# Import Library

```
## import pandas as pd
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import datasets
from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV

import warnings
warnings.filterwarnings('ignore')
```

# Load the Dataset

```
# Load the Dataset from sklearn
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]]),
 '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, 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]),
 '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============== ==== ==== ======= =====
====================\nsepal length:   4.3  7.9   5.84   0.83
0.7826\nsepal width:    2.0  4.4   3.05   0.43   -0.4194\npetal
length:   1.0  6.9   3.76   1.76    0.9490  (high!)\npetal 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..
dropdown:: 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 ...\n',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'data_module': 'sklearn.datasets.data'}
```

# Convert Sklearn dataset into Dataframe

```
# Convert sklearn dataset into Dataframe
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

# Appemdig label to the Dataframe
df["target"] = iris.target

# View the converted Dataframe
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         0
1         0
2         0
3         0
4         0
..      ...
145       2
146       2
147       2
148       2
149       2

[150 rows x 5 columns]
```

# Dataset Description

```
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

:Number of Instances: 150 (50 in each of three classes)
:Number of Attributes: 4 numeric, predictive attributes and the class
:Attribute Information:
    - sepal length in cm
    - sepal width in cm
    - petal length in cm
    - petal width in cm
    - class:
            - Iris-Setosa
            - Iris-Versicolour
            - Iris-Virginica

:Summary Statistics:

============== ==== ==== ======= ===== ====================
                Min  Max   Mean    SD   Class Correlation
============== ==== ==== ======= ===== ====================
sepal length:   4.3  7.9   5.84   0.83    0.7826
sepal width:    2.0  4.4   3.05   0.43   -0.4194
petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
============== ==== ==== ======= ===== ====================

:Missing Attribute Values: None
:Class Distribution: 33.3% for each of 3 classes.
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset
is taken
from Fisher's paper. Note that it's the same as in R, but not as in
the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the
field and
is referenced frequently to this day.  (See Duda & Hart, for example.)
The
```

```
data set contains 3 classes of 50 instances each, where each class
refers to a
type of iris plant.  One class is linearly separable from the other 2;
the
latter are NOT linearly separable from each other.

.. dropdown:: References

  - Fisher, R.A. "The use of multiple measurements in taxonomic
problems"
    Annual Eugenics, 7, Part II, 179-188 (1936); also in
"Contributions to
    Mathematical Statistics" (John Wiley, NY, 1950).
  - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene
Analysis.
    (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
  - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New
System
    Structure and Classification Rule for Recognition in Partially
Exposed
    Environments".  IEEE Transactions on Pattern Analysis and Machine
    Intelligence, Vol. PAMI-2, No. 1, 67-71.
  - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE
Transactions
    on Information Theory, May 1972, 431-433.
  - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s
AUTOCLASS II
    conceptual clustering system finds 3 classes in the data.
  - Many, many more ...
```

```python
from IPython.display import Image
Image(url= "IRIS_Flower.png", width=700, height=600)
```

```
<IPython.core.display.Image object>
```

## Sneak Peak Data

```python
# Return the First n rows
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
```

```python
# Return the last n rows
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
```

```python
# Number of rows and columns in the dataset
df.shape
```

```
(150, 5)
```

```python
# General information about dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
```

```
 4   target             150 non-null    int32
dtypes: float64(4), int32(1)
memory usage: 5.4 KB
```

## Handling Missing Values

```python
# checks the null values and returns the sum of it
# ... Your answer here ...
# Checks for null values in the dataset
missing_values = df.isnull().sum()

# Display the sum of missing values
print("Missing values in each column:\n", missing_values)
```

```
Missing values in each column:
 sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
target               0
dtype: int64
```

## Check Randomness of the Dataframe

```python
# Shuffle the dataset to check randomness
df_shuffled = df.sample(frac=1,
random_state=42).reset_index(drop=True)

# Display the first 5 rows
print(df_shuffled.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)  \
0                6.1               2.8                4.7
1.2
1                5.7               3.8                1.7
0.3
2                7.7               2.6                6.9
2.3
3                6.0               2.9                4.5
1.5
4                6.8               2.8                4.8
1.4

   target
0       1
1       0
2       2
```

```
3       1
4       1

# Datafram before sorting
print("Before sorting")
df

Before sorting

     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         0
1         0
2         0
3         0
4         0
..      ...
145       2
146       2
147       2
148       2
149       2

[150 rows x 5 columns]
```

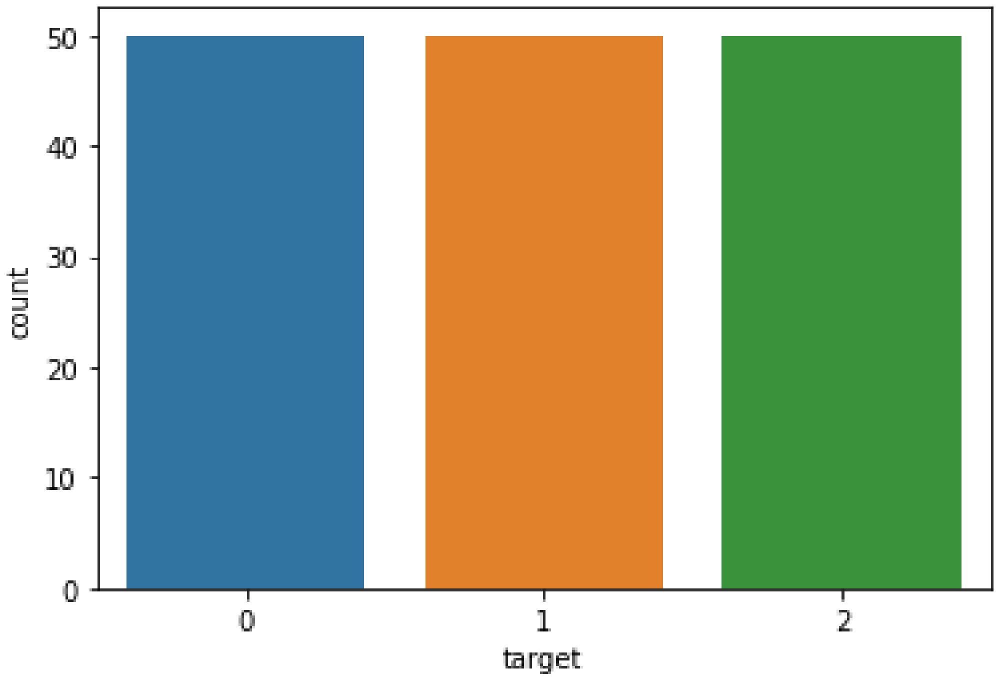
```python
# Sorting on Sepal width
df.sort_values("sepal width (cm)", axis = 0,
                 inplace = True, na_position ='last')

# Dataframe After Sorting
print("Afore sorting")
df
```

```
Afore sorting

     sepal length (cm)  sepal width (cm)  petal length (cm)  petal
width (cm)  \
60                 5.0               2.0                3.5
1.0
62                 6.0               2.2                4.0
1.0
119                6.0               2.2                5.0
1.5
68                 6.2               2.2                4.5
1.5
41                 4.5               2.3                1.3
0.3
..                 ...               ...                ...
...
16                 5.4               3.9                1.3
0.4
14                 5.8               4.0                1.2
0.2
32                 5.2               4.1                1.5
0.1
33                 5.5               4.2                1.4
0.2
15                 5.7               4.4                1.5
0.4

     target
60        1
62        1
119       2
68        1
41        0
..      ...
16        0
14        0
32        0
33        0
15        0

[150 rows x 5 columns]
```

## Exploratory Data Analysis

```
# Univariate analysis on targetnfeature.
sns.countplot(df['target'])
```

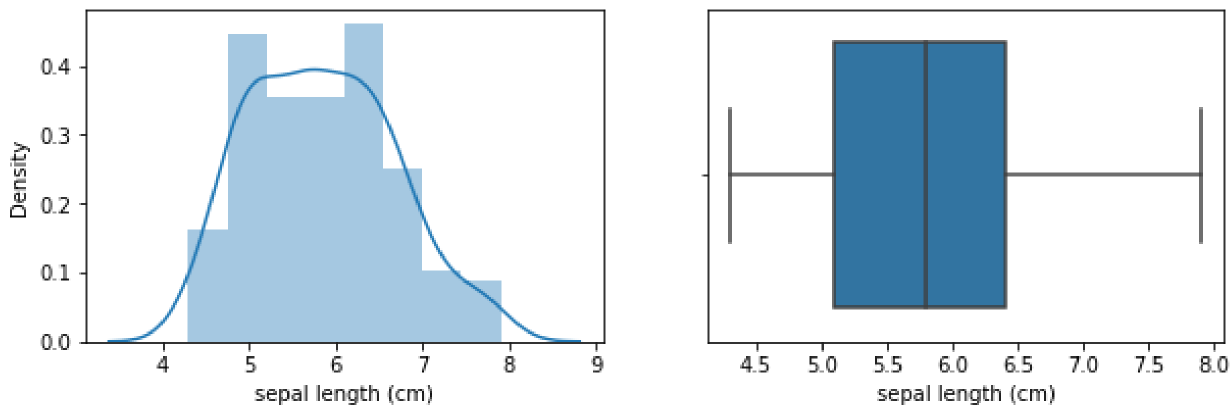
```
<AxesSubplot:xlabel='target', ylabel='count'>
```




- Here all the classes in target feature having equal number of counts. Hence it's advisable to choose this dataset.

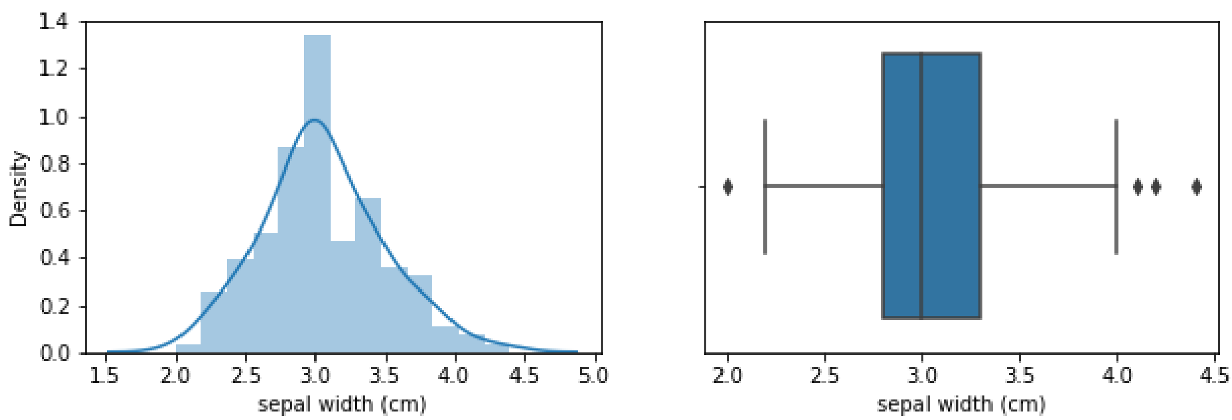
```
# Univariate analysis on Sepal length

f = plt.figure(figsize=(10,3))

f.add_subplot(1,2,1)
sns.distplot(df['sepal length (cm)'])

f.add_subplot(1,2,2)
sns.boxplot(df['sepal length (cm)'])
```

```
<AxesSubplot:xlabel='sepal length (cm)'>
```



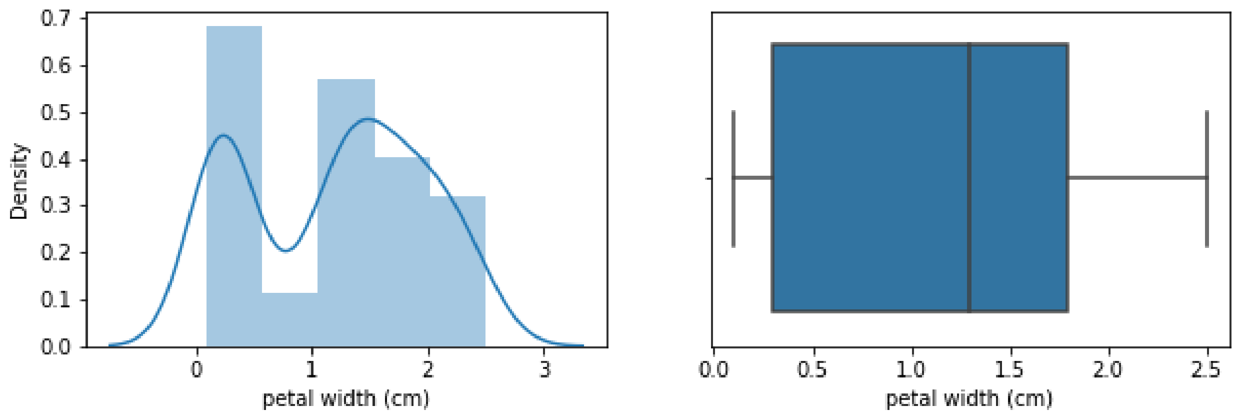

```
# Univariate analysis on Sepal Width

f = plt.figure(figsize=(10,3))

f.add_subplot(1,2,1)
sns.distplot(df['sepal width (cm)'])

f.add_subplot(1,2,2)
sns.boxplot(df['sepal width (cm)'])
```

```
<AxesSubplot:xlabel='sepal width (cm)'>
```

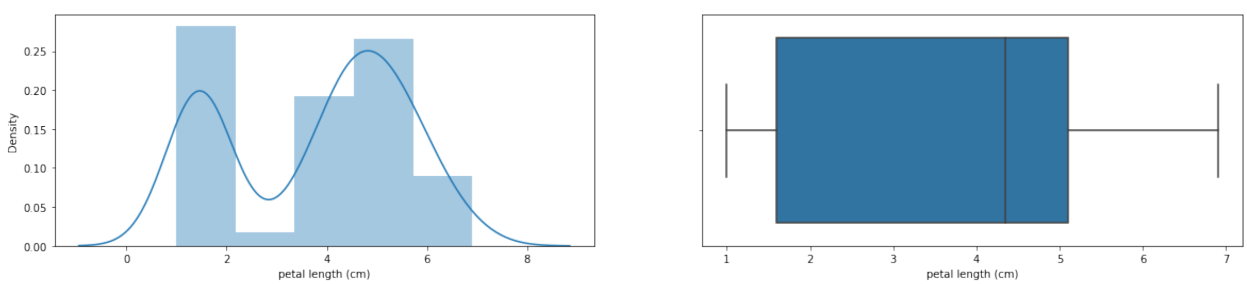



```
#Univariate analysis on Petal Width

f = plt.figure(figsize=(10,3))

f.add_subplot(1,2,1)
sns.distplot(df['petal width (cm)'])

f.add_subplot(1,2,2)
sns.boxplot(df['petal width (cm)'])
```

```
<AxesSubplot:xlabel='petal width (cm)'>
```




```
# Univariate analysis for Petal Length

f = plt.figure(figsize=(20,4))

f.add_subplot(1,2,1)
sns.distplot(df['petal length (cm)'])

f.add_subplot(1,2,2)
sns.boxplot(df['petal length (cm)'])

<AxesSubplot:xlabel='petal length (cm)'>
```




- Sepal length , Petal length and Petal width doesn't have outliers.
- Here Sepal width have outliers.

## KNN Model Development

```
# Create a KNN object
# ... Your answer here ...
# Create a KNN classifier object
knn = KNeighborsClassifier(n_neighbors=5)  # You can adjust
'n_neighbors' as needed

# Display the KNN object
print(knn)

KNeighborsClassifier()
```

```
# Create x and y variables
x = df.drop(columns=['target'])
y = df['target']
```

```
60     1
62     1
119    2
68     1
41     0
      ..
16     0
14     0
32     0
33     0
15     0
Name: target, Length: 150, dtype: int64
```

```
# Tranform the dataset
# ... Your answer here ...
# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the feature columns (excluding the target)
X_scaled = scaler.fit_transform(df.drop(columns=["target"]))

# Convert back to DataFrame for easy visualization (optional)
df_scaled = pd.DataFrame(X_scaled, columns=df.columns[:-1])

# Display the first 5 rows of the transformed dataset
print(df_scaled.head())
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width
(cm)
0          -0.900681          1.019004          -1.340227         -
1.315444
1          -1.143017         -0.131979          -1.340227         -
1.315444
2          -1.385353          0.328414          -1.397064         -
1.315444
3          -1.506521          0.098217          -1.283389         -
1.315444
4          -1.021849          1.249201          -1.340227         -
1.315444
```

```
# Split data into training and testing
# ... Your answer here ...
# Define features (X) and target (y)
X = df.drop(columns=["target"])  # Feature variables
y = df["target"]  # Target variable

# Split the dataset into 80% training and 20% testing
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

# Display the shape of training and testing sets
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (120, 4)
X_test shape: (30, 4)
y_train shape: (120,)
y_test shape: (30,)
```

```
# Train the KNN model on the training data
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

```
# Check model performance
knn.score(X_test,y_test)
```

```
1.0
```

# Model Evaluation

```
# Total number of Instances
y_test.value_counts()
```

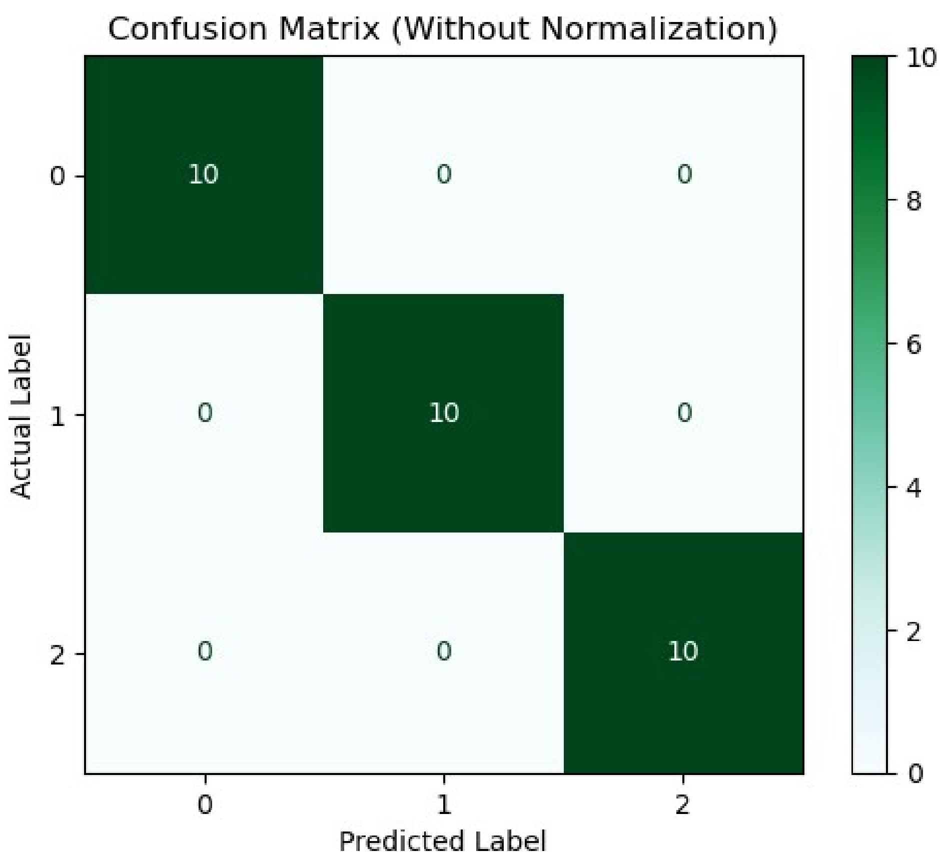
```
target
0    10
2    10
1    10
Name: count, dtype: int64
```

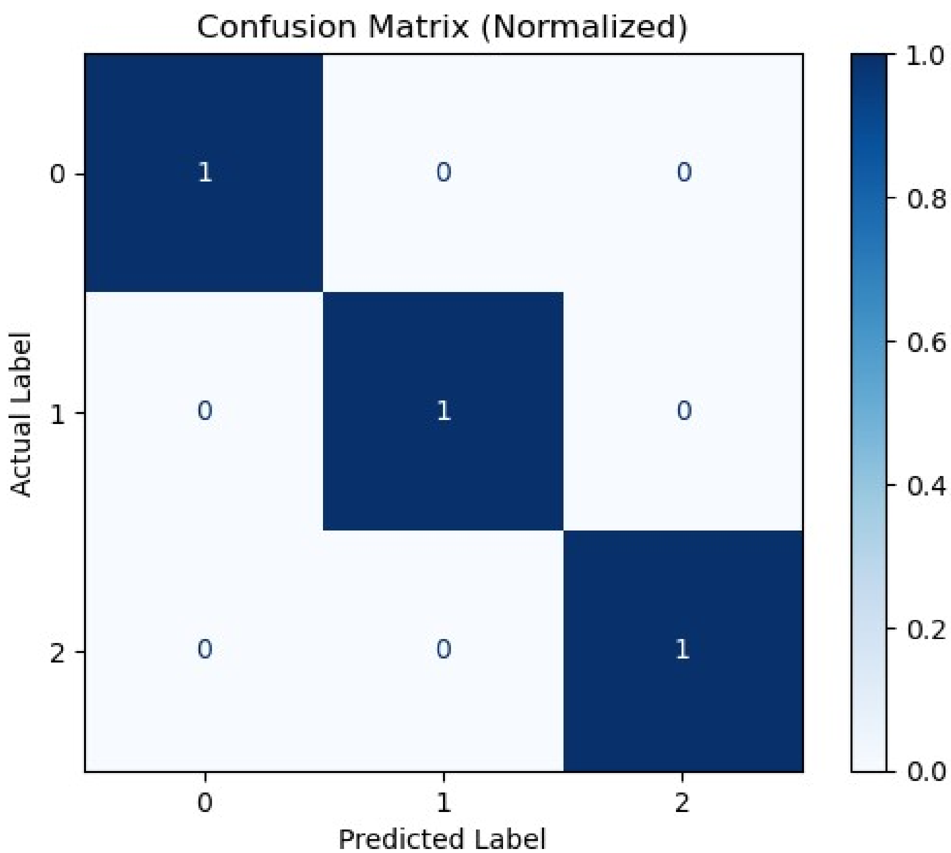
```
from sklearn.metrics import confusion_matrix
y_pred = knn.predict(X_test)
confusion_matrix(y_test, y_pred)
```

```
array([[10,  0,  0],
       [ 0, 10,  0],
       [ 0,  0, 10]], dtype=int64)
```

```
# https://matplotlib.org/stable/tutorials/colors/colormaps.html --->
Cmap colours

from sklearn.metrics import ConfusionMatrixDisplay

# Confusion Matrix without normalization
disp = ConfusionMatrixDisplay.from_estimator(knn, X_test, y_test,
```

```
                                             display_labels=['0', '1',
'2'],
                                             cmap=plt.cm.BuGn)
plt.title('Confusion Matrix (Without Normalization)')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show()

# Confusion Matrix with normalization
disp_norm = ConfusionMatrixDisplay.from_estimator(knn, X_test, y_test,

                                                  display_labels=['0',
'1', '2'],

                                                  cmap=plt.cm.Blues,
                                                  normalize='true')
plt.title('Confusion Matrix (Normalized)')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.show()
```






## Hyperparameter Tuning Using Grid Search

```python
# List of Hyperparameters to be tested
# n_neighbors = Numbers of neighbors
# leaf_size = reduces the time of execution of KNN
# p = 1:manhattan_distance, 2:Euclidean_distance.

# Define the hyperparameter grid
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],  # Different values for the number of neighbors
    'leaf_size': [10, 20, 30, 40],  # Optimizes search execution time
    'p': [1, 2]  # 1 = Manhattan Distance, 2 = Euclidean Distance
}

# Display the hyperparameter grid
print(param_grid)
```

```
{'n_neighbors': [3, 5, 7, 9, 11], 'leaf_size': [10, 20, 30, 40], 'p': [1, 2]}
```

```python
# Define hyperparameter options
leaf_size = [10, 20, 30, 40]  # Reduces execution time
n_neighbors = [3, 5, 7, 9, 11]  # Number of neighbors
```

```python
p = [1, 2]  # 1: Manhattan, 2: Euclidean
weights = ['uniform', 'distance']  # Weighting method

# Create a dictionary of hyperparameters
hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors,
p=p, weights=weights)

# Display the dictionary
print(hyperparameters)
```

```
{'leaf_size': [10, 20, 30, 40], 'n_neighbors': [3, 5, 7, 9, 11], 'p':
[1, 2], 'weights': ['uniform', 'distance']}
```

```python
# Import necessary libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

# Define the KNN model
knn_2 = KNeighborsClassifier()

# Define hyperparameter grid
hyperparameters = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'leaf_size': [10, 20, 30, 40],
    'p': [1, 2],  # Distance metric: 1 (Manhattan), 2 (Euclidean)
    'weights': ['uniform', 'distance']
}

# Perform Grid Search
grid_search = GridSearchCV(knn_2, hyperparameters, cv=5,
scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Get the best parameters
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)
```

```
Best Hyperparameters: {'leaf_size': 10, 'n_neighbors': 5, 'p': 2,
'weights': 'uniform'}
Best Accuracy: 0.975
```

```python
# cv is cross validation cv=10
clf = GridSearchCV(knn, hyperparameters, cv=5)

best_model = clf.fit(X,y)

# Best value hyperpaameters
print('Best leaf_size:', best_model.best_estimator_.get_params()
['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()
```

```
['n_neighbors'])
print('Best weights:', best_model.best_estimator_.get_params()
['weights'])
```

```
Best leaf_size: 10
Best p: 2
Best n_neighbors: 11
Best weights: distance
```

```
# Check model performance

# ... Your answer here ...
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

# Predict on the test data
y_pred = grid_search.best_estimator_.predict(X_test)

# Model Evaluation
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}\n")

# Classification Report
print("Classification Report:\n", classification_report(y_test,
y_pred))

# Confusion Matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 1.0000

Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        10
           2       1.00      1.00      1.00        10

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

Confusion Matrix:
 [[10  0  0]
 [ 0 10  0]
 [ 0  0 10]]
```

```
# Check the params-- old knn model
knn.get_params()
```

```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

```
#new model with hyperparameter tunning

best_model.best_estimator_.get_params()
```

```
{'algorithm': 'auto',
 'leaf_size': 10,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 11,
 'p': 2,
 'weights': 'distance'}
```