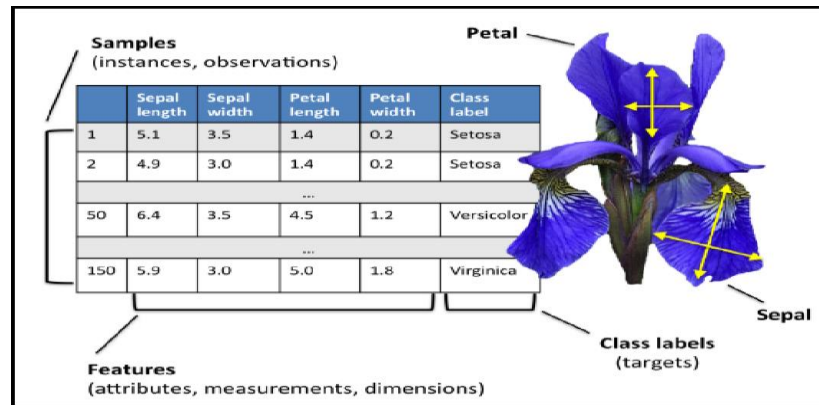


## Description of Dataset-



### 4. Panda Dataframe functions for Load Dataset

# The columns of the resulting DataFrame have different dtypes.

**iris.dtypes**

1. The dataset is downloaded from UCI repository.

**csv\_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'**

2. Now Read CSV File as a DataFrame in Python from from path where you saved the same  
The Iris data set is stored in .csv format. '.csv' stands for comma separated values. It is easier to load .csv files in Pandas data frame and perform various analytical operations on it.

Load Iris.csv into a Pandas data frame —

**Syntax-**

**iris = pd.read\_csv(csv\_url, header = None)**

3. The csv file at the UCI repository does not contain the variable/column names. They are located in a separate file.

**col\_names = ['Sepal\_Length', 'Sepal\_Width', 'Petal\_Length', 'Petal\_Width', 'Species']**

4. read in the dataset from the UCI Machine Learning Repository link and specify column names to use

**iris = pd.read\_csv(csv\_url, names = col\_names)**

	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. Panda Dataframe functions for Data Preprocessing :

### Dataframe Operations:

Sr. No	Data Frame Function	Description
1	<code>dataset.head(n=5)</code>	Return the first n rows.
2	<code>dataset.tail(n=5)</code>	Return the last n rows.
3	<code>dataset.index</code>	The index (row labels) of the Dataset.
4	<code>dataset.columns</code>	The column labels of the Dataset.
5	<code>dataset.shape</code>	Return a tuple representing the dimensionality of the Dataset.
6	<code>dataset.dtypes</code>	Return the dtypes in the Dataset. This returns a Series with the data type of each column. The result's index is the original Dataset's columns. Columns with mixed types are stored with the object dtype.
7	<code>dataset.columns.values</code>	Return the columns values in the Dataset in arrayformat
8	<code>dataset.describe(include='all')</code>	Generate descriptive statistics.to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values.Analyzes both numeric and object series, as well as Dataset column sets of mixed data types.
9	<code>dataset['Column name']</code>	Read the Data Column wise.
10	<code>dataset.sort_index(axis=1, ascending=False)</code>	Sort object by labels (along an axis).
11	<code>dataset.sort_values(by="Column name")</code>	Sort values by column name.
12	<code>dataset.iloc[5]</code>	Purely integer-location based indexing for selection by position.
13	<code>dataset[0:3]</code>	Selecting via [], which slices the rows.

14	<code>dataset.loc[:, ["Col_name1", "col_name2"]]</code>	Selection by label
15	<code>dataset.iloc[:n, :]</code>	a subset of the first n rows of the original data
16	<code>dataset.iloc[:, :n]</code>	a subset of the first n columns of the original data
17	<code>dataset.iloc[:m, :n]</code>	a subset of the first m rows and the first n columns

### Few Examples of iLoc to slice data for iris Dataset

Sr. No	Data Frame Function	Description	Output																		
1	dataset.iloc[3:5, 0:2]	Slice the data	<table><tr><th colspan="2">Id</th><th>SepalLengthCm</th></tr><tr><td>3</td><td>4</td><td>4.6</td></tr><tr><td>4</td><td>5</td><td>5.0</td></tr></table>	Id		SepalLengthCm	3	4	4.6	4	5	5.0									
			Id		SepalLengthCm																
			3	4	4.6																
4	5	5.0																			
2	dataset.iloc[[1, 2, 4], [0, 2]]	By lists of integer position locations, similar to the NumPy/Python style:	<table><tr><th colspan="2">Id</th><th>SepalWidthCm</th></tr><tr><td>1</td><td>2</td><td>3.0</td></tr><tr><td>2</td><td>3</td><td>3.2</td></tr><tr><td>4</td><td>5</td><td>3.6</td></tr></table>	Id		SepalWidthCm	1	2	3.0	2	3	3.2	4	5	3.6						
			Id		SepalWidthCm																
			1	2	3.0																
2	3	3.2																			
4	5	3.6																			
3	dataset.iloc[1:3, :]	For slicing rows explicitly:	<table><tr><th>Id</th><th>SepalLengthCm</th><th>SepalWidthCm</th><th>PetalLengthCm</th><th>PetalWidthCm</th><th>Species</th></tr><tr><td>1</td><td>2</td><td>4.9</td><td>3.0</td><td>1.4</td><td>0.2 Iris-setosa</td></tr><tr><td>2</td><td>3</td><td>4.7</td><td>3.2</td><td>1.3</td><td>0.2 Iris-setosa</td></tr></table>	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	1	2	4.9	3.0	1.4	0.2 Iris-setosa	2	3	4.7	3.2	1.3	0.2 Iris-setosa
Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species																
1	2	4.9	3.0	1.4	0.2 Iris-setosa																
2	3	4.7	3.2	1.3	0.2 Iris-setosa																
4	dataset.iloc[:, 1:3]	For slicing Column explicitly:	<table><tr><th colspan="2">SepalLengthCm</th><th>SepalWidthCm</th></tr><tr><td>0</td><td>5.1</td><td>3.5</td></tr><tr><td>1</td><td>4.9</td><td>3.0</td></tr><tr><td>2</td><td>4.7</td><td>3.2</td></tr><tr><td>3</td><td>4.6</td><td>3.1</td></tr></table>	SepalLengthCm		SepalWidthCm	0	5.1	3.5	1	4.9	3.0	2	4.7	3.2	3	4.6	3.1			
SepalLengthCm		SepalWidthCm																			
0	5.1	3.5																			
1	4.9	3.0																			
2	4.7	3.2																			
3	4.6	3.1																			
4	dataset.iloc[1, 1]	For getting a value explicitly:	4.9																		
5	dataset['SepalLengthCm'].iloc[5]	Accessing Column and Rows by position	5.4																		

6	<code>cols_2_4=dataset.columns[2:4]</code> <code>dataset[cols_2_4]</code>	Get Column Name then get data from column	<table><tr><th></th><th>SepalWidthCm</th><th>PetalLengthCm</th></tr><tr><td>0</td><td>3.5</td><td>1.4</td></tr><tr><td>1</td><td>3.0</td><td>1.4</td></tr><tr><td>2</td><td>3.2</td><td>1.3</td></tr><tr><td>3</td><td>3.1</td><td>1.5</td></tr></table>		SepalWidthCm	PetalLengthCm	0	3.5	1.4	1	3.0	1.4	2	3.2	1.3	3	3.1	1.5			
	SepalWidthCm	PetalLengthCm																			
0	3.5	1.4																			
1	3.0	1.4																			
2	3.2	1.3																			
3	3.1	1.5																			
7	<code>dataset[dataset.columns[2:4]].iloc[5:10]</code>	in one Expression answer for the above two commands	<table><tr><th></th><th>SepalWidthCm</th><th>PetalLengthCm</th></tr><tr><td>5</td><td>3.9</td><td>1.7</td></tr><tr><td>6</td><td>3.4</td><td>1.4</td></tr><tr><td>7</td><td>3.4</td><td>1.5</td></tr><tr><td>8</td><td>2.9</td><td>1.4</td></tr><tr><td>9</td><td>3.1</td><td>1.5</td></tr></table>		SepalWidthCm	PetalLengthCm	5	3.9	1.7	6	3.4	1.4	7	3.4	1.5	8	2.9	1.4	9	3.1	1.5
	SepalWidthCm	PetalLengthCm																			
5	3.9	1.7																			
6	3.4	1.4																			
7	3.4	1.5																			
8	2.9	1.4																			
9	3.1	1.5																			

### Checking of Missing Values in Dataset:

- `isnull()` is the function that is used to check missing values or null values in pandas python.
- `isna()` function is also used to get the count of missing values of column and row wise count of missing values
- The dataset considered for explanation is:

	Name	State	Gender	Score
0	George	Arizona	M	63.0
1	Andrea	Georgia	F	48.0
2	micheal	Newyork	M	56.0
3	maggie	Indiana	F	75.0
4	Ravi	Florida	M	NaN
5	Xien	California	M	77.0
6	Jalpa	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN

- a. is there any missing values in dataframe as a whole

Function: `DataFrame.isnull()`

Output:

	Name	State	Gender	Score
0	False	False	False	False
1	False	False	False	False
2	False	False	False	False
3	False	False	False	False
4	False	False	False	True
5	False	False	False	False
6	False	True	True	True
7	True	True	True	True

- b. is there any missing values across each column

Function: `DataFrame.isnull().any()`

**Output:**

```
Name      True
State      True
Gender     True
Score      True
dtype: bool
```

**c. count of missing values across each column using isna() and isnull()**

In order to get the count of missing values of the entire dataframe isnull() function is used. sum() which does the column wise sum first and doing another sum() will get the count of missing values of the entire dataframe.

**Function:** dataframe.isnull().sum().sum()

**Output :** 8

**d. count row wise missing value using isnull()**

**Function:** dataframe.isnull().sum(axis = 1)

**Output:**

```
0      0
1      0
2      0
3      0
4      1
5      0
6      3
7      4
dtype: int64
```

**e. count Column wise missing value using isnull()**

**Method 1:**

**Function:** dataframe.isnull().sum()

```
Name      1
State      2
Gender     2
Score      3
dtype: int64
```

**Output:**

**Method 2:**

**unction:** dataframe.isna().sum()

```
Name      1
State      2
Gender     2
Score      3
dtype: int64
```

**f. count of missing values of a specific column.**

**Function:** `dataframe.col_name.isnull().sum()`

```
df1.Gender.isnull().sum()
```

**Output: 2**

**g. groupby count of missing values of a column.**

In order to get the count of missing values of the particular column by group in pandas we will be using `isnull()` and `sum()` function with `apply()` and `groupby()` which performs the group wise count of missing values as shown below.

**Function:**

```
df1.groupby(['Gender'])['Score'].apply(lambda x:
x.isnull().sum())
```

**Output:**

```
Gender
F      0
M      1
Name: Score, dtype: int64
```

## 6. Panda functions for Data Formatting and Normalization

The Transforming data stage is about converting the data set into a format that can be analyzed or modelled effectively, and there are several techniques for this process.

- a. Data Formatting:** Ensuring all data formats are correct (e.g. object, text, floating number, integer, etc.) is another part of this initial ‘cleaning’ process. If you are working with dates in Pandas, they also need to be stored in the exact format to use special date-time functions.

Functions used for data formatting

Sr. No	Data Frame Function	Description	Output
1.	<b>df.dtypes</b>	To check the data type	<pre>df.dtypes sepal length (cm)    float64 sepal width (cm)     float64 petal length (cm)    float64 petal width (cm)     float64 dtype: object</pre>

2.	<b>df['petal length (cm)']= df['petal length (cm)'].astype('int')</b>	To change the data type (data type of 'petal length (cm)'changed to int)	<pre>df.dtypes</pre> <pre> sepal length (cm)    float64 sepal width (cm)     float64 petal length (cm)      int64 petal width (cm)     float64 dtype: object </pre>
----	---	--	---

**b. Data normalization:** Mapping all the nominal data values onto a uniform scale (e.g. from 0 to 1) is involved in data normalization. Making the ranges consistent across variables helps with statistical analysis and ensures better comparisons later on. It is also known as Min-Max scaling.

#### Algorithm:

**Step 1 :** Import pandas and sklearn library for preprocessing

```
from sklearn import preprocessing
```

**Step 2:** Load the iris dataset in dataframe object df

**Step 3:** Print iris dataset.

```
df.head()
```

**Step 5:** Create a minimum and maximum processor object

```
min_max_scaler = preprocessing.MinMaxScaler()
```

**Step 6:** Separate the feature from the class label

```
x=df.iloc[:, :4]
```

**Step 6:** Create an object to transform the data to fit minmax processor

```
x_scaled = min_max_scaler.fit_transform(x)
```

**Step 7:** Run the normalizer on the dataframe

```
df_normalized = pd.DataFrame(x_scaled)
```

**Step 8:** View the dataframe

```
df_normalized
```

**Output: After Step 3:**

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Output after step 8:

	0	1	2	3
0	0.222222	0.625000	0.067797	0.041667
1	0.166667	0.416667	0.067797	0.041667
2	0.111111	0.500000	0.050847	0.041667
3	0.083333	0.458333	0.084746	0.041667
4	0.194444	0.666667	0.067797	0.041667

## 7. Panda Functions for handling categorical variables

- **Categorical variables** have values that **describe a ‘quality’ or ‘characteristic’** of a data unit, like **‘what type’ or ‘which category’**.
- Categorical variables fall into **mutually exclusive (in one category or in another)** and **exhaustive (include all possible options)** categories. Therefore, categorical variables are qualitative variables and **tend to be represented by a non-numeric value.**
- Categorical features refer **to string type data** and can be easily understood by human beings. But in case of a **machine, it cannot interpret the categorical data directly.** Therefore, the categorical data must be **translated into numerical data that can be understood by machine.**

There are many ways to convert categorical data into numerical data. Here the three most used methods are discussed.

**a. Label Encoding:** Label Encoding refers to **converting the labels into a numeric form** so as to convert them into the machine-readable form. **It is an important preprocessing step for the structured dataset** in supervised learning.

**Example :** Suppose we have a column Height in some dataset. After applying label encoding, the Height column is converted into:

Height
Tall
Medium
Short



Height
0
1
2

where 0 is the label for tall, 1 is the label for medium, and 2 is a label for short height.

**Label Encoding on iris dataset:** For iris dataset the target column which is Species. It contains three species Iris-setosa, Iris-versicolor, Iris-virginica.

#### Sklearn Functions for Label Encoding:

- **preprocessing.LabelEncoder** : It Encode labels with value between 0 and n\_classes-1.
- **fit\_transform(y)** :  
**Parameters:** yarray-like of shape (n\_samples,) **Target values.**  
**Returns:** yarray-like of shape (n\_samples,) **Encoded labels.**

This transformer should be used to encode target values, and not the input.

#### Algorithm:

**Step 1 :** Import pandas and sklearn library for preprocessing

```
from sklearn import preprocessing
```

**Step 2:** Load the iris dataset in dataframe object df

**Step 3:** Observe the unique values for the Species column.

```
df['Species'].unique()
```

```
output:      array(['Iris-setosa',      'Iris-versicolor',  
                  'Iris-virginica'], dtype=object)
```

**Step 4:** define label\_encoder object knows how to understand word labels.

```
label_encoder = preprocessing.LabelEncoder()
```

**Step 5:** Encode labels in column 'species'.

```
df['Species']= label_encoder.fit_transform(df['Species'])
```

**Step 6:** Observe the unique values for the Species column.

```
df['Species'].unique()
```

```
Output: array([0, 1, 2], dtype=int64)
```

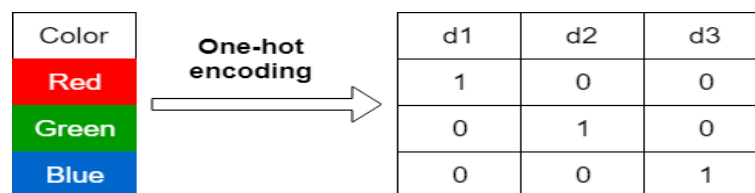
- Use LabelEncoder when there are only two possible values of a categorical feature.

For example, features having value such as yes or no. Or, maybe, gender features when there are only two possible values including male or female.

**Limitation:** Label encoding converts the data in machine-readable form, but it assigns a **unique number(starting from 0) to each class of data**. This may lead to the generation of **priority issues in the data sets**. A label with a high value may be considered to have high priority than a label having a lower value.

#### b. One-Hot Encoding:

In one-hot encoding, we create a new set of dummy (binary) variables that is equal to the number of categories (k) in the variable. For example, let's say we have a categorical variable Color with three categories called "Red", "Green" and "Blue", we need to use three dummy variables to encode this variable using one-hot encoding. A dummy (binary) variable just takes the value 0 or 1 to indicate the exclusion or inclusion of a category.



In one-hot encoding,

"Red" color is encoded as **[1 0 0]** vector of size 3.

"Green" color is encoded as **[0 1 0]** vector of size 3.

"Blue" color is encoded as **[0 0 1]** vector of size 3.

**One-hot encoding on iris dataset:** For iris dataset the target column which is Species. It contains three species Iris-setosa, Iris-versicolor, Iris-virginica.

#### Sklearn Functions for One-hot Encoding:

- **sklearn.preprocessing.OneHotEncoder()** : Encode categorical integer features using a one-hot aka one-of-K scheme

#### Algorithm:

**Step 1 :** Import pandas and sklearn library for preprocessing

```
from sklearn import preprocessing
```

**Step 2:** Load the iris dataset in dataframe object df

**Step 3:** Observe the unique values for the Species column.

```
df['Species'].unique()

output:      array(['Iris-setosa',      'Iris-versicolor',
                  'Iris-virginica'], dtype=object)
```

**Step 4:** Apply label\_encoder object for label encoding the Observe the unique values for the Species column.

```
df['Species'].unique()

Output: array([0, 1, 2], dtype=int64)
```

**Step 5:** Remove the target variable from dataset

```
features_df=df.drop(columns=['Species'])
```

**Step 6:** Apply one\_hot encoder for Species column.

```
enc = preprocessing.OneHotEncoder()
enc_df=pd.DataFrame(enc.fit_transform(df[['Species']]).toarray())
```

**Step 7:** Join the encoded values with Features variable

```
df_encode = features_df.join(enc_df)
```

**Step 8:** Observe the merge dataframe

```
df_encode
```

**Step 9:** Rename the newly encoded columns.

```
df_encode.rename(columns = {0:'Iris-Setosa',
                             1:'Iris-Versicolor',2:'Iris-virginica'}, inplace = True)
```

**Step 10:** Observe the merge dataframe

```
df_encode
```

**Output after Step 8:**

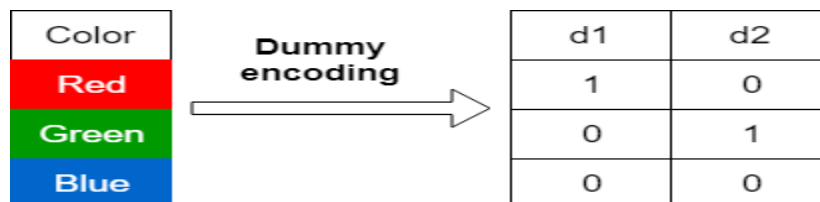
	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	0	1	2
0	5.1	3.5	1.4	0.2	1.0	0.0	0.0
1	4.9	3.0	1.4	0.2	1.0	0.0	0.0
2	4.7	3.2	1.3	0.2	1.0	0.0	0.0
3	4.6	3.1	1.5	0.2	1.0	0.0	0.0
4	5.0	3.6	1.4	0.2	1.0	0.0	0.0

## Output after Step 10:

	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Iris-Setosa	Iris-Versicolor	Iris-virginica
0	5.1	3.5	1.4	0.2	1.0	0.0	0.0
1	4.9	3.0	1.4	0.2	1.0	0.0	0.0
2	4.7	3.2	1.3	0.2	1.0	0.0	0.0
3	4.6	3.1	1.5	0.2	1.0	0.0	0.0
4	5.0	3.6	1.4	0.2	1.0	0.0	0.0

### c. Dummy Variable Encoding

Dummy encoding also uses dummy (binary) variables. Instead of creating a number of dummy variables that is equal to the number of categories (k) in the variable, dummy encoding uses k-1 dummy variables. To encode the same Color variable with three categories using the dummy encoding, we need to use only two dummy variables.



In dummy encoding,

“Red” color is encoded as **[1 0]** vector of size 2.

“Green” color is encoded as **[0 1]** vector of size 2.

“Blue” color is encoded as **[0 0]** vector of size 2.

Dummy encoding removes a duplicate category present in the one-hot encoding.

#### Pandas Functions for One-hot Encoding with dummy variables:

- `pandas.get_dummies(data, prefix=None, prefix_sep='_', dummy_na=False, columns=None, sparse=False, drop_first=False, dtype=None)`: Convert categorical variable into dummy/indicator variables.
- **Parameters:**
  - data**: array-like, Series, or DataFrame  
Data of which to get dummy indicators.
  - prefixstr**: list of str, or dict of str, default None  
String to append DataFrame column names.

**prefix\_sep:** str, default '\_'

If appending prefix, separator/delimiter to use. Or pass a list or dictionary as with prefix.

**dummy\_nabool:** default False

Add a column to indicate NaNs, if False NaNs are ignored.

**columns:** list:like, default None

Column names in the DataFrame to be encoded. If columns is None then all the columns with object or category dtype will be converted.

**sparse: bool: default False**

Whether the dummy-encoded columns should be backed by a SparseArray (True) or a regular NumPy array (False).

**drop\_first:bool, default False**

Whether to get k-1 dummies out of k categorical levels by removing the first level.

**dtype:** dtype, default np.uint8

Data type for new columns. Only a single dtype is allowed.

- **Return :** DataFrame with Dummy-coded data.

### Algorithm:

**Step 1 :** Import pandas and sklearn library for preprocessing

```
from sklearn import preprocessing
```

**Step 2:** Load the iris dataset in dataframe object df

**Step 3:** Observe the unique values for the Species column.

```
df['Species'].unique()
```

```
output:      array(['Iris-setosa',      'Iris-versicolor',  
                  'Iris-virginica'], dtype=object)
```

**Step 4:** Apply label\_encoder object for label encoding the Observe the unique values for the Species column.

```
df['Species'].unique()
```

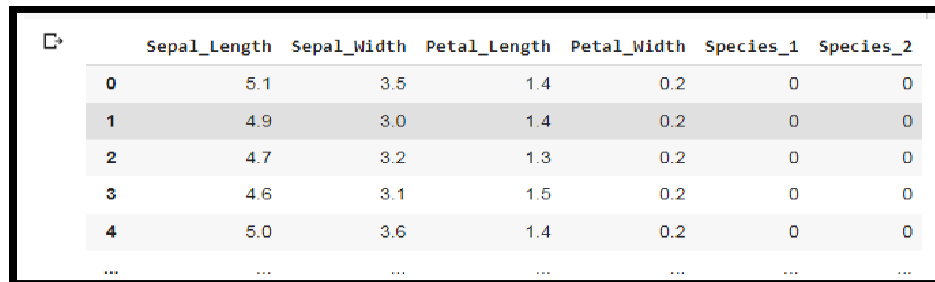
```
Output: array([0, 1, 2], dtype=int64)
```

**Step 6:** Apply one\_hot encoder with dummy variables for Species column.

```
one_hot_df = pd.get_dummies(df, prefix="Species",
                             columns=['Species'], drop_first=True)
```

**Step 7:** Observe the merge dataframe

```
one_hot_df
```



	Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species_1	Species_2
0	5.1	3.5	1.4	0.2	0	0
1	4.9	3.0	1.4	0.2	0	0
2	4.7	3.2	1.3	0.2	0	0
3	4.6	3.1	1.5	0.2	0	0
4	5.0	3.6	1.4	0.2	0	0

**Conclusion-** In this way we have explored the functions of the python library for Data Preprocessing, Data Wrangling Techniques and How to Handle missing values on Iris Dataset.

#### Assignment Question

1. Explain Data Frame with Suitable example.
2. What is the limitation of the label encoding method?
3. What is the need of data normalization?
4. What are the different Techniques for Handling the Missing Data?