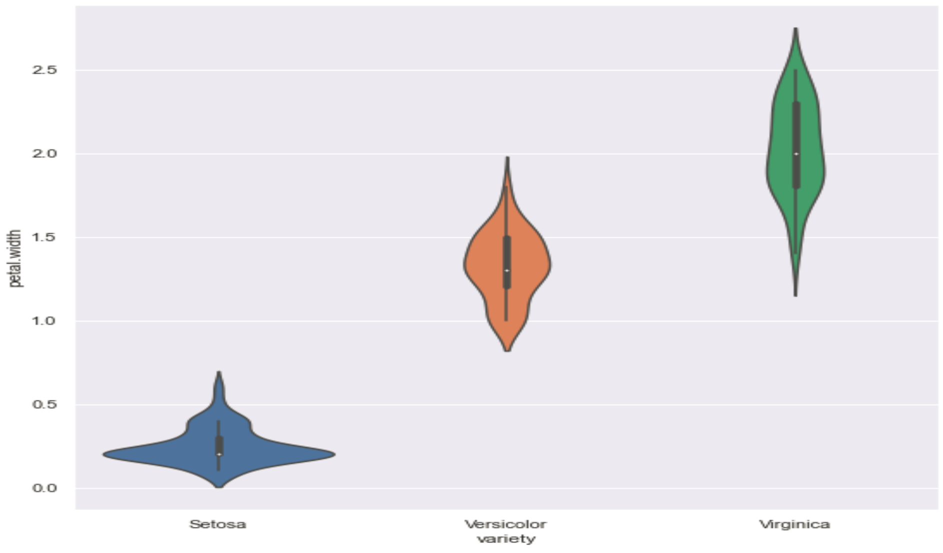
```
sns.set(rc={'figure.figsize':(5,5)})
sns.violinplot(df['sepal.width'],orient='vertical')
```



Comparing the variable 'sepal.width' according to the 'variety' of species mentioned in the dataset



```
sns.set(rc={'figure.figsize':(9,9)})
sns.violinplot(x=df['variety'], y=df['petal.width'],data=df)
```



Practice Exercise

1. **Univariate Analysis** for Continuous Variables and Categorical Variables

2. **Bivariate Analysis** for Continuous Variable vs Continuous Variable, Categorical Variable vs Categorical Variable

https://medium.com/mlearning-ai/univariate-bivariate-and-multivariate-data-analysis-in-python-341493c3d173