Predictive Analytics

ALY 6020, CRN 80405

Professor Vladimir Shapiro

Module 1: Midweek Project - Iris Classification

Submitted By - Richa Umesh Rambhia

Implementation of Nearest Neighbors - Iris Classification

Table of Contents

Introduction

Nearest Neighbor Model

K-nearest algorithm or KNN is one of the simplest classification and regression model in machine learning which is categorized into supervised learning and is implemented by calculating the distance between the data points in order to predict the category for the new data point. This algorithm uses the feature similarity concept in order to predict the values of the data points based on how closely the data point matches the points present in the training set. [2]

KNN algorithm calculates the distance between the two points and there are four methods of calculating the promixity of the data points, which are as follows.

1. Euclidean distance
2. Manhattan distance
3. Minkowski distance
4. Hamming distance

The most common method that is used for calculating the distance between the instances is the **Euclidean method**.

After the distance or the K-nearest neighbors are identified, the algorithm will now start predicting the class of the test data points based on the training dataset and the majority class of its K neighbors, where K will be the number of neighbors which is any integer value. [1]

The algorithm thus does not require much training but it can be expensive and memory-intensive when implemented for large datasets. Apart from that, the *K value* and the *distance metric* are important parameters that help to decide the **efficiency of the KNN algorithm**.

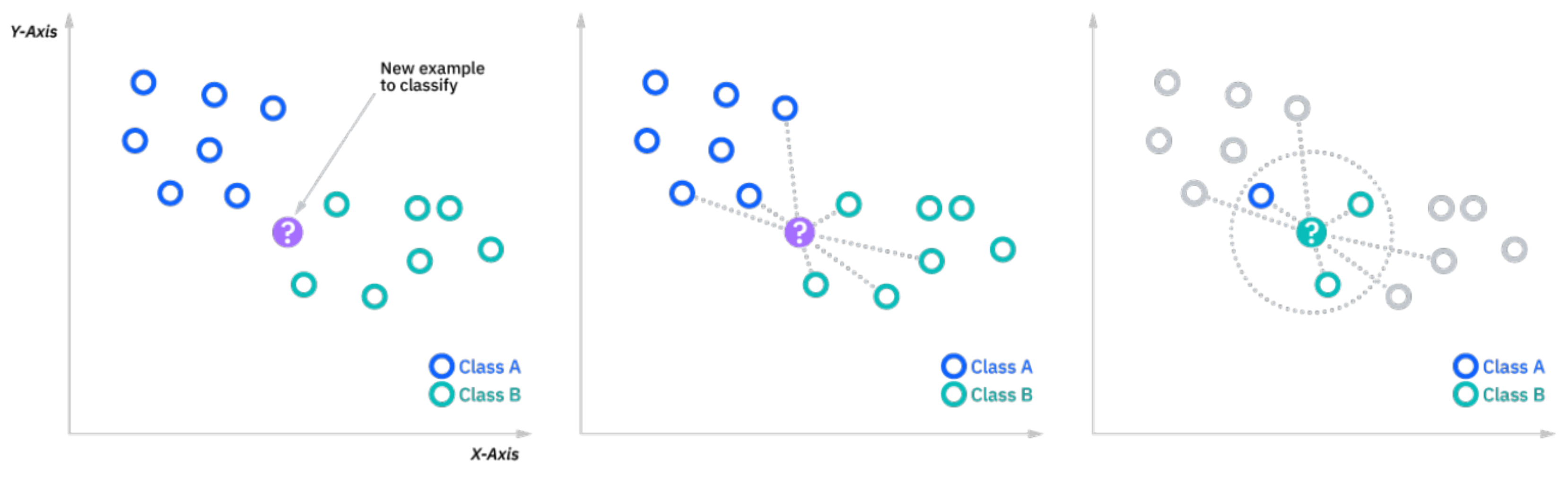


Figure 1. KNN Diagram [5]

The above diagram well explains the working of the KNN algorithm such that all the new data points are classified into either of the classes based on the distances between the two instances and depending on the number of neighbors selected for classification, the data is classified into *Class A or Class B*, for example, based on the majority of those classes present.

Iris Data Classification

The Iris dataset in machine learning is used for classification problems which has three classes or type of Iris flowers that needs to be classified based on the four features that the dataset contains: *Sepal Length, Sepal Width, Petal Length, Petal Width*. The types of classes in the Iris dataset are **Setosa, Versicolour, and Virginica.**

In the following project, the classes of the dataset or type of Iris flowers are classified based on its different parameters or features using the **K-Nearest or KNN algorithm**.

Analysis

Installing packages & libraries

```
In [11]: !pip install nbconvert[PDF]
```

```
Requirement already satisfied: nbconvert[PDF] in c:\users\rramb\appdata\local\programs\python\python38\lib\s
ite-packages (6.4.5)
Requirement already satisfied: traitlets>=5.0 in c:\users\rramb\appdata\local\programs\python\python38\lib\s
ite-packages (from nbconvert[PDF]) (5.1.1)
Requirement already satisfied: pygments>=2.4.1 in c:\users\rramb\appdata\local\programs\python\python38\lib
\site-packages (from nbconvert[PDF]) (2.10.0)
Requirement already satisfied: jinja2>=2.4 in c:\users\rramb\appdata\local\programs\python\python38\lib\site
-packages (from nbconvert[PDF]) (3.1.2)
Requirement already satisfied: defusedxml in c:\users\rramb\appdata\local\programs\python\python38\lib\site-
packages (from nbconvert[PDF]) (0.7.1)
Requirement already satisfied: jupyterlab-pygments in c:\users\rramb\appdata\local\programs\python\python38
\lib\site-packages (from nbconvert[PDF]) (0.1.2)
Requirement already satisfied: entrypoints>=0.2.2 in c:\users\rramb\appdata\local\programs\python\python38\l
ib\site-packages (from nbconvert[PDF]) (0.3)
Requirement already satisfied: bleach in c:\users\rramb\appdata\local\programs\python\python38\lib\site-pack
ages (from nbconvert[PDF]) (4.1.0)
Requirement already satisfied: beautifulsoup4 in c:\users\rramb\appdata\local\programs\python\python38\lib\s
ite-packages (from nbconvert[PDF]) (4.10.0)
Requirement already satisfied: nbclient<0.6.0,>=0.5.0 in c:\users\rramb\appdata\local\programs\python\python
38\lib\site-packages (from nbconvert[PDF]) (0.5.13)
Requirement already satisfied: jupyter-core in c:\users\rramb\appdata\local\programs\python\python38\lib\sit
e-packages (from nbconvert[PDF]) (4.9.1)
Requirement already satisfied: nbformat>=4.4 in c:\users\rramb\appdata\local\programs\python\python38\lib\si
te-packages (from nbconvert[PDF]) (5.3.0)
Requirement already satisfied: pandocfilters>=1.4.1 in c:\users\rramb\appdata\local\programs\python\python38
\lib\site-packages (from nbconvert[PDF]) (1.5.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\rramb\appdata\local\programs\python\python38\lib
\site-packages (from nbconvert[PDF]) (2.0.1)
Requirement already satisfied: testpath in c:\users\rramb\appdata\local\programs\python\python38\lib\site-pa
ckages (from nbconvert[PDF]) (0.6.0)
Requirement already satisfied: mistune<2,>=0.8.1 in c:\users\rramb\appdata\local\programs\python\python38\li
b\site-packages (from nbconvert[PDF]) (0.8.4)
Requirement already satisfied: nest-asyncio in c:\users\rramb\appdata\local\programs\python\python38\lib\sit
e-packages (from nbclient<0.6.0,>=0.5.0->nbconvert[PDF]) (1.5.1)
Requirement already satisfied: jupyter-client>=6.1.5 in c:\users\rramb\appdata\local\programs\python\python3
8\lib\site-packages (from nbclient<0.6.0,>=0.5.0->nbconvert[PDF]) (7.0.6)
Requirement already satisfied: jsonschema>=2.6 in c:\users\rramb\appdata\roaming\python\python38\site-packag
es (from nbformat>=4.4->nbconvert[PDF]) (3.2.0)
Requirement already satisfied: fastjsonschema in c:\users\rramb\appdata\local\programs\python\python38\lib\s
ite-packages (from nbformat>=4.4->nbconvert[PDF]) (2.15.3)
Requirement already satisfied: soupsieve>1.2 in c:\users\rramb\appdata\local\programs\python\python38\lib\si
te-packages (from beautifulsoup4->nbconvert[PDF]) (2.3.1)
Requirement already satisfied: webencodings in c:\users\rramb\appdata\local\programs\python\python38\lib\sit
e-packages (from bleach->nbconvert[PDF]) (0.5.1)
Requirement already satisfied: six>=1.9.0 in c:\users\rramb\appdata\local\programs\python\python38\lib\site-
packages (from bleach->nbconvert[PDF]) (1.12.0)
Requirement already satisfied: packaging in c:\users\rramb\appdata\local\programs\python\python38\lib\site-p
ackages (from bleach->nbconvert[PDF]) (23.0)
Requirement already satisfied: pywin32>=1.0 in c:\users\rramb\appdata\local\programs\python\python38\lib\sit
e-packages (from jupyter-core->nbconvert[PDF]) (302)
Requirement already satisfied: attrs>=17.4.0 in c:\users\rramb\appdata\local\programs\python\python38\lib\si
te-packages (from jsonschema>=2.6->nbformat>=4.4->nbconvert[PDF]) (21.4.0)
Requirement already satisfied: pyrsistent>=0.14.0 in c:\users\rramb\appdata\local\programs\python\python38\l
ib\site-packages (from jsonschema>=2.6->nbformat>=4.4->nbconvert[PDF]) (0.18.1)
Requirement already satisfied: setuptools in c:\users\rramb\appdata\local\programs\python\python38\lib\site-
packages (from jsonschema>=2.6->nbformat>=4.4->nbconvert[PDF]) (49.2.1)
Requirement already satisfied: tornado>=4.1 in c:\users\rramb\appdata\local\programs\python\python38\lib\sit
e-packages (from jupyter-client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert[PDF]) (6.2)
Requirement already satisfied: pyzmq>=13 in c:\users\rramb\appdata\local\programs\python\python38\lib\site-p
ackages (from jupyter-client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert[PDF]) (22.3.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\rramb\appdata\local\programs\python\python38
\lib\site-packages (from jupyter-client>=6.1.5->nbclient<0.6.0,>=0.5.0->nbconvert[PDF]) (2.8.2)

WARNING: Ignoring invalid distribution -illow (c:\users\rramb\appdata\roaming\python\python38\site-packages)
WARNING: Ignoring invalid distribution -treamlit (c:\users\rramb\appdata\local\programs\python\python38\lib
\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\rramb\appdata\roaming\python\python38\site-packages)
WARNING: Ignoring invalid distribution -treamlit (c:\users\rramb\appdata\local\programs\python\python38\lib
\site-packages)
WARNING: nbconvert 6.4.5 does not provide the extra 'pdf'
WARNING: Ignoring invalid distribution -illow (c:\users\rramb\appdata\roaming\python\python38\site-packages)
WARNING: Ignoring invalid distribution -treamlit (c:\users\rramb\appdata\local\programs\python\python38\lib
\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\rramb\appdata\roaming\python\python38\site-packages)
WARNING: Ignoring invalid distribution -treamlit (c:\users\rramb\appdata\local\programs\python\python38\lib
\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\rramb\appdata\roaming\python\python38\site-packages)
WARNING: Ignoring invalid distribution -illow (c:\users\rramb\appdata\roaming\python\python38\site-packages)
WARNING: There was an error checking the latest version of pip.
```

In [51]:
```
!pip install pandas-profiling
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-whee
ls/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Requirement already satisfied: pandas-profiling in /usr/local/lib/python3.9/dist-packages (3.6.6)
Requirement already satisfied: ydata-profiling in /usr/local/lib/python3.9/dist-packages (from pandas-profil
ing) (4.1.2)
Requirement already satisfied: requests<2.29,>=2.24.0 in /usr/local/lib/python3.9/dist-packages (from ydata-
profiling->pandas-profiling) (2.27.1)
Requirement already satisfied: matplotlib<3.7,>=3.2 in /usr/local/lib/python3.9/dist-packages (from ydata-pr
ofiling->pandas-profiling) (3.6.3)
Requirement already satisfied: phik<0.13,>=0.11.1 in /usr/local/lib/python3.9/dist-packages (from ydata-prof
iling->pandas-profiling) (0.12.3)
Requirement already satisfied: PyYAML<6.1,>=5.0.0 in /usr/local/lib/python3.9/dist-packages (from ydata-prof
iling->pandas-profiling) (6.0)
Requirement already satisfied: numpy<1.24,>=1.16.0 in /usr/local/lib/python3.9/dist-packages (from ydata-pro
filing->pandas-profiling) (1.22.4)
Requirement already satisfied: htmlmin==0.1.12 in /usr/local/lib/python3.9/dist-packages (from ydata-profili
ng->pandas-profiling) (0.1.12)
Requirement already satisfied: tqdm<4.65,>=4.48.2 in /usr/local/lib/python3.9/dist-packages (from ydata-prof
iling->pandas-profiling) (4.64.1)
Requirement already satisfied: seaborn<0.13,>=0.10.1 in /usr/local/lib/python3.9/dist-packages (from ydata-p
rofiling->pandas-profiling) (0.12.2)
Requirement already satisfied: typeguard<2.14,>=2.13.2 in /usr/local/lib/python3.9/dist-packages (from ydata
-profiling->pandas-profiling) (2.13.3)
Requirement already satisfied: multimethod<1.10,>=1.4 in /usr/local/lib/python3.9/dist-packages (from ydata-
profiling->pandas-profiling) (1.9.1)
Requirement already satisfied: pydantic<1.11,>=1.8.1 in /usr/local/lib/python3.9/dist-packages (from ydata-p
rofiling->pandas-profiling) (1.10.7)
Requirement already satisfied: pandas!=1.4.0,<1.6,>1.1 in /usr/local/lib/python3.9/dist-packages (from ydata
-profiling->pandas-profiling) (1.5.3)
Requirement already satisfied: visions[type_image_path]==0.7.5 in /usr/local/lib/python3.9/dist-packages (fr
om ydata-profiling->pandas-profiling) (0.7.5)
Requirement already satisfied: jinja2<3.2,>=2.11.1 in /usr/local/lib/python3.9/dist-packages (from ydata-pro
filing->pandas-profiling) (3.1.2)
Requirement already satisfied: imagehash==4.3.1 in /usr/local/lib/python3.9/dist-packages (from ydata-profil
ing->pandas-profiling) (4.3.1)
Requirement already satisfied: scipy<1.10,>=1.4.1 in /usr/local/lib/python3.9/dist-packages (from ydata-prof
iling->pandas-profiling) (1.9.3)
Requirement already satisfied: statsmodels<0.14,>=0.13.2 in /usr/local/lib/python3.9/dist-packages (from yda
ta-profiling->pandas-profiling) (0.13.5)
Requirement already satisfied: pillow in /usr/local/lib/python3.9/dist-packages (from imagehash==4.3.1->ydat
a-profiling->pandas-profiling) (8.4.0)
Requirement already satisfied: PyWavelets in /usr/local/lib/python3.9/dist-packages (from imagehash==4.3.1->
ydata-profiling->pandas-profiling) (1.4.1)
Requirement already satisfied: attrs>=19.3.0 in /usr/local/lib/python3.9/dist-packages (from visions[type_im
age_path]==0.7.5->ydata-profiling->pandas-profiling) (22.2.0)
Requirement already satisfied: tangled-up-in-unicode>=0.0.4 in /usr/local/lib/python3.9/dist-packages (from
visions[type_image_path]==0.7.5->ydata-profiling->pandas-profiling) (0.2.0)
Requirement already satisfied: networkx>=2.4 in /usr/local/lib/python3.9/dist-packages (from visions[type_im
age_path]==0.7.5->ydata-profiling->pandas-profiling) (3.1)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.9/dist-packages (from jinja2<3.2,>=
2.11.1->ydata-profiling->pandas-profiling) (2.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.9/dist-packages (from matplotl
ib<3.7,>=3.2->ydata-profiling->pandas-profiling) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib<
3.7,>=3.2->ydata-profiling->pandas-profiling) (1.4.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib<3.
7,>=3.2->ydata-profiling->pandas-profiling) (23.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib<3.7,>
=3.2->ydata-profiling->pandas-profiling) (0.11.0)
Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib<
3.7,>=3.2->ydata-profiling->pandas-profiling) (3.0.9)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib<
3.7,>=3.2->ydata-profiling->pandas-profiling) (1.0.7)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib<
3.7,>=3.2->ydata-profiling->pandas-profiling) (4.39.3)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas!=1.4.0,<
1.6,>1.1->ydata-profiling->pandas-profiling) (2022.7.1)
Requirement already satisfied: joblib>=0.14.1 in /usr/local/lib/python3.9/dist-packages (from phik<0.13,>=0.
11.1->ydata-profiling->pandas-profiling) (1.2.0)
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.9/dist-packages (from pyda
ntic<1.11,>=1.8.1->ydata-profiling->pandas-profiling) (4.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests<
2.29,>=2.24.0->ydata-profiling->pandas-profiling) (2022.12.7)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.9/dist-packages (from request
s<2.29,>=2.24.0->ydata-profiling->pandas-profiling) (1.26.15)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests<2.29,>=
2.24.0->ydata-profiling->pandas-profiling) (3.4)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/dist-packages (from req
uests<2.29,>=2.24.0->ydata-profiling->pandas-profiling) (2.0.12)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.9/dist-packages (from statsmodels<0.1
4,>=0.13.2->ydata-profiling->pandas-profiling) (0.5.3)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from patsy>=0.5.2->statsmodels
<0.14,>=0.13.2->ydata-profiling->pandas-profiling) (1.16.0)
```

Importing libraries

```
In [12]: import nbconvert
import pandas as pd
import numpy as np
import pandas_profiling
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
```

Loading the Iris dataset

```
In [3]: # loading the dataset using the sklearn library

iris = datasets.load_iris()
```

Converting the dataset into a pandas DataFrame

```
In [4]: iris_data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_data['target'] = iris.target
```

```
In [55]: print(iris_data)
```

```
     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]
```

Table 1. Iris Classification Data

Exploratory Data Analysis

Exploratory Data Analysis or EDA is performed on the Iris dataset in order analyze and understand the structure of the data and to explore the various patterns and relationships between the field values of the dataset.

```
In [56]: print("Size of the dataset:", iris_data.size)
         print("\n-----------------------------------------------------------------------\n")

         print("Column Names/Field Values\n")
         print(iris_data.columns)
         print("\n-----------------------------------------------------------------------\n")

         print("Starting rows of the dataset\n")
         print(iris_data.head())
         print("\n-----------------------------------------------------------------------\n")

         print("Last rows of the dataset\n")
         print(iris_data.tail())
         print("\n-----------------------------------------------------------------------\n")

         print("Total number of rows & columns:", iris_data.shape)
```

```
Size of the dataset: 750

-----------------------------------------------------------------------

Column Names/Field Values

Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'target'],
      dtype='object')

-----------------------------------------------------------------------

Starting rows of the dataset

   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

-----------------------------------------------------------------------

Last rows of the dataset

     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

-----------------------------------------------------------------------

Total number of rows & columns: (150, 5)
```

Table 2. Exploratory Data Analysis

Statistical Analysis

In [57]:
```
print("Describing the dataset \n")
print(round(iris_data.describe(),2))

print("\n--------------------------------------------------------------------\n")

print("Dataset Info \n")
print(iris_data.info())
```

```
Describing the dataset 

       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count             150.00            150.00             150.00   
mean                5.84              3.06               3.76   
std                 0.83              0.44               1.77   
min                 4.30              2.00               1.00   
25%                 5.10              2.80               1.60   
50%                 5.80              3.00               4.35   
75%                 6.40              3.30               5.10   
max                 7.90              4.40               6.90   

       petal width (cm)  target  
count            150.00  150.00  
mean               1.20    1.00  
std                0.76    0.82  
min                0.10    0.00  
25%                0.30    0.00  
50%                1.30    1.00  
75%                1.80    2.00  
max                2.50    2.00  

--------------------------------------------------------------------

Dataset 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    int64  
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
None
```

Table 3. Descriptive and Statistical Analysis

In the **descriptive and statistical analysis** implemented above, various measures for the parameters of the dataset are analyzed and it is observed that the *minimum, maximum, standard deviation, count, etc.* values for each of the field value is known which further helps in the analysis to better understand the dataset.

Data Profiling

In [58]:
```
# Generating the data profiling report

iris_report = iris_data.profile_report(title='Iris Data Profiling Report', explorative = True)
iris_report
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]

Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]

Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Out[58]:

In [59]:
```
# Saving the report to HTML

iris_report.to_file(output_file = "Iris Data Profiling Report.html")
```

```
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

From the data profiling report generated for the Iris dataset, it is observed that there are **150 rows of data** and **5 field values**, of which *4 are numerical type and 1 categorical data type*.

There are no missing values in the dataset and the percentage of duplicate rows is 0.7%, i.e., it has 1 duplicate row of data.

Input Data Visualization

In [60]:
```
data_values = iris.data[:, :4]
target_values = iris.target
```

Refered from [Xuhewen, 2016 (https://www.kaggle.com/code/xuhewen/iris-dataset-visualization-and-machine-learning)](https://www.kaggle.com/code/xuhewen/iris-dataset-visualization-and-machine-learning)

In [61]:
```
# Visualizing data points for Sepal Length & Width

colors = ['purple','green','blue']
for i in range(3):
  plt.scatter(data_values[target_values == i, 0], data_values[target_values == i, 1], label=iris.target_names

plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Iris Data Classification - Sepal Length & Width')
plt.legend()
plt.show()
```

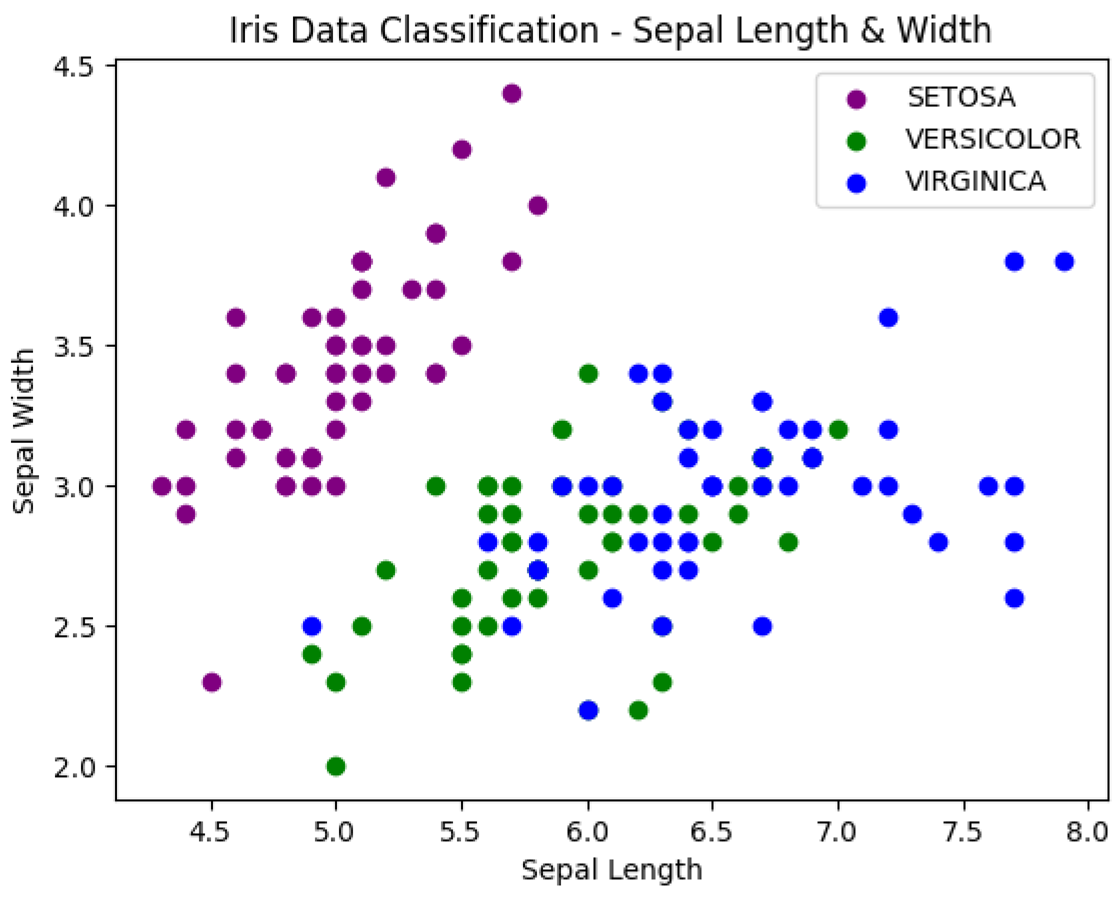


Figure 2. Iris Data Classification - Sepal Length & Width

```
In [62]: # Visualizing data points for Petal Length & Width

         colors = ['purple','green','blue']
         for i in range(3):
           plt.scatter(data_values[target_values == i, 2], data_values[target_values == i, 3], label=iris.target_names

         plt.xlabel('Petal Length')
         plt.ylabel('Petal Width')
         plt.title('Iris Data Classification - Petal Length & Width')
         plt.legend()
         plt.show()
```

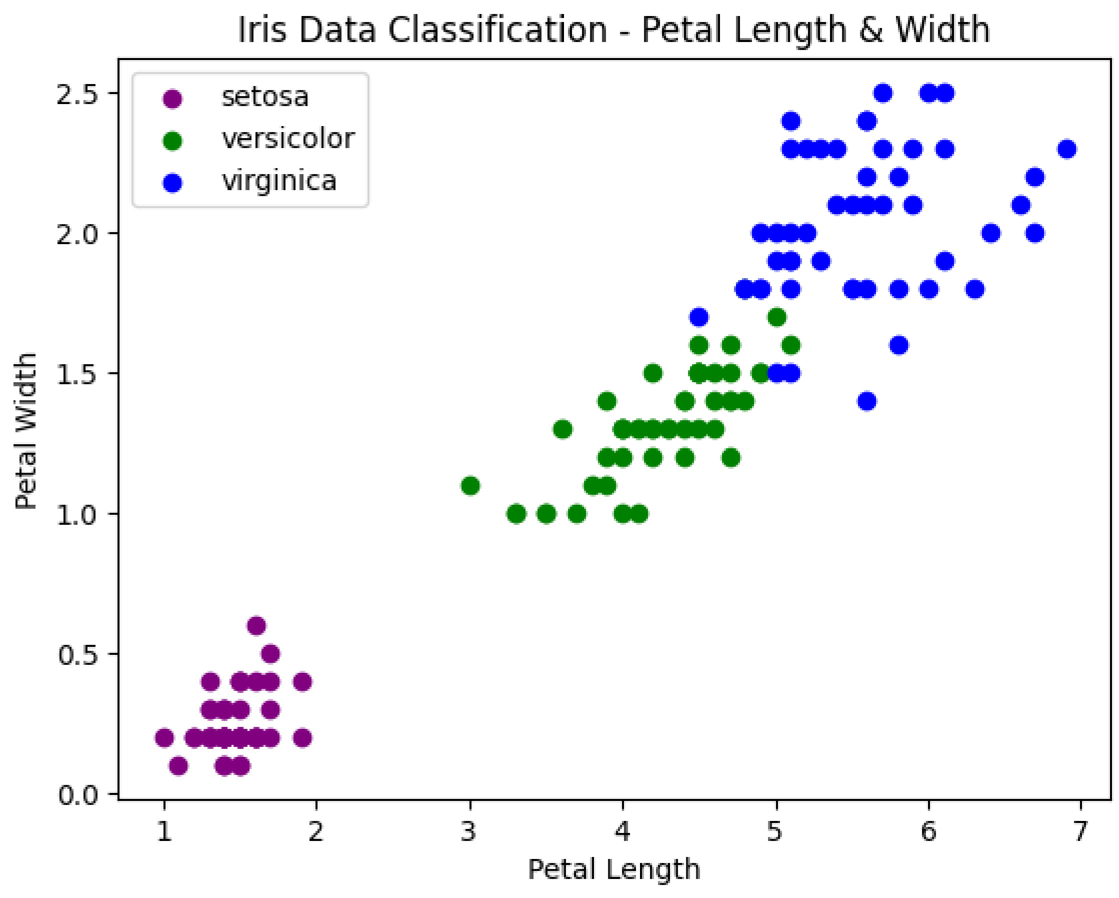


Figure 3. Iris Data Classification - Petal Length & Width

Model Building: KNN Algorithm

The K-nearest neighbor algorithm or KNN model is implemented on the Iris data, where the data values and target values are extracted, after which the data is *split into training and testing set.* The **train_test_split()** function is used along where one of its parameter is **'stratify'**, which helps in the uniform distribution of the data points in the train and test set.

Extracting the **independent variables (features)** into variable X and **dependent variable (target labels)** into variable y.

```
In [5]: X = iris_data.iloc[:, :-1].values   # selecting all features for training except the last column, i.e., the t
        y = iris_data.iloc[:, 4].values     # target variable
```

Splitting the dataset into train & test data

Splitting the dataset into training and testing data, where **70%** data is considered for *train dataset* and **30%** is considered for *testing*.

```
In [6]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state = 0, stratify=y)
```

Standardizing the training & testing data

```
In [7]: scaler = StandardScaler()
        scaler.fit(X_train)
        X_train = scaler.transform(X_train)
        X_test = scaler.transform(X_test)
```

Building the KNN model & fitting it to the training data

```
In [8]: knn_model = KNeighborsClassifier(n_neighbors=3)
        knn_model.fit(X_train, y_train)

Out[8]: ▾       KNeighborsClassifier
        KNeighborsClassifier(n_neighbors=3)
```

Here the value for k for the nearest neighbors is set to **3** (n_neighbors=3). This implies that the model will look for the 3 data points which are nearest to the new data point that needs to be classified.

Refered from [KNN Algorithm - Finding Nearest Neighbors. (n.d.) (https://www.tutorialspoint.com/machine_learning_with_python/knn_algorithm_finding_nearest_neighbors.htm)](https://www.tutorialspoint.com/machine_learning_with_python/knn_algorithm_finding_nearest_neighbors.htm)

Predicting the test data

The data points of the test dataset are predicted based on the training of the model and a class is assigned to each of the data point using the KNN model built.

In [67]:
```
y_pred = knn_model.predict(X_test)
```

In [68]:
```
for data, pdata, label in zip(X_test, y_pred, y_test):
    print(f"Data: {data}")
    print(f"Predicted label: {pdata}, True label: {label} \n")
```

```
Data: [0.50882419 0.79070506 1.02517078 1.57352382]
Predicted label: 2, True label: 2

Data: [-0.07443498 -0.75495509  0.74702367  0.91736814]
Predicted label: 2, True label: 2

Data: [-0.89099782  0.79070506 -1.25563553 -1.31356119]
Predicted label: 0, True label: 0

Data: [-0.89099782  1.67393943 -1.03311784 -1.05109892]
Predicted label: 0, True label: 0

Data: [1.32538703 0.34908787 0.52450598 0.26121245]
Predicted label: 1, True label: 1

Data: [-0.89099782  0.56989646 -1.14437668 -0.91986778]
Predicted label: 0, True label: 0

Data: [-0.42439048 -0.97576368  0.35761771 -0.00124982]
```

Table 4. Predicted Label & Actual Label of the test data

Evaluation metrics of predicted data

In [69]:
```
result1 = classification_report(y_test, y_pred)
print("Classification Report:\n")
print (result1)

result2 = accuracy_score(y_test,y_pred)
print("\nAccuracy:",result2)
```

```
Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00        15
           1       0.94      1.00      0.97        15
           2       1.00      0.93      0.97        15

    accuracy                           0.98        45
   macro avg       0.98      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45


Accuracy: 0.9777777777777777
```

Table 5. Classification Report

Based on the evaluation metrics, i.e., the **classification report** and the **accuracy of the model**, it is observed that the accuracy of the model is about **97.7%** and the score value for **precision and recall** also seems to be a good score, which indicates that the model is able to accurately classify the 30% of the test data into its *correct class labels*.

Confusion Matrix

```
In [70]: confusionmatrix_result = confusion_matrix(y_test, y_pred)
         sns.heatmap(confusionmatrix_result, annot=True, cmap="Blues", fmt="d", xticklabels=iris.target_names, ytickla
```

```
Out[70]: <Axes: >
```

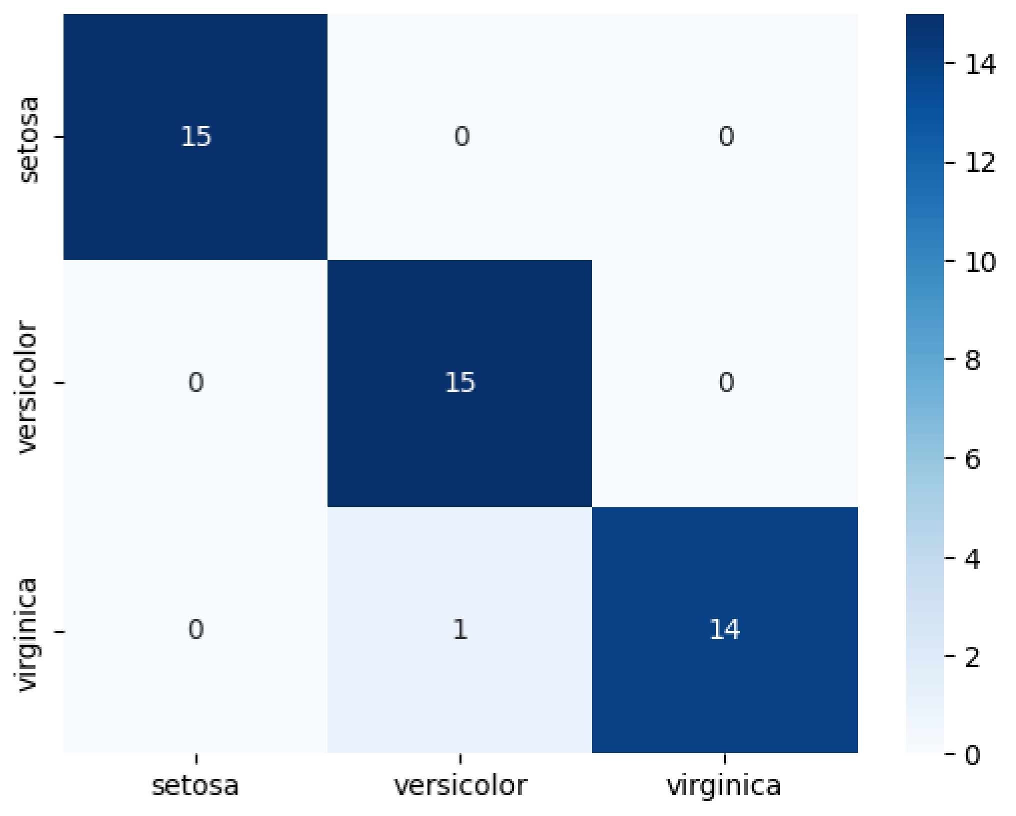


Figure 4. Confusion Matrix

The **confusion matrix** as shown above, helps in understanding the classification made by the model for the test dataset and how efficiently the labels are rightly classified. It basically summarizes the **performance** of the model and shows the number of *correct and incorrect predictions* that are made by the model.

As it is observed from the confusion matrix, all the data points from the test data for the two classes, **Setosa & Versicolor** are *correctly classified* into their respective classes, but only **14 are correctly classified** into **Virginica class**, which means that **1 of the instance is wrongly classified** into a different class. Here, the *1 instance of the Virginica class is classified as Versicolor*, which is the wrong prediction.

Thus, the confusion matrix helps in understanding which labels are correctly classified and that which are wrongly classified into a different class.

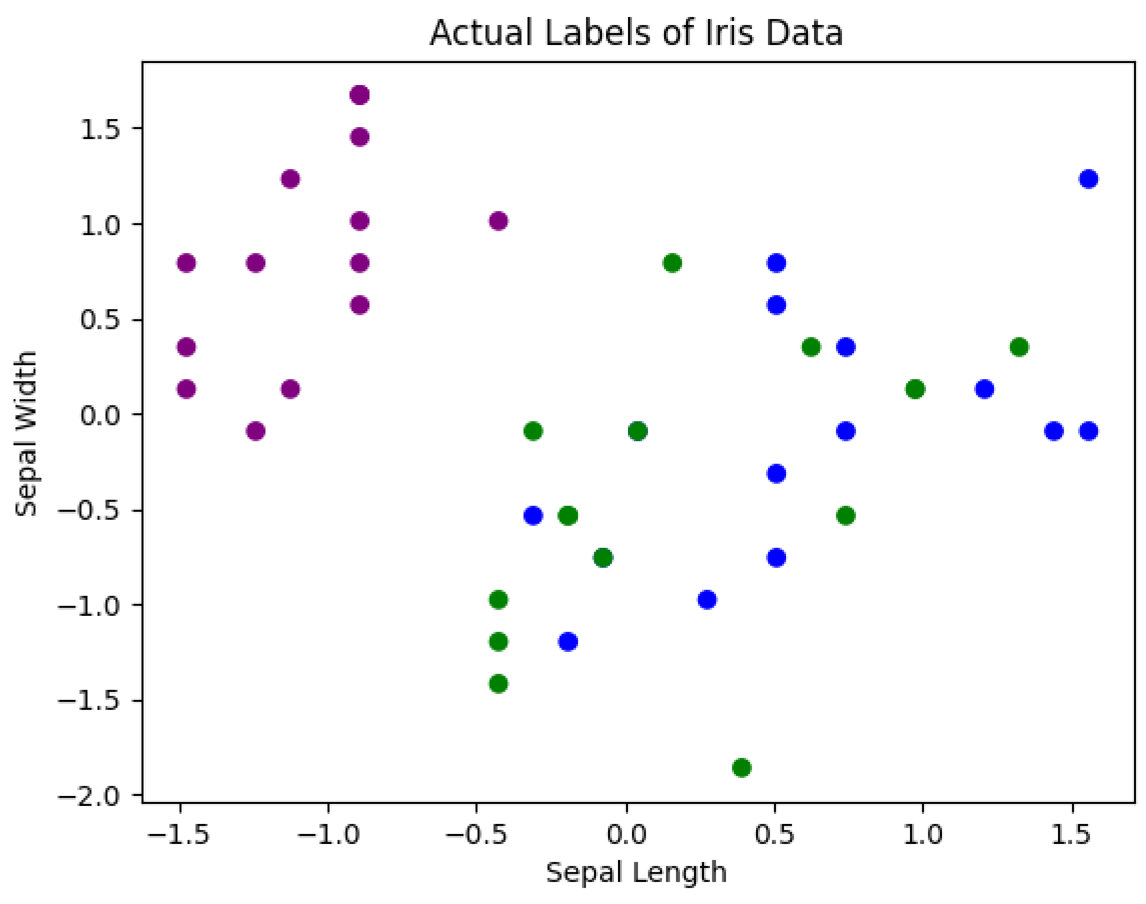
Output Data Visualization

```
In [71]: x = X_test[:, 0]      # Sepal Length
         y = X_test[:, 1]      # Sepal Width

         colors1 = ['purple' if label == 0 else 'green' if label == 1 else 'blue' for label in y_test]
         colors2 = ['purple' if label == 0 else 'green' if label == 1 else 'blue' for label in y_pred]

         plt.scatter(x, y, c=colors1)
         plt.title("Actual Labels of Iris Data")
         plt.xlabel('Sepal Length')
         plt.ylabel('Sepal Width')
         plt.show()


         plt.scatter(x, y, c=colors2)
         plt.title("Predicted Labels of Iris Data")
         plt.xlabel('Sepal Length')
         plt.ylabel('Sepal Width')
         plt.show()
```



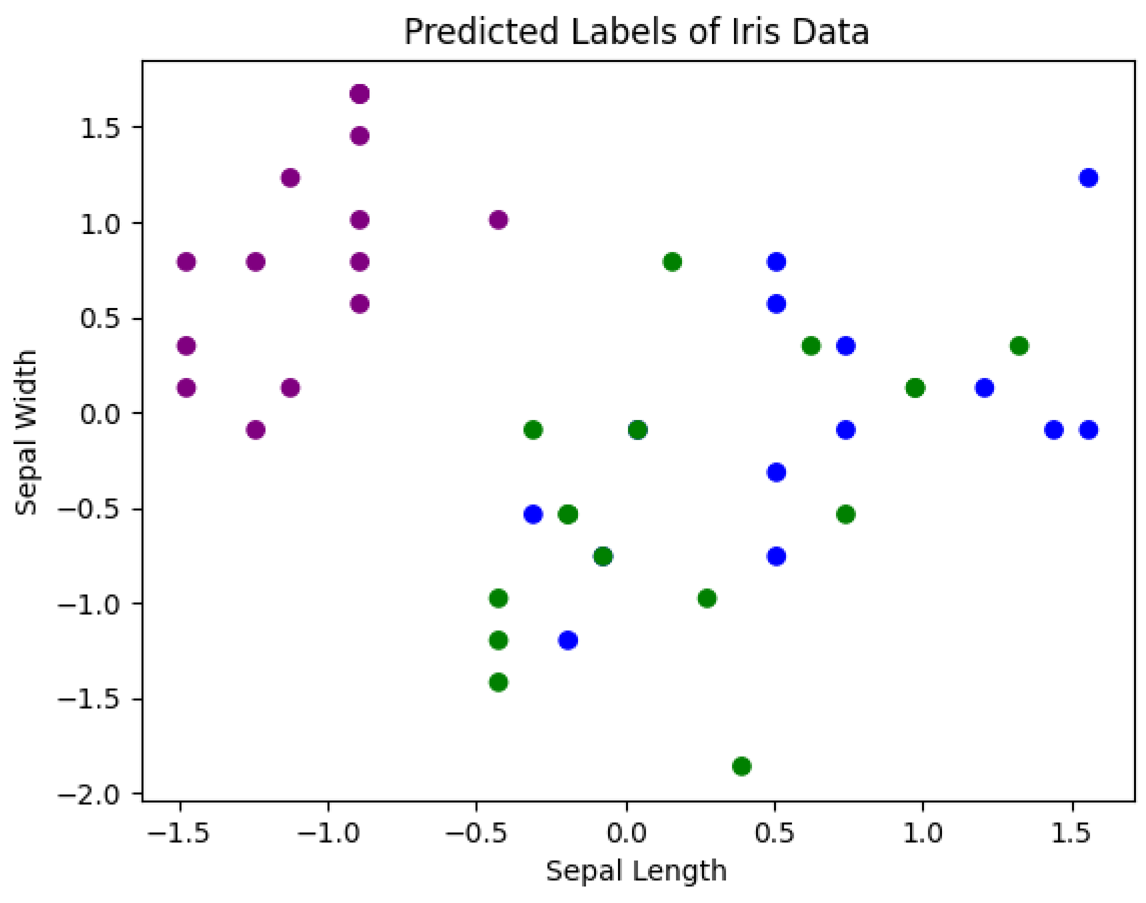


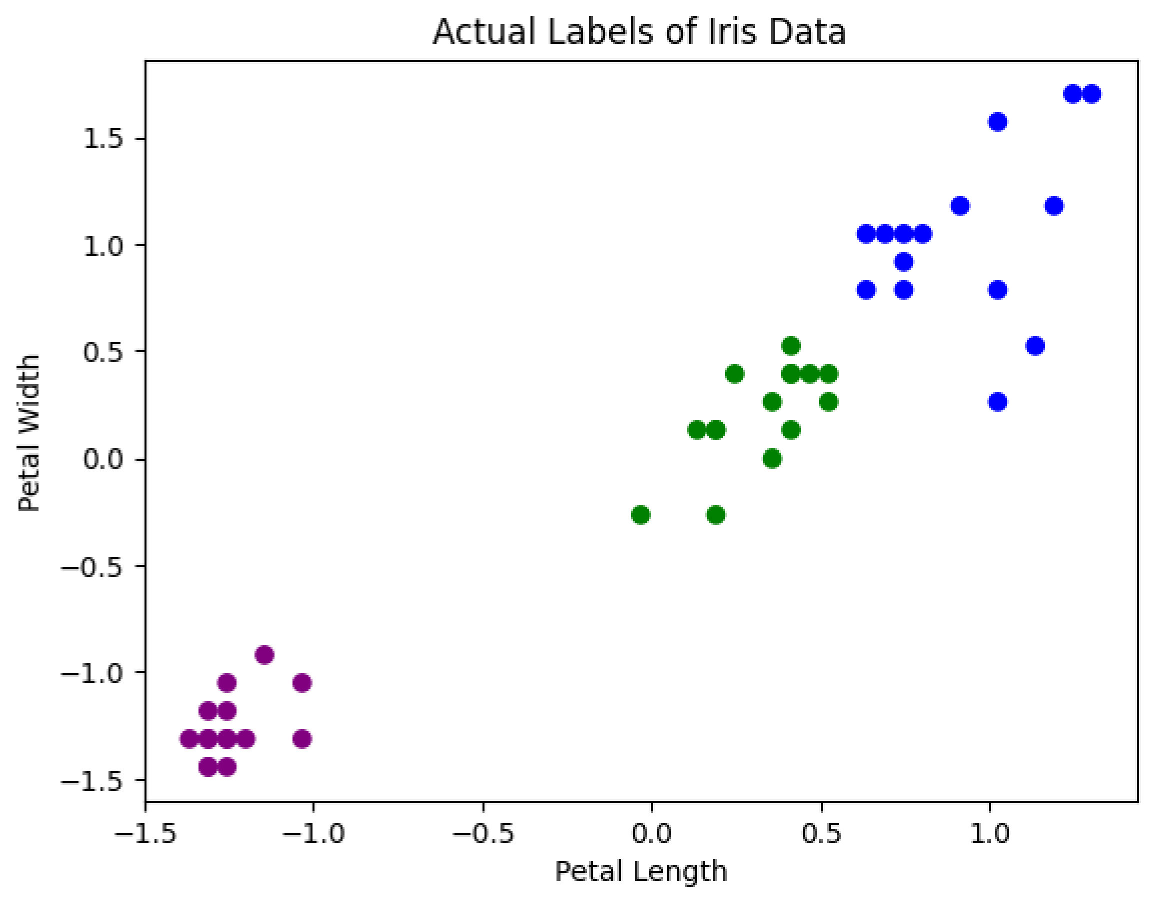
Figure 5. Actual & Predicted Labels for Sepal Length & Width

```
In [72]: x = X_test[:, 2]      # Petal Length
         y = X_test[:, 3]      # Petal Width

         colors1 = ['purple' if label == 0 else 'green' if label == 1 else 'blue' for label in y_test]
         colors2 = ['purple' if label == 0 else 'green' if label == 1 else 'blue' for label in y_pred]

         plt.scatter(x, y, c=colors1)
         plt.title("Actual Labels of Iris Data")
         plt.xlabel('Petal Length')
         plt.ylabel('Petal Width')
         plt.show()


         plt.scatter(x, y, c=colors2)
         plt.title("Predicted Labels of Iris Data")
         plt.xlabel('Petal Length')
         plt.ylabel('Petal Width')
         plt.show()
```



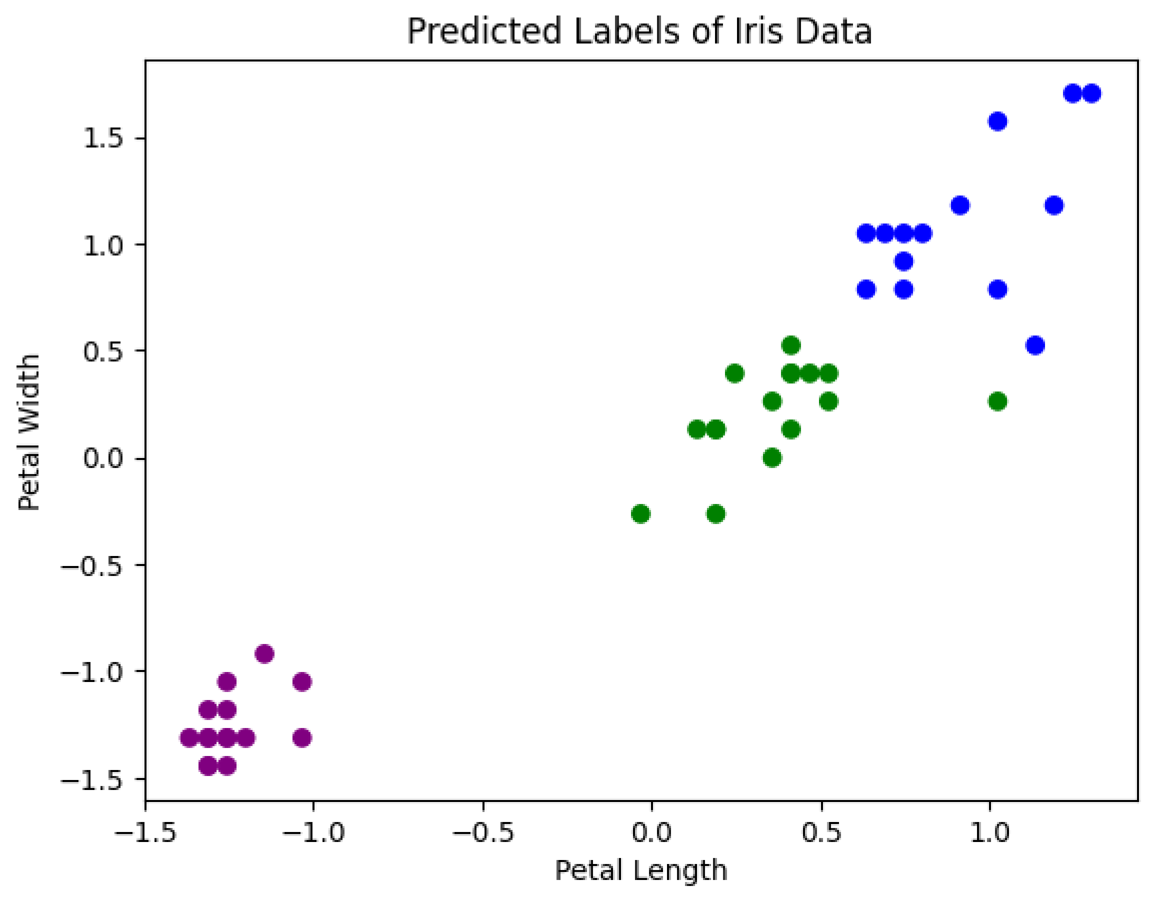


Figure 6. Actual & Predicted Labels for Petal Length & Width

Thus, the visualizations are created for the actual labels and the predicted labels based on the features of the dataset.

Results

Q1: What was the overall accuracy of the model?

A1. The overall accuracy of the model is 97.7%

Q2: What was the accuracy of each type of iris?

A2. The accuracy of each type of Iris is as follows.

> Accuracy for Setosa: **100%**

> Accuracy for Versicolor: **100%**

> Accuracy for Virginica: **93.3%**

In [73]:
```
for i in range(3):
    accuracy_iris = accuracy_score(y_test[y_test == i], y_pred[y_test == i])
    accuracy_percent = round(accuracy_iris * 100, 2)
    print(f"Accuracy for Iris, {iris.target_names[i]}: {accuracy_percent}")
```

```
Accuracy for Iris, setosa: 100.0
Accuracy for Iris, versicolor: 100.0
Accuracy for Iris, virginica: 93.33
```

Q3: Would you classify the model as a good model or not?

A3. Based on the evaluation metrics, it is observed that the model has achieved an **accuracy of 97%**, which classifies the model as an *efficient model*. But it is important to also consider the other parameters of evaluation metrics in order to understand the depth of performance, as sometimes the accuracy metric of the model could be misleading.

Thus, upon analysis based on the **classification report and confusion matrix**, we can conclude that the model is a **good model** as the precision and recall score is pretty good and also the values in the confusion matrix imply that only 1 class is incorrectly classified, which can be also be checked manually for better performance.

Overall, *I would classify the model as a good model*, for following two reasons.

1. Accuracy of the model is 97%
2. Classification Report and Confusion matrix imply good performance score of the model

Conclusion

KNN algorithms are simple and effective machine learning models that can be implemented on small datasets, for both classification or regression based problems.

The Iris data classification is implemented using the KNN model where it is concluded that the accuracy of the model obtained is 97.7% which indicates a good result. Also, the classification report and confusion matrix help in understanding the performance of the model indicating that the model has good precision and recall scores.

Thus, KNN algorithm can be a suitable method that can be used in order to classify the classes of the Iris data and predict the type of Iris flower based on the various features or parameters.

References

[1] Avuluri, V. S. R. (2021, December 10). K Nearest Neighbors and implementation on Iris data set. Medium. https://medium.com/@avulurivenkatasaireddy/k-nearest-neighbors-and-implementation-on-iris-data-set-f5817dd33711 (https://medium.com/@avulurivenkatasaireddy/k-nearest-neighbors-and-implementation-on-iris-data-set-f5817dd33711)

[2] KNN Algorithm - Finding Nearest Neighbors. (n.d.). https://www.tutorialspoint.com/machine_learning_with_python/knn_algorithm_finding_nearest_neighbors.htm (https://www.tutorialspoint.com/machine_learning_with_python/knn_algorithm_finding_nearest_neighbors.htm)

[3] Xuhewen. (2016). Iris Dataset Visualization and Machine Learning. Kaggle. https://www.kaggle.com/code/xuhewen/iris-dataset-visualization-and-machine-learning (https://www.kaggle.com/code/xuhewen/iris-dataset-visualization-and-machine-learning)

[4] seaborn.heatmap — seaborn 0.12.2 documentation. (n.d.). https://seaborn.pydata.org/generated/seaborn.heatmap.html (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

[5] What is the k-nearest neighbors algorithm? | IBM. (n.d.). https://www.ibm.com/topics/knn (https://www.ibm.com/topics/knn)