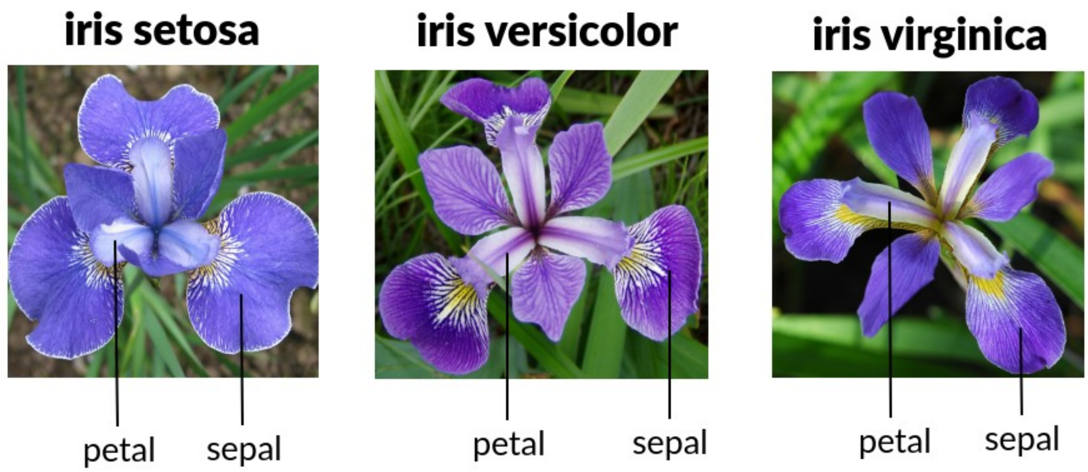
```python
import skimage.io as io
import matplotlib.pyplot as plt

img_path=r"C:\Users\Arigala.Adarsh\Downloads\iris-dataset.png"
from IPython.display import Image
Image(img_path)
```



Importing Necessary Packages

```python
import numpy as np
import pandas as pd
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

#Available Datasets Names
from sklearn import datasets
dir(datasets)
```

```
['__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__getattr__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '_arff_parser',
```

```
'_base',
'_california_housing',
'_covtype',
'_kddcup99',
'_lfw',
'_olivetti_faces',
'_openml',
'_rcv1',
'_samples_generator',
'_species_distributions',
'_svmlight_format_fast',
'_svmlight_format_io',
'_twenty_newsgroups',
'clear_data_home',
'data',
'descr',
'dump_svmlight_file',
'fetch_20newsgroups',
'fetch_20newsgroups_vectorized',
'fetch_california_housing',
'fetch_covtype',
'fetch_kddcup99',
'fetch_lfw_pairs',
'fetch_lfw_people',
'fetch_olivetti_faces',
'fetch_openml',
'fetch_rcv1',
'fetch_species_distributions',
'get_data_home',
'load_breast_cancer',
'load_diabetes',
'load_digits',
'load_files',
'load_iris',
'load_linnerud',
'load_sample_image',
'load_sample_images',
'load_svmlight_file',
'load_svmlight_files',
'load_wine',
'make_biclusters',
'make_blobs',
'make_checkerboard',
'make_circles',
'make_classification',
'make_friedman1',
'make_friedman2',
'make_friedman3',
'make_gaussian_quantiles',
```

```
 'make_hastie_10_2',
 'make_low_rank_matrix',
 'make_moons',
 'make_multilabel_classification',
 'make_regression',
 'make_s_curve',
 'make_sparse_coded_signal',
 'make_sparse_spd_matrix',
 'make_sparse_uncorrelated',
 'make_spd_matrix',
 'make_swiss_roll',
 'textwrap']
```

```
iris=datasets.load_iris()
iris.data
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
```

```
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
       [6.6, 2.9, 4.6, 1.3],
       [5.2, 2.7, 3.9, 1.4],
       [5. , 2. , 3.5, 1. ],
       [5.9, 3. , 4.2, 1.5],
       [6. , 2.2, 4. , 1. ],
       [6.1, 2.9, 4.7, 1.4],
       [5.6, 2.9, 3.6, 1.3],
       [6.7, 3.1, 4.4, 1.4],
       [5.6, 3. , 4.5, 1.5],
       [5.8, 2.7, 4.1, 1. ],
       [6.2, 2.2, 4.5, 1.5],
       [5.6, 2.5, 3.9, 1.1],
       [5.9, 3.2, 4.8, 1.8],
       [6.1, 2.8, 4. , 1.3],
       [6.3, 2.5, 4.9, 1.5],
       [6.1, 2.8, 4.7, 1.2],
       [6.4, 2.9, 4.3, 1.3],
       [6.6, 3. , 4.4, 1.4],
       [6.8, 2.8, 4.8, 1.4],
       [6.7, 3. , 5. , 1.7],
       [6. , 2.9, 4.5, 1.5],
       [5.7, 2.6, 3.5, 1. ],
       [5.5, 2.4, 3.8, 1.1],
       [5.5, 2.4, 3.7, 1. ],
       [5.8, 2.7, 3.9, 1.2],
```

```
       [6. , 2.7, 5.1, 1.6],
       [5.4, 3. , 4.5, 1.5],
       [6. , 3.4, 4.5, 1.6],
       [6.7, 3.1, 4.7, 1.5],
       [6.3, 2.3, 4.4, 1.3],
       [5.6, 3. , 4.1, 1.3],
       [5.5, 2.5, 4. , 1.3],
       [5.5, 2.6, 4.4, 1.2],
       [6.1, 3. , 4.6, 1.4],
       [5.8, 2.6, 4. , 1.2],
       [5. , 2.3, 3.3, 1. ],
       [5.6, 2.7, 4.2, 1.3],
       [5.7, 3. , 4.2, 1.2],
       [5.7, 2.9, 4.2, 1.3],
       [6.2, 2.9, 4.3, 1.3],
       [5.1, 2.5, 3. , 1.1],
       [5.7, 2.8, 4.1, 1.3],
       [6.3, 3.3, 6. , 2.5],
       [5.8, 2.7, 5.1, 1.9],
       [7.1, 3. , 5.9, 2.1],
       [6.3, 2.9, 5.6, 1.8],
       [6.5, 3. , 5.8, 2.2],
       [7.6, 3. , 6.6, 2.1],
       [4.9, 2.5, 4.5, 1.7],
       [7.3, 2.9, 6.3, 1.8],
       [6.7, 2.5, 5.8, 1.8],
       [7.2, 3.6, 6.1, 2.5],
       [6.5, 3.2, 5.1, 2. ],
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
```

```
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

```
iris
```

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
        [5.2, 3.5, 1.5, 0.2],
        [5.2, 3.4, 1.4, 0.2],
```

```
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
       [6.6, 2.9, 4.6, 1.3],
       [5.2, 2.7, 3.9, 1.4],
       [5. , 2. , 3.5, 1. ],
       [5.9, 3. , 4.2, 1.5],
       [6. , 2.2, 4. , 1. ],
       [6.1, 2.9, 4.7, 1.4],
       [5.6, 2.9, 3.6, 1.3],
       [6.7, 3.1, 4.4, 1.4],
       [5.6, 3. , 4.5, 1.5],
       [5.8, 2.7, 4.1, 1. ],
       [6.2, 2.2, 4.5, 1.5],
       [5.6, 2.5, 3.9, 1.1],
       [5.9, 3.2, 4.8, 1.8],
       [6.1, 2.8, 4. , 1.3],
       [6.3, 2.5, 4.9, 1.5],
       [6.1, 2.8, 4.7, 1.2],
       [6.4, 2.9, 4.3, 1.3],
       [6.6, 3. , 4.4, 1.4],
       [6.8, 2.8, 4.8, 1.4],
       [6.7, 3. , 5. , 1.7],
```

```
       [6. , 2.9, 4.5, 1.5],
       [5.7, 2.6, 3.5, 1. ],
       [5.5, 2.4, 3.8, 1.1],
       [5.5, 2.4, 3.7, 1. ],
       [5.8, 2.7, 3.9, 1.2],
       [6. , 2.7, 5.1, 1.6],
       [5.4, 3. , 4.5, 1.5],
       [6. , 3.4, 4.5, 1.6],
       [6.7, 3.1, 4.7, 1.5],
       [6.3, 2.3, 4.4, 1.3],
       [5.6, 3. , 4.1, 1.3],
       [5.5, 2.5, 4. , 1.3],
       [5.5, 2.6, 4.4, 1.2],
       [6.1, 3. , 4.6, 1.4],
       [5.8, 2.6, 4. , 1.2],
       [5. , 2.3, 3.3, 1. ],
       [5.6, 2.7, 4.2, 1.3],
       [5.7, 3. , 4.2, 1.2],
       [5.7, 2.9, 4.2, 1.3],
       [6.2, 2.9, 4.3, 1.3],
       [5.1, 2.5, 3. , 1.1],
       [5.7, 2.8, 4.1, 1.3],
       [6.3, 3.3, 6. , 2.5],
       [5.8, 2.7, 5.1, 1.9],
       [7.1, 3. , 5.9, 2.1],
       [6.3, 2.9, 5.6, 1.8],
       [6.5, 3. , 5.8, 2.2],
       [7.6, 3. , 6.6, 2.1],
       [4.9, 2.5, 4.5, 1.7],
       [7.3, 2.9, 6.3, 1.8],
       [6.7, 2.5, 5.8, 1.8],
       [7.2, 3.6, 6.1, 2.5],
       [6.5, 3.2, 5.1, 2. ],
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
```

```
        [6.1, 3. , 4.9, 1.8],
        [6.4, 2.8, 5.6, 2.1],
        [7.2, 3. , 5.8, 1.6],
        [7.4, 2.8, 6.1, 1.9],
        [7.9, 3.8, 6.4, 2. ],
        [6.4, 2.8, 5.6, 2.2],
        [6.3, 2.8, 5.1, 1.5],
        [6.1, 2.6, 5.6, 1.4],
        [7.7, 3. , 6.1, 2.3],
        [6.3, 3.4, 5.6, 2.4],
        [6.4, 3.1, 5.5, 1.8],
        [6. , 3. , 4.8, 1.8],
        [6.9, 3.1, 5.4, 2.1],
        [6.7, 3.1, 5.6, 2.4],
        [6.9, 3.1, 5.1, 2.3],
        [5.8, 2.7, 5.1, 1.9],
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'],
dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\
n--------------------\n\n**Data Set Characteristics:**\n\n    :Number
of Instances: 150 (50 in each of three classes)\n    :Number of
Attributes: 4 numeric, predictive attributes and the class\
n    :Attribute Information:\n        - sepal length in cm\n        -
sepal width in cm\n        - petal length in cm\n        - petal width
in cm\n        - class:\n                - Iris-Setosa\n
- Iris-Versicolour\n                - Iris-Virginica\
n                \n    :Summary Statistics:\n\n    ============== ====
==== ======= ===== ====================\n                    Min  Max
```

```
Mean    SD   Class Correlation\n    ============== ==== ==== =======
===== ====================\n    sepal length:   4.3  7.9   5.84   0.83
0.7826\n    sepal width:    2.0  4.4   3.05   0.43   -0.4194\n
petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)\n    petal
width:    0.1  2.5   1.20   0.76    0.9565  (high!)\n
============== ==== ==== ======= ===== ====================\n\
n    :Missing Attribute Values: None\n    :Class Distribution: 33.3%
for each of 3 classes.\n    :Creator: R.A. Fisher\n    :Donor: Michael
Marshall (MARSHALL%PLU@io.arc.nasa.gov)\n    :Date: July, 1988\n\nThe
famous Iris database, first used by Sir R.A. Fisher. The dataset is
taken\nfrom Fisher\'s paper. Note that it\'s the same as in R, but not
as in the UCI\nMachine Learning Repository, which has two wrong data
points.\n\nThis is perhaps the best known database to be found in the\
npattern recognition literature.  Fisher\'s paper is a classic in the
field and\nis referenced frequently to this day.  (See Duda & Hart,
for example.)  The\ndata set contains 3 classes of 50 instances each,
where each class refers to a\ntype of iris plant.  One class is
linearly separable from the other 2; the\nlatter are NOT linearly
separable from each other.\n\n.. topic:: References\n\n   - Fisher,
R.A. "The use of multiple measurements in taxonomic problems"\n
Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions
to\n     Mathematical Statistics" (John Wiley, NY, 1950).\n   - Duda,
R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n
(Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n
- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New
System\n     Structure and Classification Rule for Recognition in
Partially Exposed\n     Environments".  IEEE Transactions on Pattern
Analysis and Machine\n     Intelligence, Vol. PAMI-2, No. 1, 67-71.\n
- Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE
Transactions\n     on Information Theory, May 1972, 431-433.\n   - See
also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n
conceptual clustering system finds 3 classes in the data.\n   - Many,
many more ...',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'data_module': 'sklearn.datasets.data'}
```

```
iris.target
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
```

```
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

#Converting Dataset into DataFrame
df=pd.DataFrame(iris.data,columns=iris.feature_names)
df

     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
..                 ...               ...                ...
...
145                6.7               3.0                5.2
2.3
146                6.3               2.5                5.0
1.9
147                6.5               3.0                5.2
2.0
148                6.2               3.4                5.4
2.3
149                5.9               3.0                5.1
1.8

[150 rows x 4 columns]

df.head()

   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
```

```
 df.describe()
```

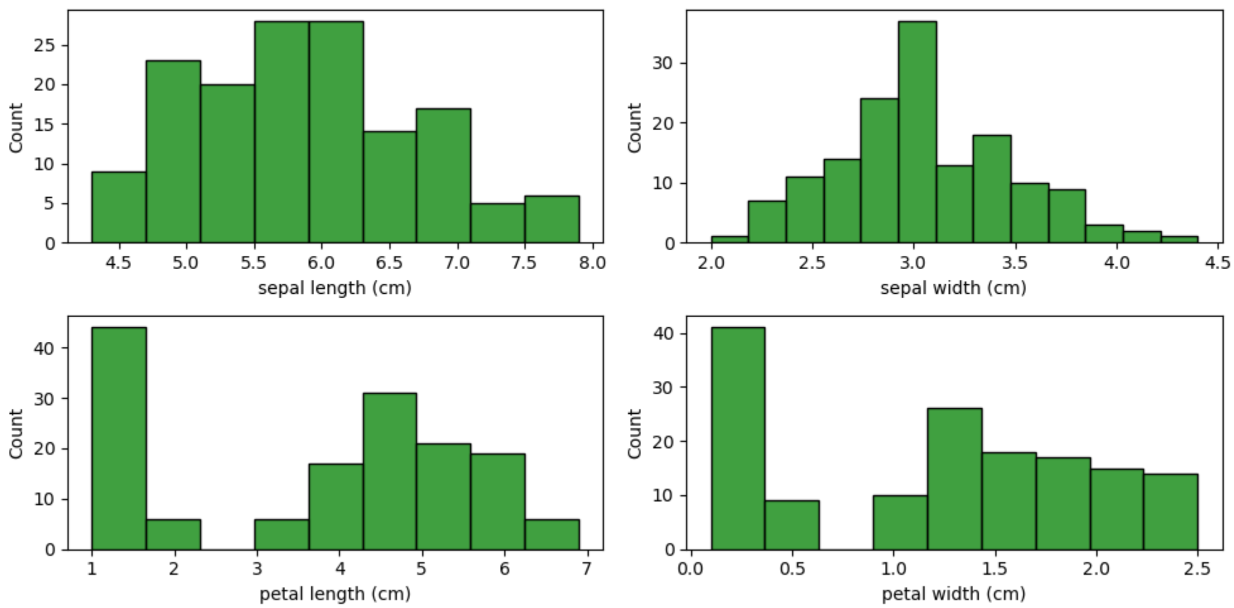
```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)
count        150.000000
mean           1.199333
std            0.762238
min            0.100000
25%            0.300000
50%            1.300000
75%            1.800000
max            2.500000
```

Insights from describe () : -

```
Sepal Length:
    The sepal length has a mean of approximately 5.84 cm.
    The values range from a minimum of 4.3 cm to a maximum of 7.9 cm.
    The data shows moderate variability, with a standard deviation of
approximately 0.83 cm.

Sepal Width:
    The sepal width has a mean of approximately 3.06 cm.
    The values range from a minimum of 2.0 cm to a maximum of 4.4 cm.
    The data has relatively low variability compared to the other
features, with a standard deviation of approximately 0.44 cm.

Petal Length:
    The petal length has a mean of approximately 3.76 cm.
    The values range from a minimum of 1.0 cm to a maximum of 6.9 cm.
    The data exhibits significant variability, with a relatively large
standard deviation of approximately 1.77 cm.

Petal Width:
    The petal width has a mean of approximately 1.20 cm.
    The values range from a minimum of 0.1 cm to a maximum of 2.5 cm.
    The data has moderate variability, with a standard deviation of
approximately 0.76 cm.
```

Exploratory Data Analysis

```
## Univariate Analysis of Numerical Data

plt.figure(figsize=(10,5))                     # Set Canvas size
plotnumber = 1                                  # Create a variable
for i in df.columns:                              # use for loop to iterate
Numerical Variable
    plt.subplot(2,2,plotnumber)                # set number of rows &
columns according to No.of Variable
    sns.histplot(x=df[i],color='green')   # plot histogram
    plotnumber = plotnumber + 1
plt.tight_layout()
```



Insights from univariate Analysis :-

```
Sepal Length:
    The histogram for sepal length shows a roughly normal distribution
with a peak around 5.8 cm.
    It appears to be unimodal (one peak) and relatively symmetric.
    The range of sepal length values spans from approximately 4.3 cm
to 7.9 cm.

Sepal Width:
    The histogram for sepal width shows a distribution that is
somewhat skewed to the right.
    It is unimodal with a peak around 3.0 cm.
    Most of the values cluster between 2.8 cm and 3.4 cm.

Petal Length:
```

```
    The histogram for petal length reveals a bimodal distribution with
clear separation between two groups.
    One mode is around 1.5 cm, and the other is around 4.5 cm.
    This bimodality indicates that there are two distinct groups of
iris species based on petal length.

Petal Width:
    The histogram for petal width also shows a bimodal distribution,
similar to petal length.
    One mode is around 0.2 cm, and the other is around 1.3 cm.
    The bimodal distribution suggests the presence of two distinct
groups based on petal width, corresponding to different species.
```

```
df['target']=iris.target
```

```
df.head()
```

```
   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

   target
0       0
1       0
2       0
3       0
4       0
```

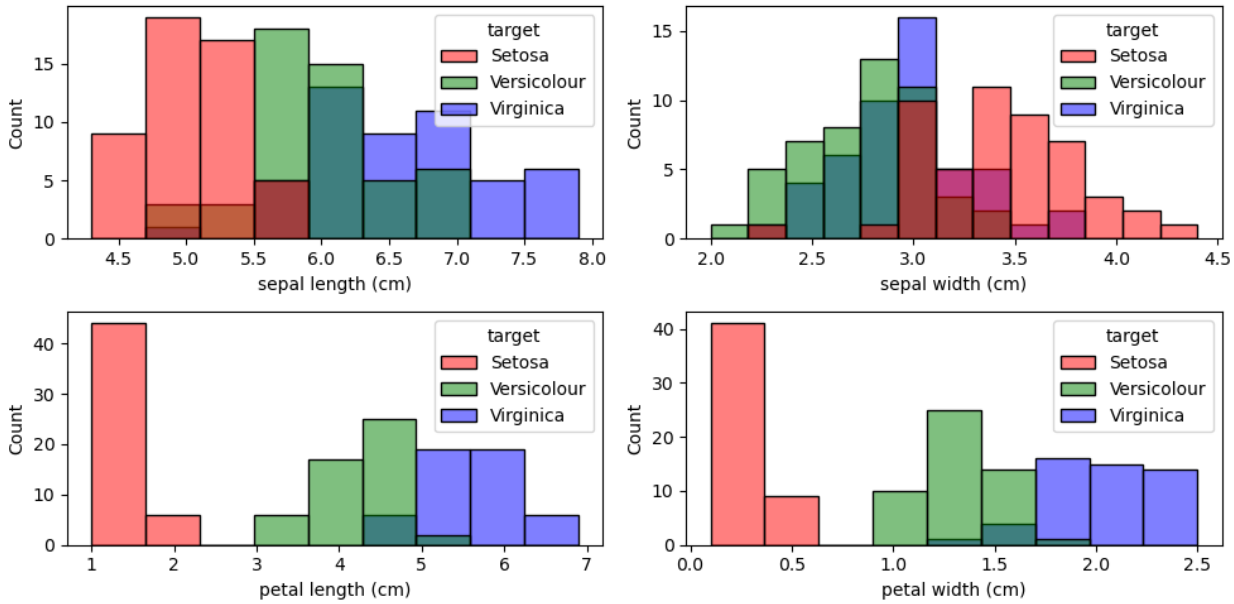
```
df.target.unique()
```

```
array([0, 1, 2])
```

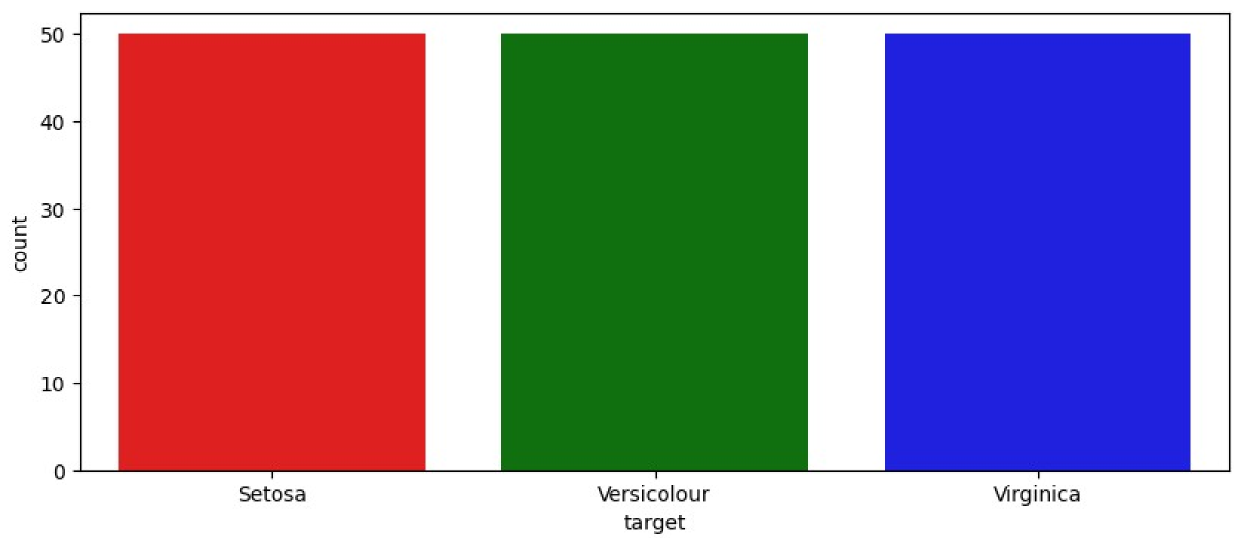
```
# Map the target values to custom class labels if desired
class_mapping = {
    0: "Setosa",
    1: "Versicolour",
    2: "Virginica"
}

df["target"] = df["target"].map(class_mapping)
df
```

```
     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
..                 ...               ...                ...
...
145                6.7               3.0                5.2
2.3
146                6.3               2.5                5.0
1.9
147                6.5               3.0                5.2
2.0
148                6.2               3.4                5.4
2.3
149                5.9               3.0                5.1
1.8

        target
0       Setosa
1       Setosa
2       Setosa
3       Setosa
4       Setosa
..         ...
145  Virginica
146  Virginica
147  Virginica
148  Virginica
149  Virginica

[150 rows x 5 columns]

plt.figure(figsize=(10,5))                # Set Canvas size
plotnumber = 1                             # Create a variable
for i in df.drop('target',axis=1):         # use for loop to iterate
Numerical Variable
    plt.subplot(2,2,plotnumber)           # set number of rows &
columns according to No.of Variable

sns.histplot(x=df[i],hue=df['target'],palette=['red','green','blue'])
# plot histogram
```

```
    plotnumber = plotnumber + 1
plt.tight_layout()
```

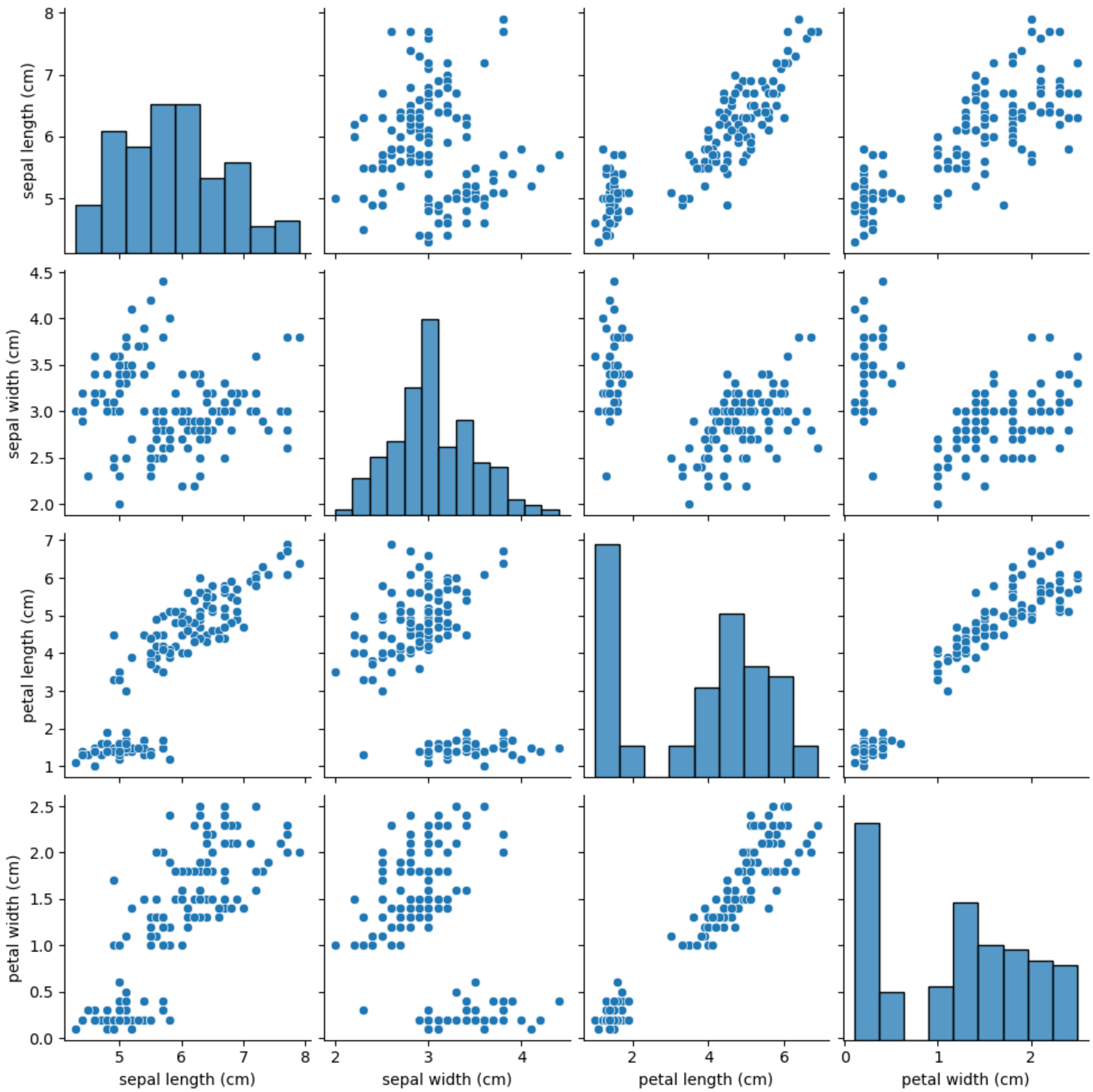



```
# count plot for Species
plt.figure(figsize=(10,4))
sns.countplot(x=df['target'],palette=['red','green','blue'])
plt.show()
```

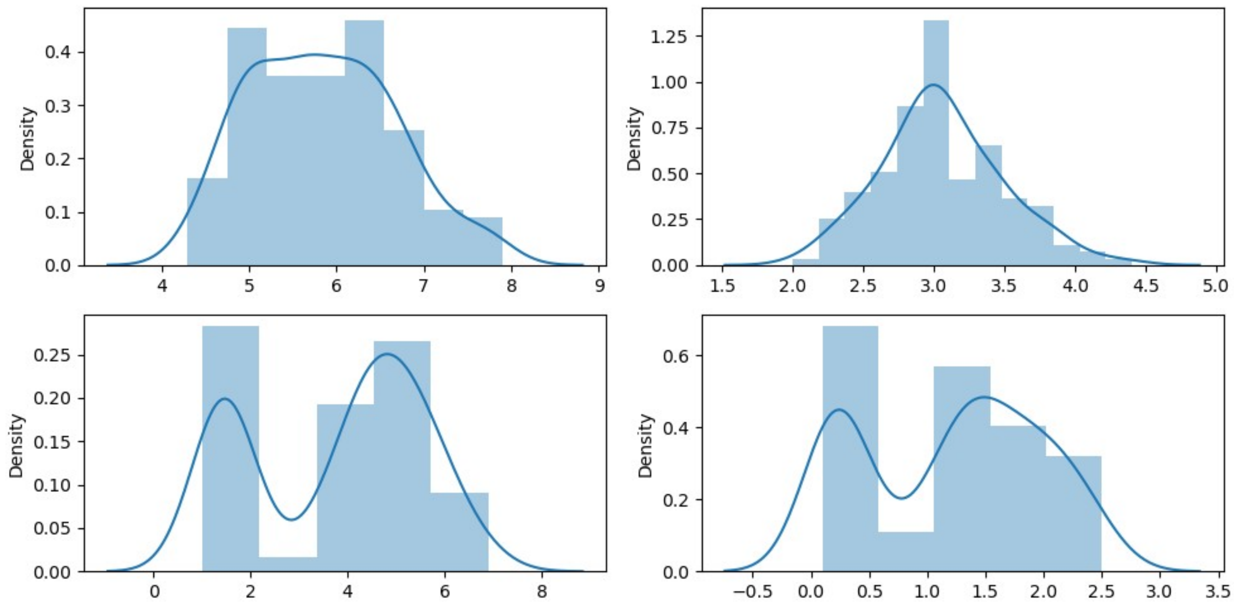



```
# multivariate Analysis
sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x147bfe835e0>
```



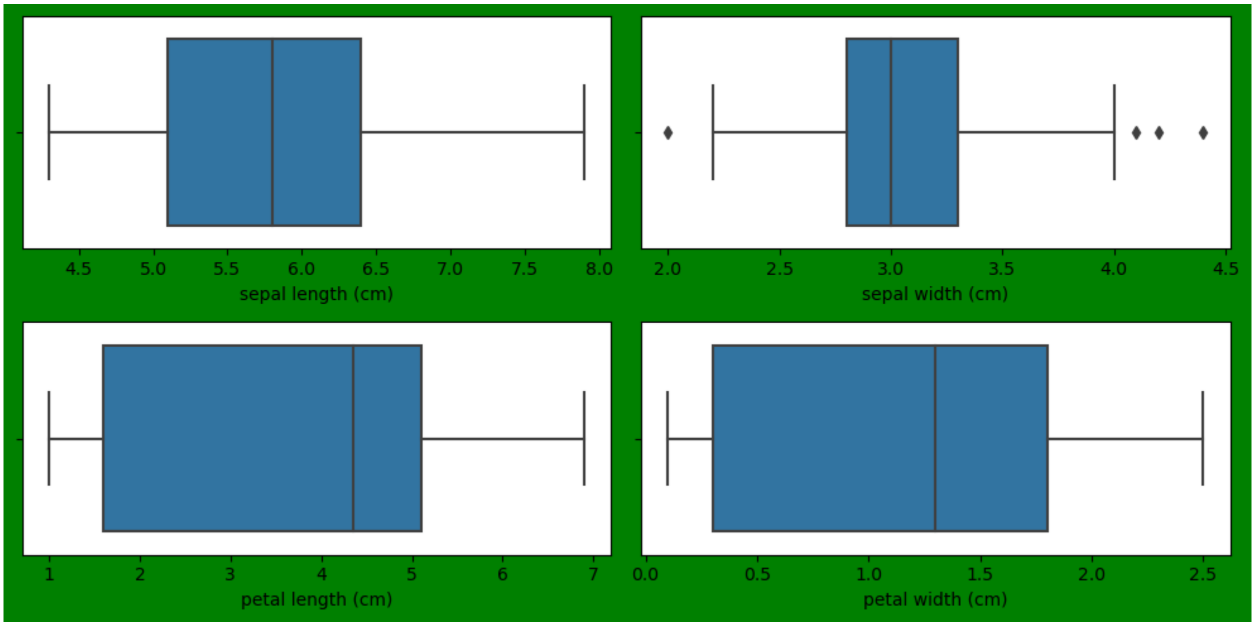

```
plt.figure(figsize=(10,5))                       # Set Canvas size
plotnumber = 1                                    # Create a variable
for i in df.drop('target',axis=1):                # use for loop to iterate
Numerical Variable
    plt.subplot(2,2,plotnumber)                  # set number of rows &
columns according to No.of Variable
    sns.distplot(x=df[i] )   # plot dist histogram
    plotnumber = plotnumber + 1
plt.tight_layout()
```



Data preproccesing

```
df.isnull().sum()

sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
target               0
dtype: int64

# value counts for Species
df['target'].value_counts()

Setosa          50
Versicolour     50
Virginica       50
Name: target, dtype: int64

from sklearn.preprocessing import LabelEncoder     # import Label
Encoder
encoder = LabelEncoder()                           # Create an
instance of the LabelEncoder

# Fit and transform the Species data
df['target'] = encoder.fit_transform(df['target'])


df.head()
```

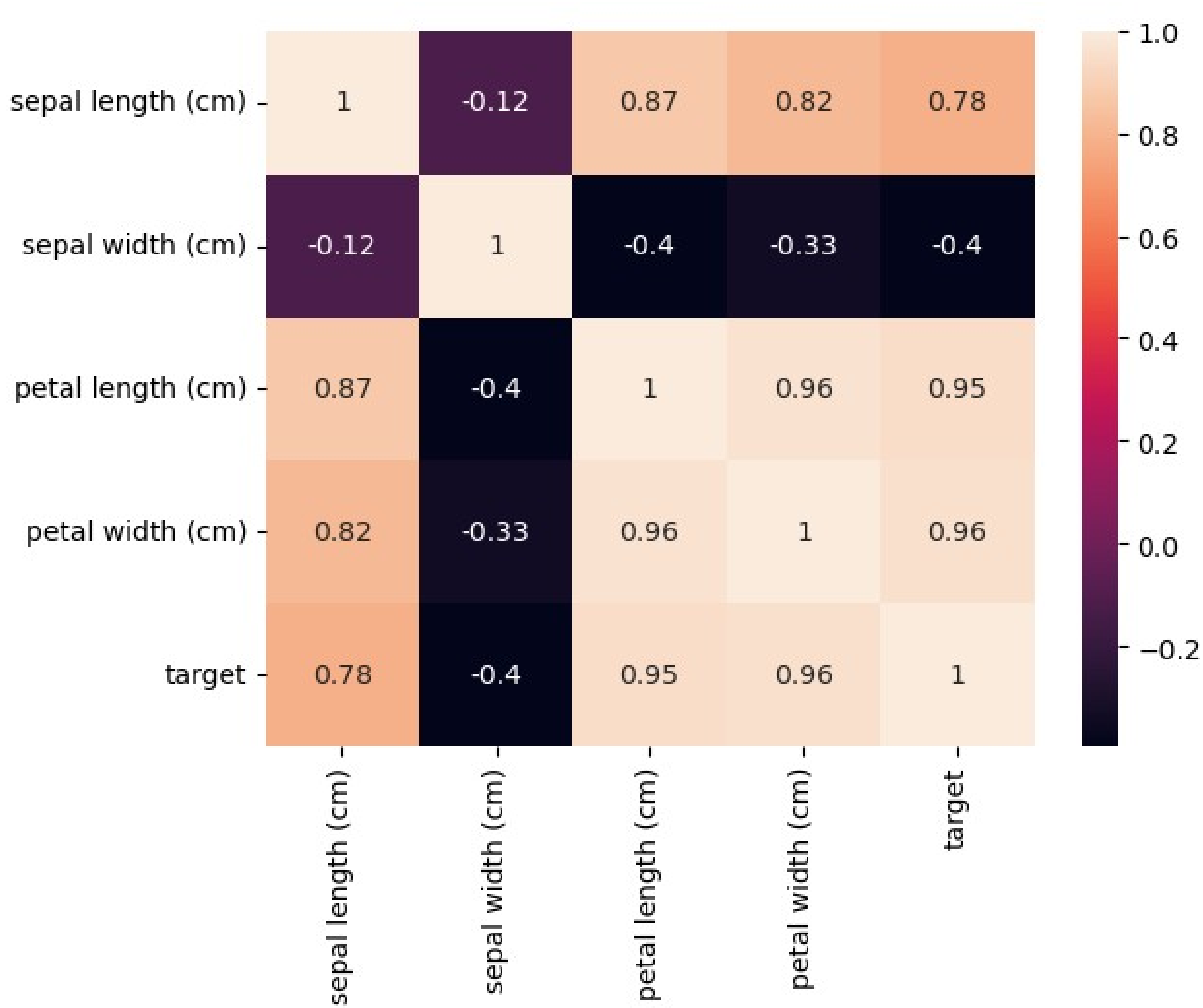
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)  \
0                5.1               3.5                1.4
0.2
1                4.9               3.0                1.4
0.2
2                4.7               3.2                1.3
0.2
3                4.6           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

   target
0       0
1       0
2       0
3       0
4       0

## plot Box - Plot to see the 5 number Summery

plt.figure(figsize=(10,5),facecolor='green')                       #
Set Canvas size
plotnumber = 1                                   # Create a variable
for i in df.drop('target',axis=1) :           # use for loop to
iterate Numerical Variable
    plt.subplot(2,2,plotnumber)                  # set number of rows
& columns according to No.of Variable
    sns.boxplot(x=df[i])           # plot Box - plot
    plotnumber = plotnumber + 1
plt.tight_layout()
```




```
## Sepal _ Width have outlier , so we will handle outlier

# Calculate the IQR (Interquartile Range)
Q1 = df['sepal width (cm)'].quantile(0.25)
Q3 = df['sepal width (cm)'].quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds
lower_bound = Q1 - (1.5 * IQR)
upper_bound = Q3 + (1.5 * IQR)

 df.loc[(df['sepal width (cm)'] < lower_bound) | (df['sepal width
(cm)'] > upper_bound),'sepal width (cm)'] = df['sepal width
(cm)'].median()

df.tail()

     sepal length (cm)  sepal width (cm)  petal length (cm)  petal
width (cm)  \
145                6.7               3.0                5.2
2.3
146                6.3               2.5                5.0
1.9
147                6.5               3.0                5.2
2.0
148                6.2               3.4                5.4
2.3
149                5.9               3.0                5.1
1.8

     target
```

```
145         2
146         2
147         2
148         2
149         2

## Check corellation using Heatmap

sns.heatmap(df.corr(),annot=True)
```

```
<AxesSubplot:>
```



Segregation of Data(Independent and Dependents)

```
# Dependent and Independent Variable Creation

x = df.drop('target',axis=1)       # Independent Features
y = df['target']

x

     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
..                 ...               ...                ...
...
145                6.7               3.0                5.2
2.3
146                6.3               2.5                5.0
1.9
147                6.5               3.0                5.2
2.0
148                6.2               3.4                5.4
2.3
149                5.9               3.0                5.1
1.8

[150 rows x 4 columns]

y

0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
```

```
149    2
Name: target, Length: 150, dtype: int32
```

Spliting Dataset into Test data and Train Data

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test
=train_test_split(x,y,test_size=0.2,random_state=1)

x_train

     sepal length (cm)  sepal width (cm)  petal length (cm)  petal
width (cm)
91                 6.1               3.0                4.6
1.4
135                7.7               3.0                6.1
2.3
69                 5.6               2.5                3.9
1.1
128                6.4               2.8                5.6
2.1
114                5.8               2.8                5.1
2.4
..                 ...               ...                ...
...
133                6.3               2.8                5.1
1.5
137                6.4               3.1                5.5
1.8
72                 6.3               2.5                4.9
1.5
140                6.7               3.1                5.6
2.4
37                 4.9               3.6                1.4
0.1

[120 rows x 4 columns]

x_test

     sepal length (cm)  sepal width (cm)  petal length (cm)  petal
width (cm)
14                 5.8               4.0                1.2
0.2
98                 5.1               2.5                3.0
1.1
75                 6.6               3.0                4.4
1.4
16                 5.4               3.9                1.3
```

```
0.4
131                 7.9                 3.8                 6.4
2.0
56                  6.3                 3.3                 4.7
1.6
141                 6.9                 3.1                 5.1
2.3
44                  5.1                 3.8                 1.9
0.4
29                  4.7                 3.2                 1.6
0.2
120                 6.9                 3.2                 5.7
2.3
94                  5.6                 2.7                 4.2
1.3
5                   5.4                 3.9                 1.7
0.4
102                 7.1                 3.0                 5.9
2.1
51                  6.4                 3.2                 4.5
1.5
78                  6.0                 2.9                 4.5
1.5
42                  4.4                 3.2                 1.3
0.2
92                  5.8                 2.6                 4.0
1.2
66                  5.6                 3.0                 4.5
1.5
31                  5.4                 3.4                 1.5
0.4
35                  5.0                 3.2                 1.2
0.2
90                  5.5                 2.6                 4.4
1.2
84                  5.4                 3.0                 4.5
1.5
77                  6.7                 3.0                 5.0
1.7
40                  5.0                 3.5                 1.3
0.3
125                 7.2                 3.2                 6.0
1.8
99                  5.7                 2.8                 4.1
1.3
33                  5.5                 3.0                 1.4
0.2
19                  5.1                 3.8                 1.5
0.3
```

```
73                  6.1               2.8                 4.7
1.2
146                 6.3               2.5                 5.0
1.9
```

y_train

```
91     1
135    2
69     1
128    2
114    2
      ..
133    2
137    2
72     1
140    2
37     0
Name: target, Length: 120, dtype: int32
```

y_test

```
14     0
98     1
75     1
16     0
131    2
56     1
141    2
44     0
29     0
120    2
94     1
5      0
102    2
51     1
78     1
42     0
92     1
66     1
31     0
35     0
90     1
84     1
77     1
40     0
125    2
99     1
33     0
19     0
```

```
73     1
146    2
Name: target, dtype: int32

x_train.shape

(120, 4)

x_test.shape

(30, 4)
```

Choosing the model

Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression

model=LogisticRegression()

model.fit(x_train,y_train)

LogisticRegression()

y_pred=model.predict(x_test)

y_pred

array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1,
1,
       2, 0, 2, 1, 0, 0, 1, 2])

a=pd.DataFrame(y_test.values,columns=['Actual'])

a

    Actual
0        0
1        1
2        1
3        0
4        2
5        1
6        2
7        0
8        0
9        2
10       1
```

11	0
12	2
13	1
14	1
15	0
16	1
17	1
18	0
19	0
20	1
21	1
22	1
23	0
24	2
25	1
26	0
27	0
28	1
29	2

```
a['Prediction']=y_pred
a
```

	Actual	Prediction
0	0	0
1	1	1
2	1	1
3	0	0
4	2	2
5	1	1
6	2	2
7	0	0
8	0	0
9	2	2
10	1	1
11	0	0
12	2	2
13	1	1
14	1	1
15	0	0
16	1	1
17	1	1
18	0	0
19	0	0
20	1	1
21	1	1
22	1	2
23	0	0
24	2	2
25	1	1

```
26       0           0
27       0           0
28       1           1
29       2           2
```

Model Evoluation

```python
from sklearn.metrics import accuracy_score , precision_score ,
recall_score , f1_score , classification_report , confusion_matrix

# print Confusion matrics
confusion_matrix(y_test,y_pred)
```

```
array([[11,  0,  0],
       [ 0, 12,  1],
       [ 0,  0,  6]], dtype=int64)
```

```python
# print Classification report
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      0.92      0.96        13
           2       0.86      1.00      0.92         6

    accuracy                           0.97        30
   macro avg       0.95      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30
```

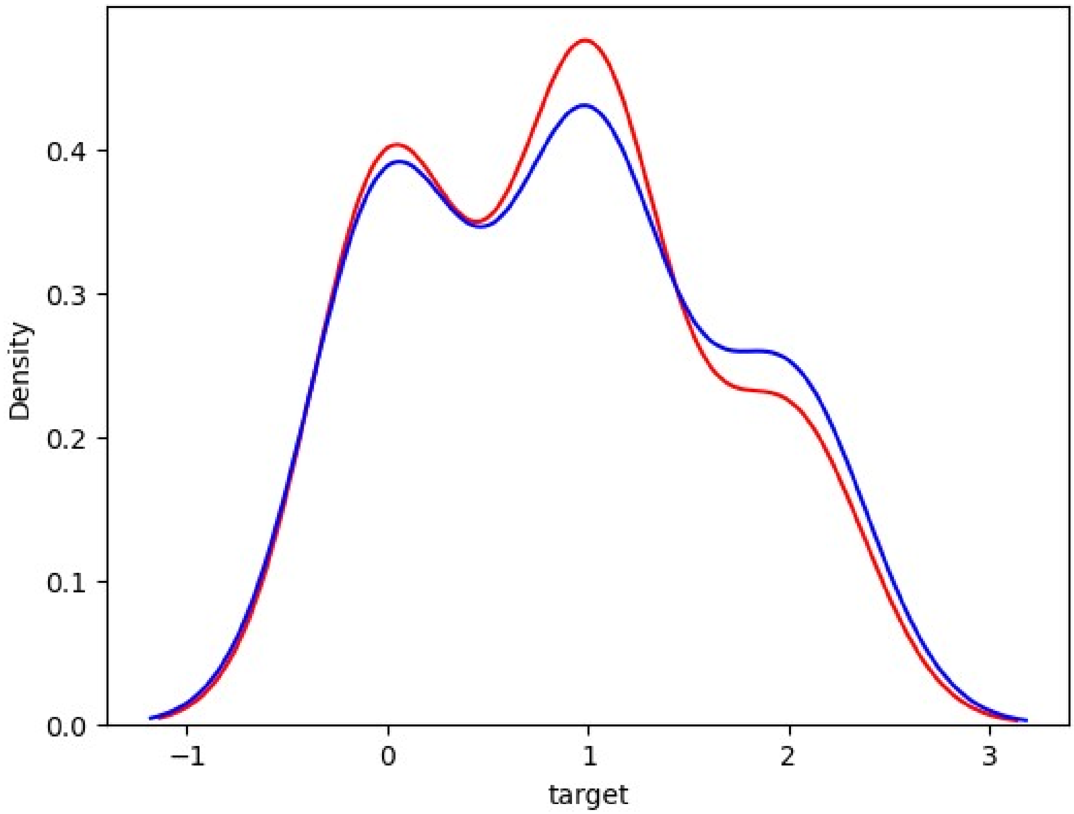
```python
f1_log = f1_score(y_test , y_pred , average='weighted')
print("f1_score of  : " , f1_log)
```

```
f1_score of  :  0.9672820512820512
```

```python
# test model performance
acc_log = accuracy_score(y_test,y_pred)
print("Accuracy of Logistic regression : ",acc_log)
```

```
Accuracy of Logistic regression :  0.9666666666666667
```

```python
sns.distplot(y_test,hist=False,color="red")
sns.distplot(y_pred,hist=False,color="blue")
plt.show()
```



Random Froest Classifier Model

```
from sklearn.ensemble import RandomForestClassifier
Random=RandomForestClassifier(random_state=1)
Random.fit(x_train,y_train)

RandomForestClassifier(random_state=1)

pred=Random.predict(x_test)
```

Model Evolution

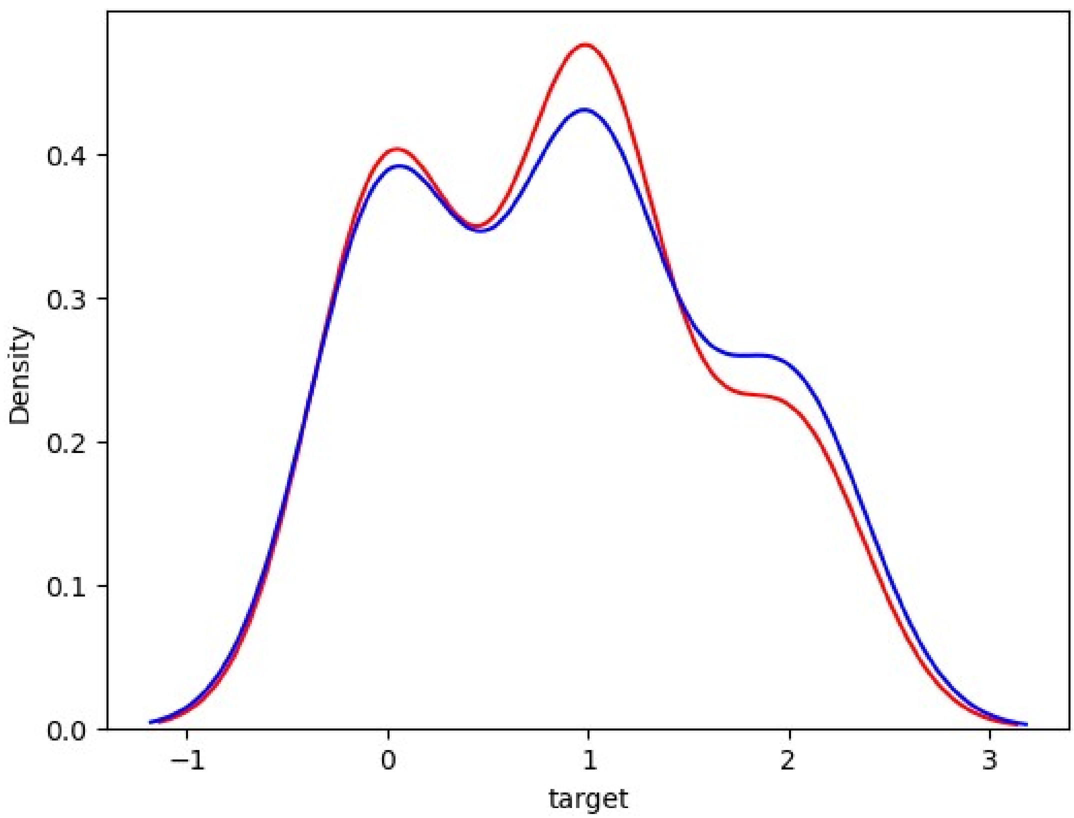
```
acc_forest = accuracy_score(y_test,pred)
print("Accuracy of Random forest classifier : ",acc_forest)

Accuracy of Random forest classifier :  0.9666666666666667

# print Classification report
print(classification_report(y_test,pred))

              precision    recall  f1-score   support
```

```
              0       1.00      1.00      1.00        11
              1       1.00      0.92      0.96        13
              2       0.86      1.00      0.92         6

       accuracy                           0.97        30
      macro avg       0.95      0.97      0.96        30
   weighted avg       0.97      0.97      0.97        30
```

```
f1_rand = f1_score(y_test , y_pred , average='weighted')
print("f1_score of  : " , f1_rand)
```

```
f1_score of  :  0.9672820512820512
```

```
sns.distplot(y_test,hist=False,color="red")
sns.distplot(pred,hist=False,color="blue")
plt.show()
```



________Hyperparameter Tunning of Random Forest Classifier ________

```
Hyperparameter Tunning is the process of finding the best set of
hyperparameters for a machine learning model to achieve optimal
performance on a given dataset.

hyperparameter tunning the model perform and calculate accuracy with
all possible parameter given to it internally and give the best
parameters which give the best performance.

Grid Search and Random Search are two commonly used methods for
Hyperparameter tunning.

# import GridSearchCV or RandomizedSearchCV to iterate through all
parameters and make a model with all combination

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV


# create a dictionary of hyperparameters with values

Randomforesthyperparameter={
    'n_estimators':[50,100,200,300,400,500],       # Number of trees
in the forest
    'max_depth':[None,range(1,20)],               # Maximum depth of
the trees
    'min_samples_split':[2,5,10],                # Minimum samples
required to split an internal node
    'min_samples_leaf':[1,2,4],                  # Minimum samples
required at a leaf node

    'max_features':['auto','sqrt','log2']       # Number of features
to consider when splitting


}
Random_hyper=RandomizedSearchCV(RandomForestClassifier(random_state=31
),

param_distributions=Randomforesthyperparameter,
                               n_jobs=-1,
                               n_iter=100,
                               cv=3,
                               verbose=3,
                               scoring='accuracy',
                               random_state=0
                            )
```

```
# Train random random search
Random_hyper.fit(x_train,y_train)


# print best parameters and score
print("Best Score:",Random_hyper.best_score_)
print("Best Hyperparameters:",Random_hyper.best_params_)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Best Score: 0.9416666666666665
Best Hyperparameters: {'n_estimators': 50, 'min_samples_split': 2,
'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': None}
```

```
Random=RandomForestClassifier(n_estimators=50,min_samples_split=2,min_
samples_leaf=1,max_features='log2',max_depth=None)
```

```
Random.fit(x_train,y_train)
```

```
RandomForestClassifier(max_features='log2', n_estimators=50)
```

```
random_predections=Random.predict(x_test)
```

```
f1_rand = f1_score(y_test ,random_predections, average='weighted')
print("f1_score of  : " , f1_rand)
```

```
f1_score of  :  0.9672820512820512
```

```
print(classification_report(random_predections,pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      1.00      1.00        12
           2       1.00      1.00      1.00         7

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
acc_hyper_forest=accuracy_score(random_predections,pred)
print(acc_hyper_forest)
```

```
1.0
```

________Hyperparameter Tunning of Logistic Regression ________

```
# creating dictionary --> key value pair of hyperparameters having key as parameter and values as its values
Log_Hyperpara = {
    'C': [0.001, 0.01, 0.1, 1, 10, 1000],                  # Regularization strength values
    'solver': ['lbfgs', 'liblinear', 'saga'],              # Solver options
    'max_iter':[100,200,400,600,800,1000],                 # maximum iteration
    'multi_class':['ovr','multinomial']                    # how to perform multiclass classification
}

# training data on gridsearch cv for finding best parameters
Log_grid = GridSearchCV(LogisticRegression(random_state=0),         # Estimator
                        param_grid=Log_Hyperpara,      # param_grid----> hyperparametes(dictionary we created)
                        scoring='accuracy',            # scoring---> performance matrix to check performance
                        cv=3,                          # cv------> number of flods
                        verbose=3,                     # verbose=Controls the verbosity: the higher, the more messages.
                        n_jobs=-1 # Number of jobs to run in parallel,-1 means using all processors.
                        )

# training data on gridsearch cv for finding best parameters
Log_grid.fit(x_train,y_train)

print(f"Best Score: {Log_grid.best_score_})")         # printing  best score
print(f"Best paramters: {Log_grid.best_params_})")    # printing  best parameters
```

```
Fitting 3 folds for each of 216 candidates, totalling 648 fits
Best Score: 0.9833333333333334)
Best paramters: {'C': 1, 'max_iter': 400, 'multi_class': 'multinomial', 'solver': 'saga'})
```

```
log_model=LogisticRegression(C=1,max_iter=400,multi_class='multinomial',solver='saga')
log_model.fit(x_train,y_train)
log_pred=log_model.predict(x_test)
```

```
f1_log = f1_score(y_test ,log_pred, average='weighted')
print("f1_score of  : " , f1_log)
```

```
f1_score of  :  1.0
```

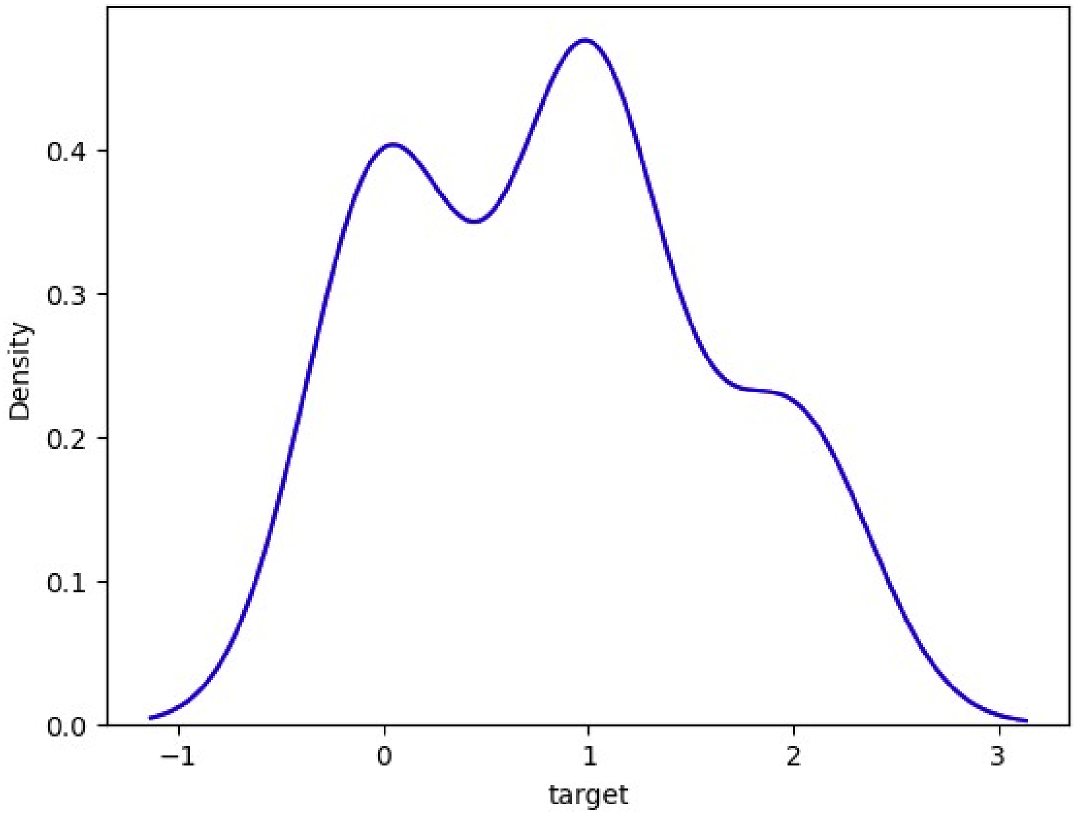
```
print(classification_report(log_pred,pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        11
           1       1.00      0.92      0.96        13
           2       0.86      1.00      0.92         6

    accuracy                           0.97        30
   macro avg       0.95      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30
```

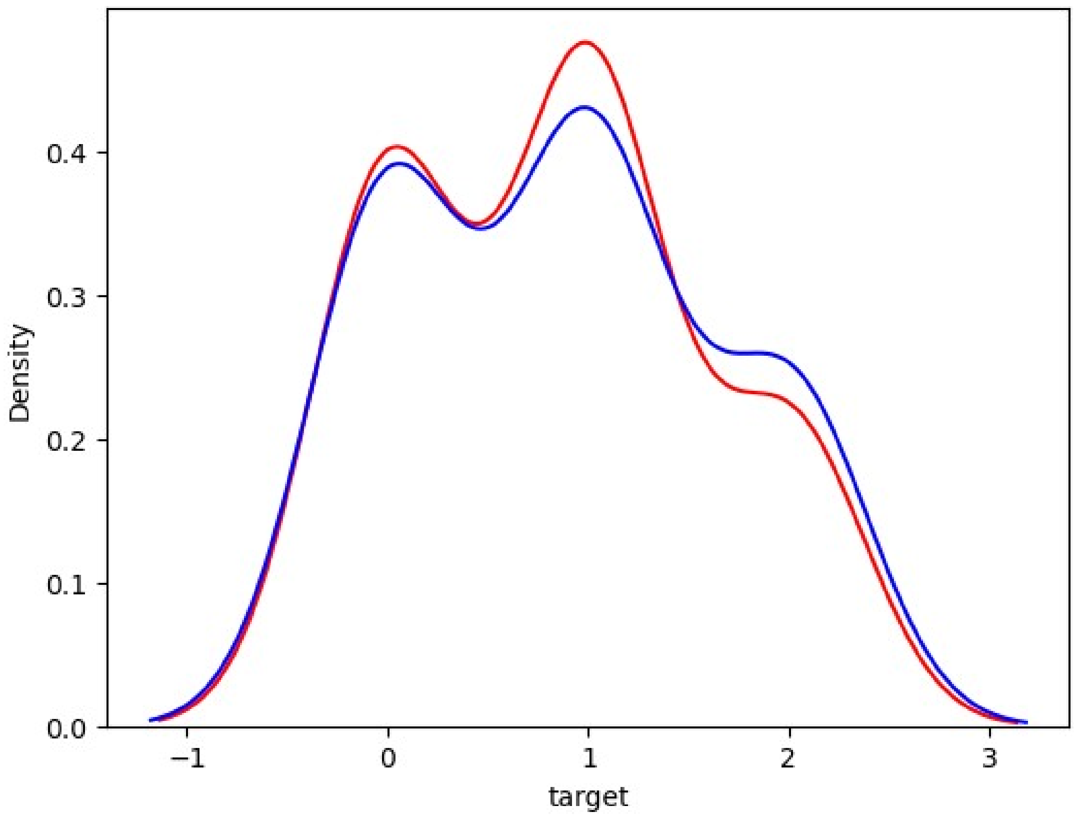
```
acc_hyper_log=accuracy_score(log_pred,pred)
print(acc_hyper_log)
```

```
0.9666666666666667
```

```
sns.distplot(y_test,hist=False,color="red")
sns.distplot(log_pred,hist=False,color="blue")

plt.show()
```




```
sns.distplot(y_test,hist=False,color="red")
sns.distplot(random_predections,hist=False,color="blue")

plt.show()
```




```
# Create a Data Frame of

Result_Comparison = pd.DataFrame({"Model_name" : ['Logistic
Regression','Random Forest'] ,

                        "Accuracy_Score" : [acc_log , acc_forest],

                        "F1_Score" : [f1_log , f1_rand] ,

                        "Hyperparameter_Score" : [acc_hyper_log ,
acc_hyper_forest]
                       })

Result_Comparison

            Model_name  Accuracy Score  F1 Score  Hyperparameter Score
0  Logistic Regression        0.966667  1.000000              0.966667
1        Random Forest        0.966667  0.967282              1.000000
```

Summary:

```
Both the Logistic Regression and Random Forest models perform well on
the given task, with high accuracy, F1 score, and hyperparameter
score.
```

Suggestion:

```
 The F1 Score for "Logistic Regression" suggests excellent
performance, it's essential to consider other factors like
interpretability, overfitting, dataset size, and computational
resources when choosing between the two models.
```