

## Experiment No. 10

### Aim: Data Visualization III:

1. Download the Iris flower dataset or any other dataset into a DataFrame. (e.g., <https://archive.ics.uci.edu/ml/datasets/Iris> ).
2. Scan the dataset and give the inference as:
  1. List down the features and their types (e.g., numeric, nominal) available in the dataset.
  2. Create a histogram for each feature in the dataset to illustrate the feature distributions.
  3. Create a boxplot for each feature in the dataset.
  4. Compare distributions and identify outliers.

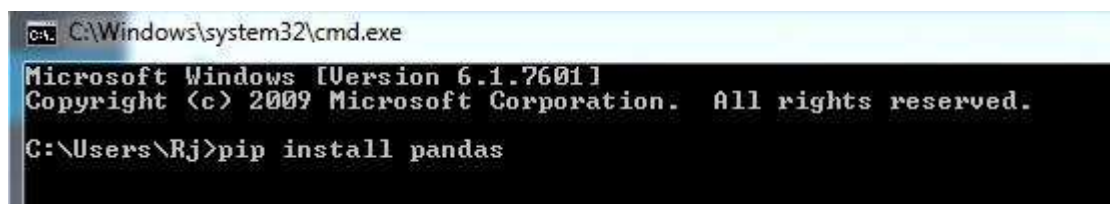
### Theory: Python – Basics of Pandas using Iris Dataset

Python language is one of the most trending programming languages as it is dynamic than others. Python is a simple high-level and an open-source language used for general-purpose programming. It has many open-source libraries and Pandas is one of them. Pandas is a powerful, fast, flexible open-source library used for data analysis and manipulations of data frames/datasets. Pandas can be used to read and write data in a dataset of different formats like CSV(comma separated values), txt, xls(Microsoft Excel) etc. Following are the various features of Pandas in Python and how to use it to practice.

**Prerequisites:** Basic knowledge about coding in Python.

#### Installation:

So if you are new to practice Pandas, then firstly you should install Pandas on your system. Go to Command Prompt and run it as administrator. Make sure you are connected with an internet connection to download and install it on your system. Then type “**pip install pandas**“, then press Enter key.



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Rj>pip install pandas
```

Download the Dataset “**Iris.csv**”.

**Iris dataset** is the Hello World for the Data Science, so if you have started your career in Data Science and Machine Learning you will be practicing basic ML algorithms on this famous dataset. Iris dataset contains five columns such as Petal Length, Petal Width, Sepal Length, Sepal Width and Species Type. Iris is a flowering plant, the researchers have measured various features of the different iris flower and recorded digitally.

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
1	5.1	3.5	1.4	0.2
2	4.9	3	1.4	0.2
3	4.7	3.2	1.3	0.2
4	4.6	3.1	1.5	0.2
5	5	3.6	1.4	0.2
6	5.4	3.9	1.7	0.4
7	4.6	3.4	1.4	0.3
8	5	3.4	1.5	0.2
9	4.4	2.9	1.4	0.2
10	4.9	3.1	1.5	0.1
11	5.4	3.7	1.5	0.2
12	4.8	3.4	1.6	0.2
13	4.8	3	1.4	0.1
14	4.3	3	1.1	0.1
15	5.8	4	1.2	0.2
16	5.7	4.4	1.5	0.4
17	5.4	3.9	1.3	0.4
18	5.1	3.5	1.4	0.3
19	5.7	3.8	1.7	0.3

### Getting Started with Pandas:

**Code: Importing pandas to use in our code as pd.**

- Python3

```
import pandas as pd
```

**Code: Reading the dataset “Iris.csv”.**

- Python3

```
data = pd.read_csv("your downloaded dataset location ")
```

```
import pandas as pd
data=pd.read_csv("C:\\Users\\Rj\\Desktop\\Iris.csv")
```

### Code: Displaying up the top rows of the dataset with their columns

The function `head()` will display the top rows of the dataset, the default value of this function is 5, that is it will show top 5 rows when no argument is given to it.

- Python3

```
data.head()
```

### Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

### Displaying the number of rows randomly.

In `sample()` function, it will also display the rows according to arguments given, but it will display the rows randomly.

- Python3

```
data.sample(10)
```

### Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
108	109	6.7	2.5	5.8	1.8	Iris-virginica
139	140	6.9	3.1	5.4	2.1	Iris-virginica
10	11	5.4	3.7	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
132	133	6.4	2.8	5.6	2.2	Iris-virginica
99	100	5.7	2.8	4.1	1.3	Iris-versicolor
140	141	6.7	3.1	5.6	2.4	Iris-virginica
1	2	4.9	3.0	1.4	0.2	Iris-setosa
107	108	7.3	2.9	6.3	1.8	Iris-virginica
42	43	4.4	3.2	1.3	0.2	Iris-setosa

**Code: Displaying the number of columns and names of the columns.**

The column() function prints all the columns of the dataset in a list form.

- Python3

data.columns

**Output:**

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
      'Species'],
      dtype='object')
```

**Code: Displaying the shape of the dataset.**

The shape of the dataset means to print the total number of rows or entries and the total number of columns or features of that particular dataset.

- Python3

#The first one is the number of rows and

# the other one is the number of columns.

`data.shape`

**Output:**

```
(150, 6)
```

**Code: Display the whole dataset**

- Python3

`print(data)`

**Output:**

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows x 6 columns

### Code: Slicing the rows.

Slicing means if you want to print or work upon a particular group of lines that is from 10th row to 20th row.

- Python3

```
#data[start:end]
```

#start is inclusive whereas end is exclusive

```
print(data[10:21])
```

# it will print the rows from 10 to 20.

# you can also save it in a variable for further use in analysis

```
sliced_data=data[10:21]
```

```
print(sliced_data)
```

### Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
10	11	5.4	3.7	1.5	0.2	Iris-setosa
11	12	4.8	3.4	1.6	0.2	Iris-setosa
12	13	4.8	3.0	1.4	0.1	Iris-setosa
13	14	4.3	3.0	1.1	0.1	Iris-setosa
14	15	5.8	4.0	1.2	0.2	Iris-setosa
15	16	5.7	4.4	1.5	0.4	Iris-setosa
16	17	5.4	3.9	1.3	0.4	Iris-setosa
17	18	5.1	3.5	1.4	0.3	Iris-setosa
18	19	5.7	3.8	1.7	0.3	Iris-setosa
19	20	5.1	3.8	1.5	0.3	Iris-setosa
20	21	5.4	3.4	1.7	0.2	Iris-setosa

### Code: Displaying only specific columns.

In any dataset, it is sometimes needed to work upon only specific features or columns, so we can do this by the following code.

- Python3

```
#here in the case of Iris dataset
```

```
#we will save it in a another variable named "specific_data"
```

```
specific_data=data[["Id","Species"]]
```

```
#data[["column_name1","column_name2","column_name3"]]
```

#now we will print the first 10 columns of the specific\_data dataframe.

```
print(specific_data.head(10))
```

### Output:

	Id	Species
0	1	Iris-setosa
1	2	Iris-setosa
2	3	Iris-setosa
3	4	Iris-setosa
4	5	Iris-setosa
5	6	Iris-setosa
6	7	Iris-setosa
7	8	Iris-setosa
8	9	Iris-setosa
9	10	Iris-setosa

### Filtering:Displaying the specific rows using “iloc” and “loc” functions.

The “loc” functions use the index name of the row to display the particular row of the dataset.

The “iloc” functions use the index integer of the row, which gives complete information about the row.

#### Code:

- Python3

#here we will use iloc

```
data.iloc[5]
```



#it will display records only with species "Iris-setosa".

```
data.loc[data["Species"] == "Iris-setosa"]
```

### Output:

```
Id          6
SepalLengthCm  5.4
SepalWidthCm   3.9
PetalLengthCm  1.7
PetalWidthCm   0.4
Species      Iris-setosa
Name: 5, dtype: object
```

```
iloc()[/caption]
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa
10	11	5.4	3.7	1.5	0.2	Iris-setosa
11	12	4.8	3.4	1.6	0.2	Iris-setosa
12	13	4.8	3.0	1.4	0.1	Iris-setosa
13	14	4.3	3.0	1.1	0.1	Iris-setosa

*loc()*

**Code: Counting the number of counts of unique values using “value\_counts()”.**

The value\_counts() function, counts the number of times a particular instance or data has occurred.

- Python3

#In this dataset we will work on the Species column, it will count number of times a particular species has occurred.

```
data["Species"].value_counts()
```

#it will display in descending order.

**Output:**

```
Iris-versicolor    50  
Iris-setosa        50  
Iris-virginica     50  
Name: Species, dtype: int64
```

**Calculating sum, mean and mode of a particular column.**

We can also calculate the sum, mean and mode of any integer columns as I have done in the following code.

- Python3

```
# data["column_name"].sum()
```

```
sum_data = data["SepalLengthCm"].sum()
```

```
mean_data = data["SepalLengthCm"].mean()
```

```
median_data = data["SepalLengthCm"].median()
```

```
print("Sum:",sum_data, "\nMean:", mean_data, "\nMedian:",median_data)
```

### Output:

```
Sum: 876.5
Mean: 5.843333333333335
Median: 5.8
```

### Code: Extracting minimum and maximum from a column.

Identifying minimum and maximum integer, from a particular column or row can also be done in a dataset.

- Python3

```
min_data=data["SepalLengthCm"].min()
```

```
max_data=data["SepalLengthCm"].max()
```

```
print("Minimum:",min_data, "\nMaximum:", max_data)
```

### Output:

```
Minimum: 4.3
Maximum: 7.9
```

### Code: Adding a column to the dataset.

If want to add a new column in our dataset, as we are doing any calculations or extracting some information from the dataset, and if you want to save it a new column. This can be done by the following code by taking a case where we have added all integer values of all columns.

- Python3

```
# For example, if we want to add a column let say "total_values",  
  
# that means if you want to add all the integer value of that particular  
  
# row and get total answer in the new column "total_values".  
  
# first we will extract the columns which have integer values.
```

```
cols = data.columns
```

```
# it will print the list of column names.
```

```
print(cols)
```

```
# we will take that columns which have integer values.
```

```
cols = cols[1:5]
```

```
# we will save it in the new dataframe variable
```

```
data1 = data[cols]
```

```
# now adding new column "total_values" to dataframe data.
```

```
data["total_values"]=data1[cols].sum(axis=1)
```

# here axis=1 means you are working in rows,

# whereas axis=0 means you are working in columns.

## Output:

```
Index(['Id', 'SepallLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',  
      'Species', 'total_values'],  
      dtype='object')
```

	Id	SepallLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	total_values
0	1	5.1	3.5	1.4	0.2	Iris-setosa	10.2
1	2	4.9	3.0	1.4	0.2	Iris-setosa	9.5
2	3	4.7	3.2	1.3	0.2	Iris-setosa	9.4
3	4	4.6	3.1	1.5	0.2	Iris-setosa	9.4
4	5	5.0	3.6	1.4	0.2	Iris-setosa	10.2
...	...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica	17.2
146	147	6.3	2.5	5.0	1.9	Iris-virginica	15.7
147	148	6.5	3.0	5.2	2.0	Iris-virginica	16.7
148	149	6.2	3.4	5.4	2.3	Iris-virginica	17.3
149	150	5.9	3.0	5.1	1.8	Iris-virginica	15.8

150 rows x 7 columns

### Code: Renaming the columns.

Renaming our column names can also be possible in python pandas libraries. We have used the `rename()` function, where we have created a dictionary “newcols” to update our new column names. The following code illustrates that.

- Python3

```

newcols={

    "Id":"id",

    "SepalLengthCm":"sepalength"

    "SepalWidthCm":"sepalwidth"}

data.rename(columns=newcols,inplace=True)

print(data.head())

```

## Output:

	id	sepalength	sepalwidth	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

## Formatting and Styling:

Conditional formatting can be applied to your dataframe by using Dataframe.style function. Styling is used to visualize your data, and most convenient way of visualizing your dataset is in tabular form.

Here we will highlight the minimum and maximum from each row and columns.

- Python3

#this is an example of rendering a datagram,

which is not visualised by any styles.

data.style

### Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.100000	3.500000	1.400000	0.200000	Iris-setosa
1	2	4.900000	3.000000	1.400000	0.200000	Iris-setosa
2	3	4.700000	3.200000	1.300000	0.200000	Iris-setosa
3	4	4.600000	3.100000	1.500000	0.200000	Iris-setosa
4	5	5.000000	3.600000	1.400000	0.200000	Iris-setosa

Now we will highlight the maximum and minimum column-wise, row-wise, and the whole dataframe wise using `Styler.apply` function. The `Styler.apply` function passes each column or row of the dataframe depending upon the keyword argument `axis`. For column-wise use `axis=0`, row-wise use `axis=1`, and for the entire table at once use `axis=None`.

- Python3

# we will here print only the top 10 rows of the dataset,

# if you want to see the result of the whole dataset remove

`#.head(10)` from the below code

```
data.head(10).style.highlight_max(color='lightgreen', axis=0)
```

```
data.head(10).style.highlight_max(color='lightgreen', axis=1)
```

```
data.head(10).style.highlight_max(color='lightgreen', axis=None)
```

Output:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.100000	3.500000	1.400000	0.200000	Iris-setosa
1	2	4.900000	3.000000	1.400000	0.200000	Iris-setosa
2	3	4.700000	3.200000	1.300000	0.200000	Iris-setosa
3	4	4.600000	3.100000	1.500000	0.200000	Iris-setosa
4	5	5.000000	3.600000	1.400000	0.200000	Iris-setosa
5	6	5.400000	3.900000	1.700000	0.400000	Iris-setosa
6	7	4.600000	3.400000	1.400000	0.300000	Iris-setosa
7	8	5.000000	3.400000	1.500000	0.200000	Iris-setosa
8	9	4.400000	2.900000	1.400000	0.200000	Iris-setosa
9	10	4.900000	3.100000	1.500000	0.100000	Iris-setosa

*for axis=0*

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.100000	3.500000	1.400000	0.200000	Iris-setosa
1	2	4.900000	3.000000	1.400000	0.200000	Iris-setosa
2	3	4.700000	3.200000	1.300000	0.200000	Iris-setosa
3	4	4.600000	3.100000	1.500000	0.200000	Iris-setosa
4	5	5.000000	3.600000	1.400000	0.200000	Iris-setosa
5	6	5.400000	3.900000	1.700000	0.400000	Iris-setosa
6	7	4.600000	3.400000	1.400000	0.300000	Iris-setosa
7	8	5.000000	3.400000	1.500000	0.200000	Iris-setosa
8	9	4.400000	2.900000	1.400000	0.200000	Iris-setosa
9	10	4.900000	3.100000	1.500000	0.100000	Iris-setosa

*for axis=1*



	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.100000	3.500000	1.400000	0.200000	Iris-setosa
1	2	4.900000	3.000000	1.400000	0.200000	Iris-setosa
2	3	4.700000	3.200000	1.300000	0.200000	Iris-setosa
3	4	4.600000	3.100000	1.500000	0.200000	Iris-setosa
4	5	5.000000	3.600000	1.400000	0.200000	Iris-setosa
5	6	5.400000	3.900000	1.700000	0.400000	Iris-setosa
6	7	4.600000	3.400000	1.400000	0.300000	Iris-setosa
7	8	5.000000	3.400000	1.500000	0.200000	Iris-setosa
8	9	4.400000	2.900000	1.400000	0.200000	Iris-setosa
9	10	4.900000	3.100000	1.500000	0.100000	Iris-setosa

*for axis=None*

### Code: Cleaning and detecting missing values

In this dataset, we will now try to find the missing values i.e NaN, which can occur due to several reasons.

- Python3

```
data.isnull()
```

#if there is data is missing, it will display True else False.

**Output:**

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
145	False	False	False	False	False	False
146	False	False	False	False	False	False
147	False	False	False	False	False	False
148	False	False	False	False	False	False
149	False	False	False	False	False	False

150 rows x 6 columns

*isnull()*

### Code: Summarizing the missing values.

We will display how many missing values are present in each column.

- Python3

```
data.isnull.sum()
```

**Output:**

```
Id          0
SepalLengthCm  0
SepalWidthCm  0
PetalLengthCm  0
PetalWidthCm  0
Species      0
dtype: int64
```

### Heatmap: Importing seaborn

The heatmap is a data visualisation technique which is used to analyse the dataset as colors in two dimensions. Basically it shows correlation between all numerical variables in the dataset. Heatmap is an attribute of the Seaborn library.

**Code:**

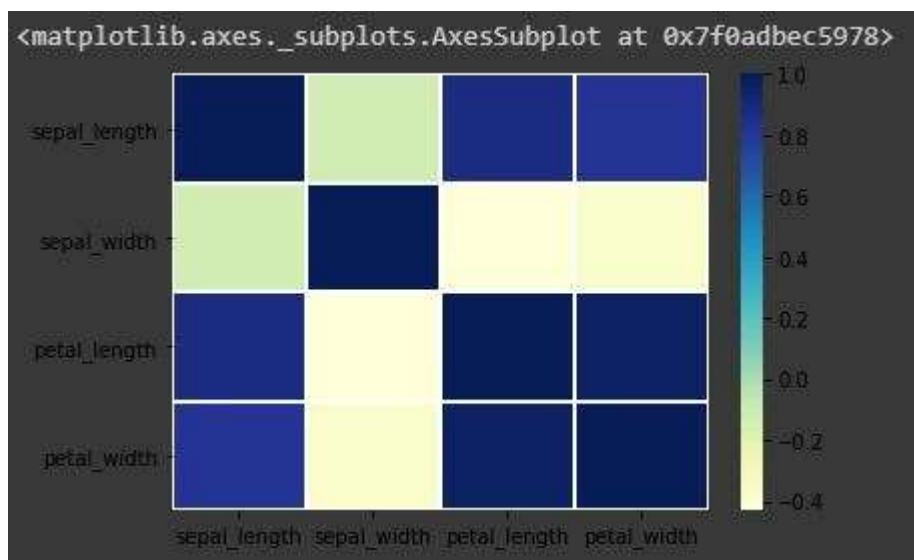
- Python3

```
import seaborn as sns
```

```
iris = sns.load_dataset("iris")
```

```
sns.heatmap(iris.corr(),cmap = "YlGnBu", linecolor = 'white', linewidths = 1)
```

**Output:**

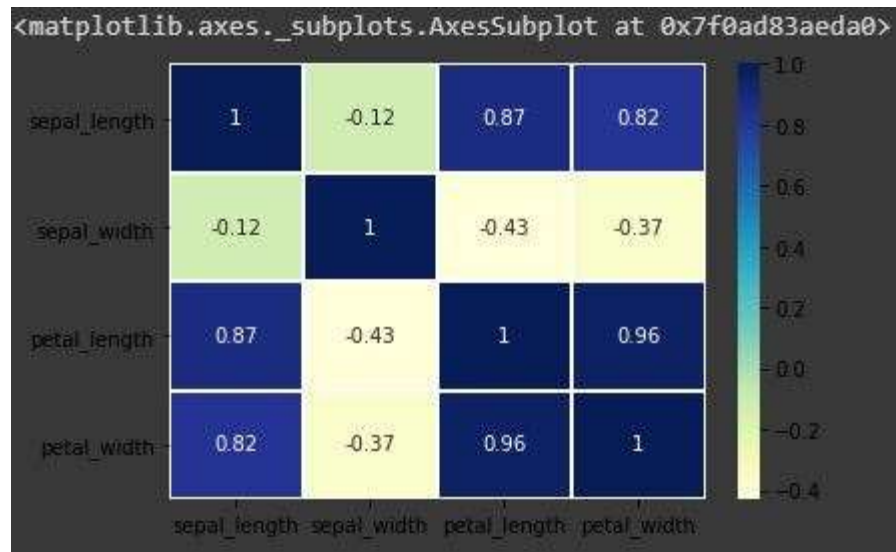


**Code: Annotate each cell with the numeric value using integer formatting**

- Python3

```
sns.heatmap(iris.corr(),cmap = "YlGnBu", linecolor = 'white', linewidths = 1, annot = True )
```

**Output:**



*heatmap with annot=True*

### **Pandas Dataframe Correlation:**

Pandas correlation is used to determine pairwise correlation of all the columns of the dataset. In `dataframe.corr()`, the missing values are excluded and non-numeric columns are also ignored.

**Code:**

- Python3

```
data.corr(method='pearson')
```

**Output:**

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Id	1.000000	0.716676	-0.397729	0.882747	0.899759
SepalLengthCm	0.716676	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.397729	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.882747	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.899759	0.817954	-0.356544	0.962757	1.000000

`data.corr()`

The output dataframe can be seen as for any cell, row variable correlation with the column variable is the value of the cell. The correlation of a variable with itself is 1. For that reason, all the diagonal values are 1.00.

### **Multivariate Analysis:**

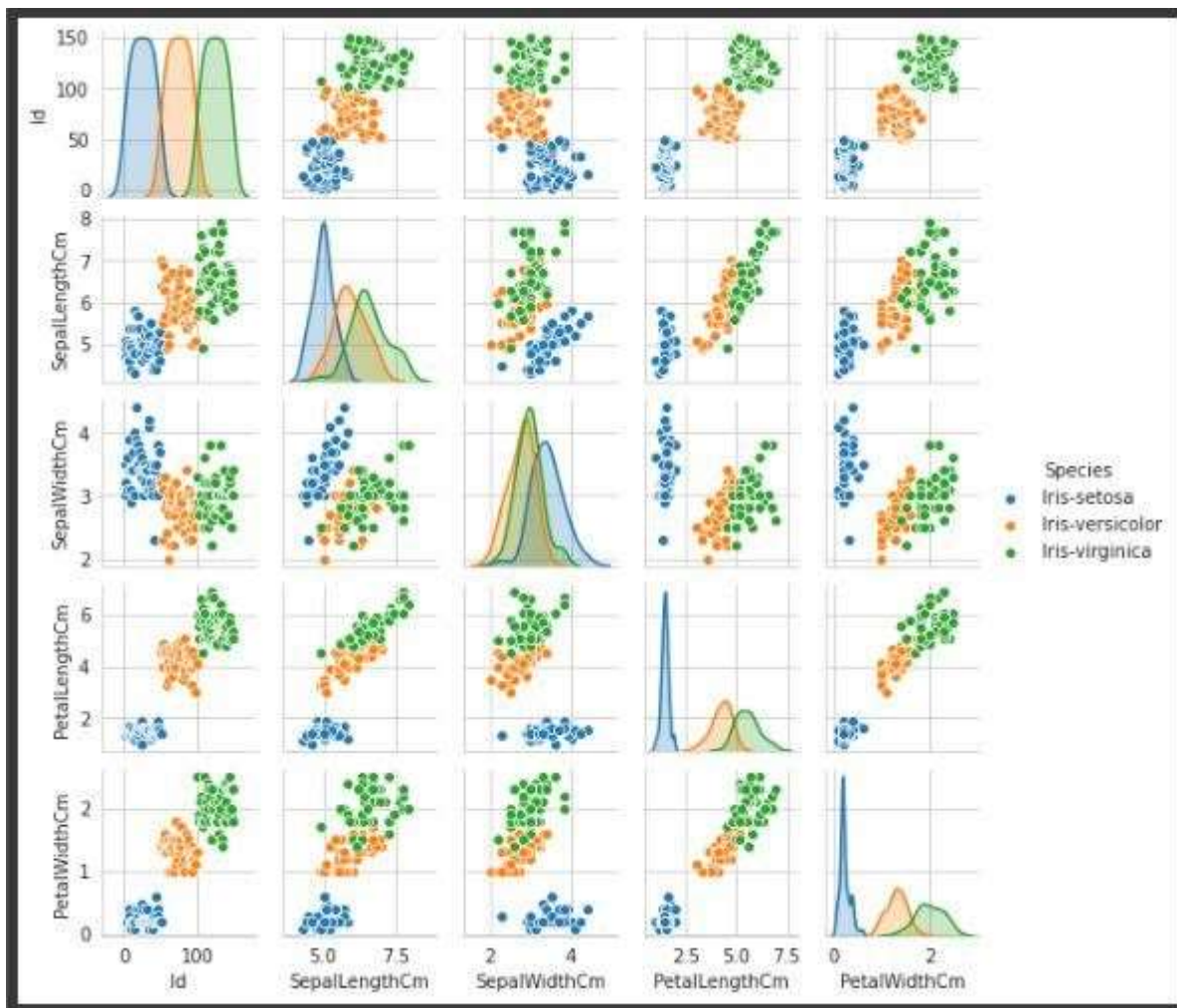
Pair plot is used to visualize the relationship between each type of column variable. It is implemented only by one line code, which is as follows :

### **Code:**

- Python3

```
g = sns.pairplot(data,hue="Species")
```

### **Output:**



*Pairplot of variable “Species”, to make it more understandable.*