Pandas Tutorial for Beginners

Introduction to Pandas

Pandas is a powerful, fast, and flexible open-source data analysis and manipulation library for Python.

It is built on top of **NumPy** and is specifically designed to handle structured data (like tabular data).

With its easy-to-use data structures (i.e., DataFrame and Series), Pandas has become the go-to tool for data analysis tasks in scientific research, including medical data analysis.

Why Pandas?

- **Data Cleaning and Preparation:** Pandas provides versatile tools for cleaning, transforming, and restructuring data.
- **Data Exploration:** It simplifies operations like filtering, aggregating, and summarizing data.
- **Performance:** Pandas is optimized for fast and efficient data manipulation and analysis, even for large datasets.

Pandas in data analysis

Pandas is particularly useful in **medical data analysis** for tasks like:

- Handling patient information, clinical trial data, and diagnostic records.
- Conducting exploratory data analysis (EDA) to uncover trends in health data.
- Cleaning and transforming datasets to prepare them for statistical modeling and machine learning.

Some public pandas objects, functions and methods

Below is a comprehensive table of commonly used Pandas bjects, functions and methods. For more details you can always refer to the official pandas documentation

Function/Method	Description	Example Usage
`pd.read_csv()`	Reads a CSV file and	`df = pd.read_csv('iris.csv')`

Function/Method	Description	Example Usage
	loads it into a DataFrame	
`df.head()`	Displays the first few rows of the DataFrame	`df.head()`
`df.tail()`	Displays the last few rows of the DataFrame	`df.tail()`
`df.info()`	Provides information about DataFrame columns, data types, and missing values	`df.info()`
`df.describe()`	Generates summary statistics for numerical columns	`df.describe()`
`df.shape`	Returns the shape (number of rows and columns) of the DataFrame	`df.shape`
`df.isnull()`	Checks for missing values in each column	`df.isnull().sum()`
`df.dropna()`	Removes rows containing missing values	`df.dropna()`
`df.fillna()`	Fills missing values with a specified value or method	`df.fillna(0)`
`df.groupby()`	Groups data by a column and applies aggregation functions	`df.groupby('species').mean()`

Function/Method	Description	Example Usage
`df.sort_values()`	Sorts DataFrame by one or more columns	`df.sort_values('sepal_length')`
`df.merge()`	Merges two DataFrames based on common columns	`df.merge(df2, on='species')`
`df.pivot_table()`	Creates a pivot table summarizing data	`df.pivot_table(index='species', values='sepal_length', aggfunc='mean')`
`df.loc[]`	Accesses data by label-based indexing	`df.loc[0]`
`df.iloc[]`	Accesses data by integer-location based indexing	`df.iloc[0]`
`df.apply()`	Applies a function along the axis of the DataFrame	`df['sepal_length'].apply(lambda * 2)`
`df.corr()`	Computes correlation between numerical columns	`df.corr()`
`df.plot()`	Creates a plot for visualizing data	`df['sepal_length'].plot(kind='h`
`df.replace()`	Replaces values in the DataFrame with new values	`df.replace(5.1, 4.8)`
`df.astype()`	Converts the type of a column	`df['sepal_length'] = df['sepal_length'].astype(float)`
`df.duplicated()`	Identifies duplicate rows in the DataFrame	`df.duplicated()`
`df.drop_duplicates()`	Removes duplicate rows from	`df.drop_duplicates()`

Function/Method	Description	Example Usage
	the DataFrame	
`df.to_csv()`	Exports the DataFrame to a CSV file	`df.to_csv('output.csv')`
`df.index`	Retrieves the index (row labels) of the DataFrame	`df.index`
`df.columns`	Retrieves the column names of the DataFrame	`df.columns`
`df.describe(include='all')`	Generates summary statistics for both numerical and categorical columns	`df.describe(include='all')`
`df.sample()`	Randomly samples rows from the DataFrame	`df.sample(5)`
`df.pivot()`	Creates a pivot table with unique values for both rows and columns	`df.pivot(index='species', columns='sepal_length', values='sepal_width')`
`df.cumsum()`	Computes the cumulative sum of numeric columns	`df['sepal_length'].cumsum()`
`df.shift()`	Shifts data by a specified number of periods	`df['sepal_length'].shift(1)`
`df.applymap()`	Applies a function to every element in the DataFrame	`df.applymap(lambda x: x ** 2)`
`df.notnull()`	Checks for non-missing	`df.notnull()`

Function/Method	Description	Example Usage
	values in the DataFrame	
`df.merge_asof()`	Merges DataFrames based on nearest key rather than exact match, useful for time series	`df.merge_asof(df2, on='time')`
`df.resample()`	Resamples time series data, commonly used for medical data analysis in time series form	`df.resample('D').mean()`

Note Don't worry about memorizing every Pandas function right away. This tutorial is just an introduction. As we work with real data, we'll get more comfortable and remember the functions we use most often. And now a days with internet we are always free to refer to the offical documentation anytime. Happy learning!

Working with a few functions

We will discuss a few of the functions mentioned above at a beginner level. Later on, as needed, you may follow others in the same manner. Discussing all functions will be beyond the current scope.

1. Loading Data into Pandas

The first step in using Pandas is loading data into a DataFrame. The **Iris dataset** is used throughout this tutorial.

In [12]:
```
import pandas as pd
```

In [13]:
```
# Load the Iris dataset into a DataFrame
df = pd.read_csv('Iris.csv')  #make sure yoiu have Iris.csv in your worki
```

In [14]:
```
# Display the first few rows of the dataset
df.head()
```

Out[14]:

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

Explanation:

- `pd.read_csv()` loads the CSV file into a DataFrame.
- `df.head()` shows the first 5 rows of the dataset to give an overview.

Expected Output:

- The first 5 rows of the Iris dataset, which should include columns like `sepal_length`, `sepal_width`, `petal_length`, and `species`.

---

2. Exploring the Dataset

You can examine the dataset's structure and get basic statistics.

In [22]:
```
# Get the structure of the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [23]:
```
# Generate summary statistics for numerical columns(notice only numerical
df.describe()
```

Out[23]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

Explanation:

- `df.info()` provides information about the number of non-null entries and the data types of each column.
- `df.describe()` gives statistics like the mean, standard deviation, minimum, and maximum values for numerical columns.

---

3. Handling Missing Data

In real-world datasets, missing data is common. Pandas provides methods for detecting and handling missing data.

A. df.isnull()

In [17]:
```
# Check for missing values
df.isnull().sum()
```

Out[17]:

	0
Id	0
SepalLengthCm	0
SepalWidthCm	0
PetalLengthCm	0
PetalWidthCm	0
Species	0

dtype: int64

Explanation:

- `df.isnull().sum()` shows the number of missing values in each column of the DataFrame.

B. df.dropna()

In [18]:
```
# Drop rows with missing values
df_cleaned = df.dropna()
```

Explanation:

- `df.dropna()` removes rows containing missing data from the DataFrame. This can be useful when you want to discard rows with incomplete information.

C. df.fillna()

In many datasets, some values may be missing. One way to handle missing data is by filling it with the **mean** of the column. This method assumes the missing values are similar to the average value of that column.

In [20]:
```
# Calculate and replace missing values in 'SepalLengthCm' column with its
df_filled = df.fillna(df['SepalLengthCm'].mean())
```

Code Explanation: This is a complex code line; no worries if you couldn't understand. Here is the explanation:

1. **`df['SepalLengthCm'].mean()`**: Computes the mean (average) of the `SepalLengthCm` column.
2. **`df.fillna()`**: Fills missing (`NaN`) values in the DataFrame with the specified value (mean in this case).
3. **`df_filled`**: Stores the updated DataFrame with missing values replaced.

Why Use the Mean?

Filling with the mean assumes that missing values are similar to the average value of the column, which is common for numerical data without extreme outliers. You may do differently as well as per the problem.

---

4. Grouping Data

Grouping data is useful for applying aggregation functions like mean, sum, etc. Here's an example where we group by `species` and calculate the mean for each group.

In [25]:
```
# Group by species and calculate the mean of each numeric column
df.groupby('Species').mean()
```

Out[25]:

Species	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Iris-setosa	25.5	5.006	3.418	1.464	0.244
Iris-versicolor	75.5	5.936	2.770	4.260	1.326
Iris-virginica	125.5	6.588	2.974	5.552	2.026

Explanation:

- `df.groupby('species')` groups the data by the `species` column.
- `.mean()` computes the mean for each group in the dataset, giving insights into the central tendency of the data based on the species.

Expected Output:

- A new DataFrame showing the mean of each numeric column for each species of the Iris flower.

---

5. Sorting Data

You can sort data by one or more columns to organize the dataset.

In [29]:
```
# Sort the DataFrame by SepalLengthCm
df_sorted = df.sort_values('SepalLengthCm')
df_sorted
```

Out[29]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
13	14	4.3	3.0	1.1	0.1	Iris-setosa
42	43	4.4	3.2	1.3	0.2	Iris-setosa
38	39	4.4	3.0	1.3	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
41	42	4.5	2.3	1.3	0.3	Iris-setosa
...	...	...	...	...	...	...
122	123	7.7	2.8	6.7	2.0	Iris-virginica
118	119	7.7	2.6	6.9	2.3	Iris-virginica
117	118	7.7	3.8	6.7	2.2	Iris-virginica
135	136	7.7	3.0	6.1	2.3	Iris-virginica
131	132	7.9	3.8	6.4	2.0	Iris-virginica

150 rows × 6 columns

Explanation:

- `df.sort_values('SepalLengthCm')` sorts the DataFrame by the `SepalLengthCm` column in ascending order. Sorting helps in identifying patterns and outliers in the data.

Ploting with pandas

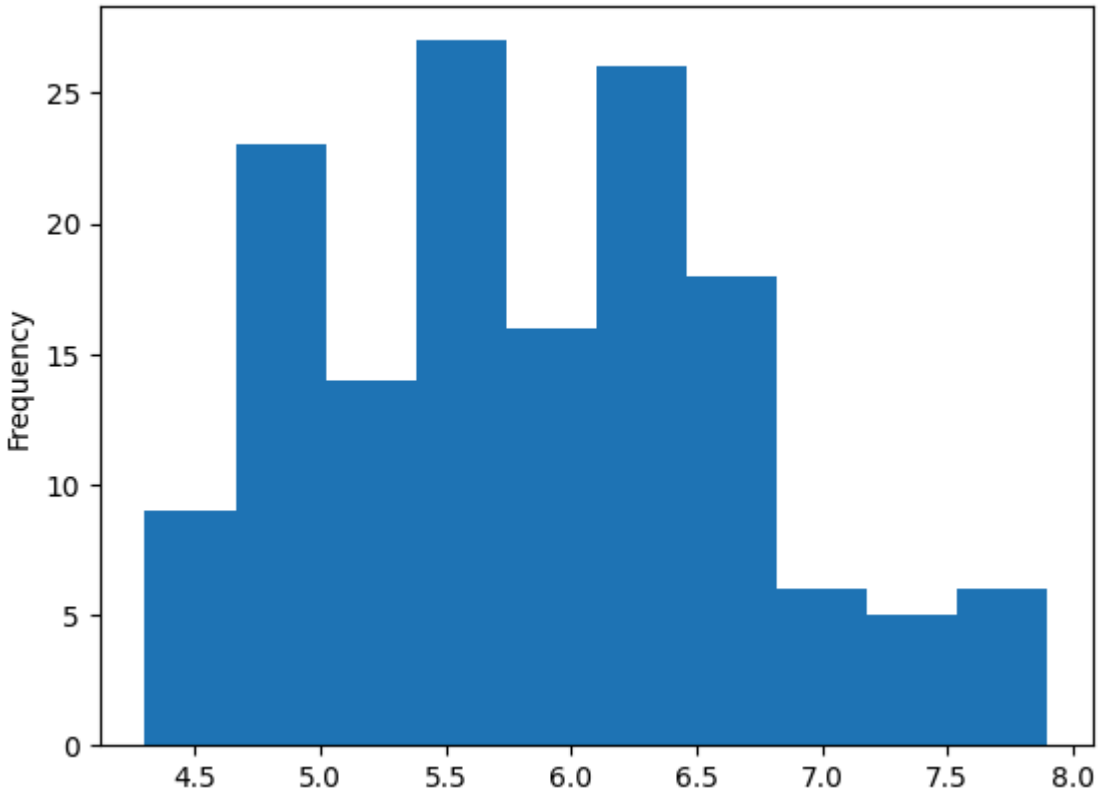
We will learn a few ways by which we can do plotting with pandas

6. Histogram with Pandas

Pandas integrates with **Matplotlib** to create visualizations. Here's an example of creating a histogram of the `sepal_length` column.

In [32]:
```
# Plotting a histogram of the sepal_length column
df['SepalLengthCm'].plot(kind='hist')
```

Out[32]:
```
<Axes: ylabel='Frequency'>
```



Explanation:

- `kind='hist'` creates a histogram.
- `df['SepalLengthCm']` specifies the column to plot.
- By default, the number of bins is automatically chosen.
- `df['sepal_length'].plot(kind='hist')` creates a histogram to visualize the distribution of the `sepal_length` data. Visualization is a crucial step in data analysis to understand the patterns and trends in the data.

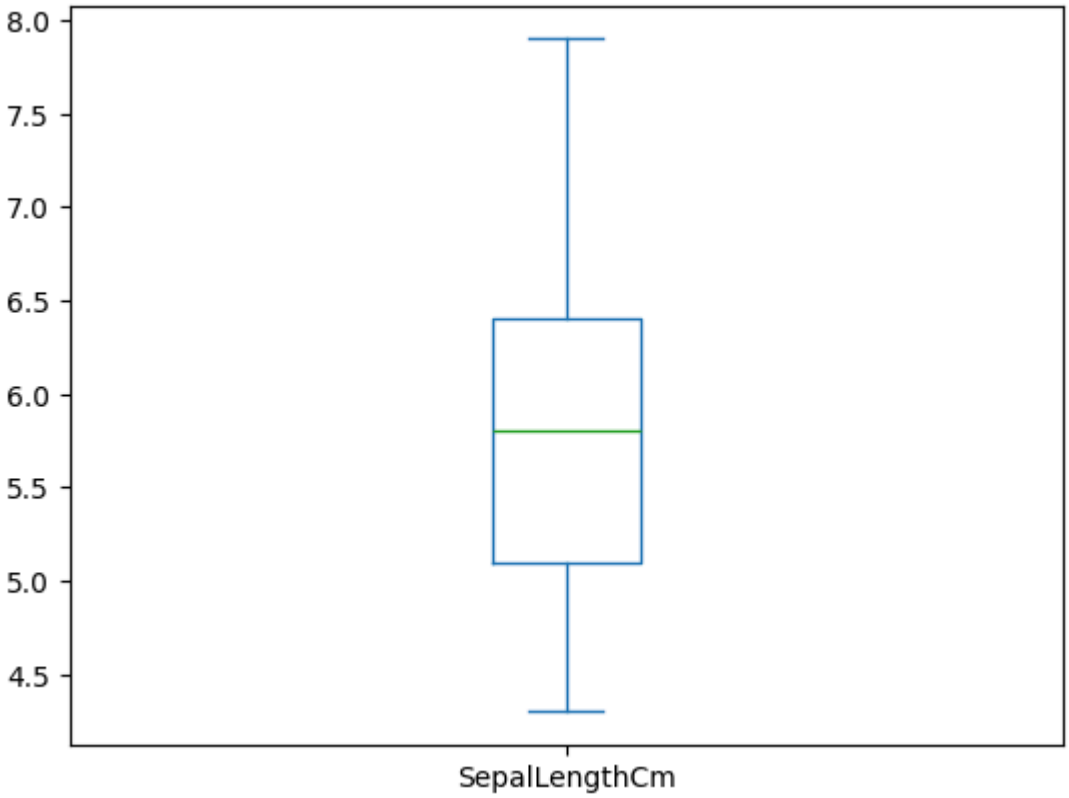
---

7. Plotting a Box Plot

Box plots are useful for identifying the distribution, spread, and outliers of a dataset.

In [33]:
```
# Plotting a box plot of the SepalLengthCm column
df['SepalLengthCm'].plot(kind='box')
```

Out[33]: <Axes: >



Explanation:

- `kind='box'` creates a box plot.
- The box plot shows the median, quartiles, and any outliers in the data.

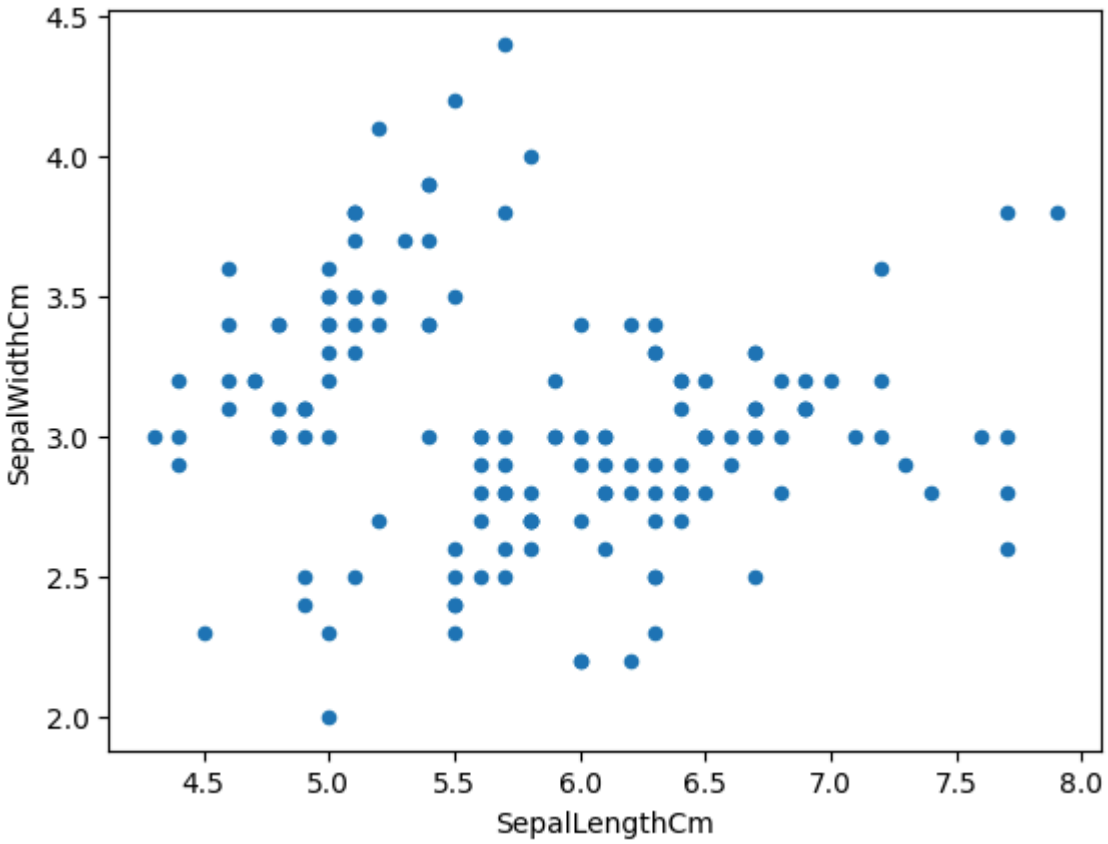
---

8. Plotting a Scatter Plot

Scatter plots help visualize relationships between two continuous variables.

In [34]:
```
# Plotting a scatter plot between SepalLengthCm and SepalWidthCm
df.plot(kind='scatter', x='SepalLengthCm', y='SepalWidthCm')
```

Out[34]:
```
<Axes: xlabel='SepalLengthCm', ylabel='SepalWidthCm'>
```



Explanation:

- `kind='scatter'` creates a scatter plot.
- `x='SepalLengthCm'` and `y='SepalWidthCm'` define the variables for the x and y axes.
- This plot helps identify trends or correlations between the two columns.

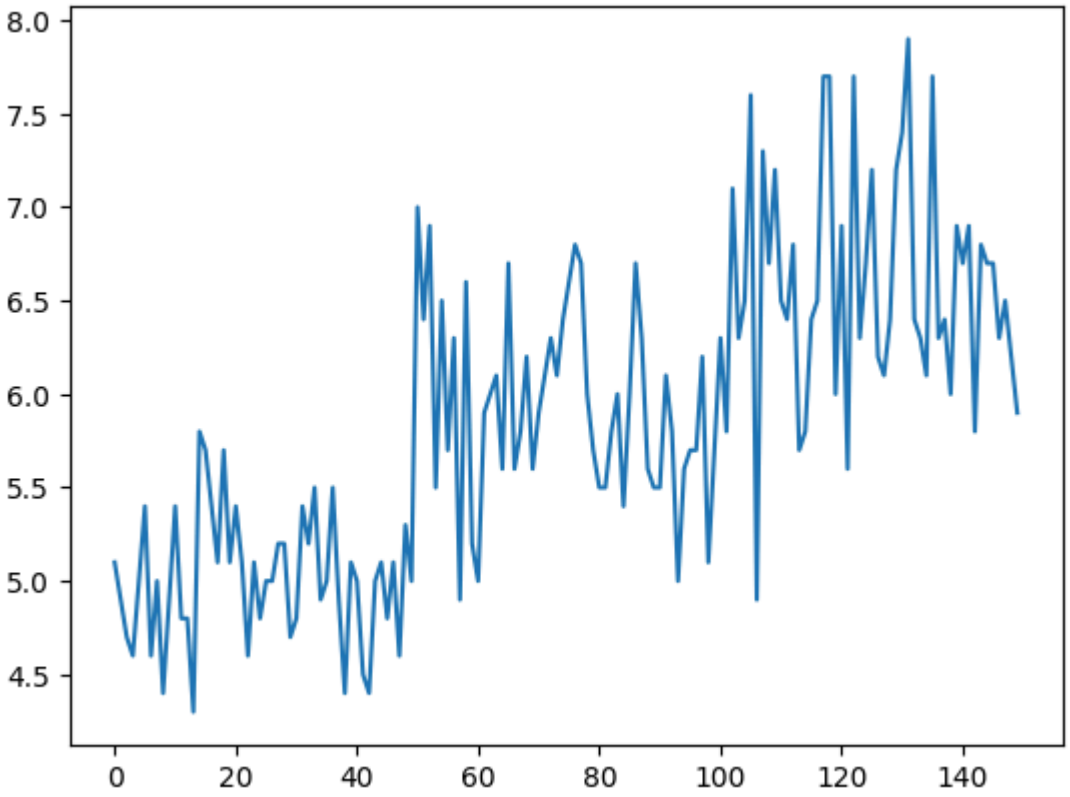
---

9. Plotting a Line Plot

Line plots are useful for displaying continuous data over time or another ordered variable.

In [63]:
```
# Plotting a line plot of SepalLengthCm
df['SepalLengthCm'].plot(kind='line')
```

Out[63]: <Axes: >



Explanation:

- `kind='line'` creates a line plot.
- The plot shows the trend of `SepalLengthCm` over the rows in the dataset.

These are some common types of plots in Pandas that can be used to visualize various aspects of data, helping with exploratory data analysis (EDA).

---

10. Replacing Values in the DataFrame

You can replace specific values in the dataset using the `replace()` method.

In [62]:
```
# Replace specific values in the DataFrame
df.replace(5.1, 4.8)
```

Out[62]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	4.8	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	4.8	1.8	Iris-virginica

150 rows × 7 columns

Explanation:

- `df.replace(5.1, 4.8)` replaces all occurrences of `5.1` with `4.8` in the DataFrame. This is useful for data correction or standardizing values.

Advance Functions in Pandas

11.Combining DataFrames

You can concatenate multiple DataFrames along a particular axis (rows or columns). This is useful when working with large datasets or combining results from different data sources.

In [38]:
```
# Concatenate two DataFrames vertically (along rows)
df_combined = pd.concat([df, df], axis=0)
df_combined.head()
```

Out[38]:

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

In [ ]:
```
# Concatenate two DataFrames horizontally (along columns)
df_combined = pd.concat([df, df], axis=1)
df_combined.head()
```

Explanation: `pd.concat([df, df2], axis=0)` concatenates two DataFrames `(df and df2)` vertically `(along rows)`. The axis=0 argument specifies that the concatenation should occur along the rows.

12. Correlation Between Variables

To understand the relationships between variables in the dataset, we can calculate the correlation matrix. Correlation helps in identifying how strongly two variables are related, which is important in predictive modeling.

In [60]:
```
# Compute correlation between numerical columns

#step1: first make List of numerical columns to check correlation
list_of_numerical_columns = ['SepalLengthCm',   'SepalWidthCm', 'PetalLen
```

In [61]:
```
#Step2:  Compute and display the correlation matrix between numerical col
df[list_of_numerical_columns].corr()
```

Out[61]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

Explanation:

- `list_of_numerical_columns`: This is the list of columns to calculate the correlation for.
- `.corr()`: This method computes the pairwise correlation between the selected numerical columns in the DataFrame.

- The result is a correlation matrix showing how each numerical column relates to others, with values ranging from `-1` (strong negative correlation) to `1` (strong positive correlation).

---

13. Accessing Data by Label or Integer Location

Pandas allows you to access data using **labels** (`loc[]`) or **integer positions** (`iloc[]`).

In [40]:
```
# Access the first row using label-based indexing
df.loc[0]
```

Out[40]:

	0
Id	1
SepalLengthCm	5.1
SepalWidthCm	3.5
PetalLengthCm	1.4
PetalWidthCm	0.2
Species	Iris-setosa

dtype: object

Explanation:

- `df.loc[0]` retrieves the row with the label `0` from the DataFrame. This is useful when working with labeled indices.

In [46]:
```
# Access the first row using integer-location based indexing
df.iloc[0]
```

Out[46]:

	0
Id	1
SepalLengthCm	5.1
SepalWidthCm	3.5
PetalLengthCm	1.4
PetalWidthCm	0.2
Species	Iris-setosa

dtype: object

Explanation:

- `df.iloc[0]` retrieves the first row by integer position (0-based indexing). This is useful when working with DataFrames that don't have labeled indices.

---

14. Filtering Data

You can filter the rows in a DataFrame based on conditions.

In [48]:
```
# Filter rows where sepal_length is greater than 5.0
df_filtered = df[df['SepalLengthCm'] > 5.0]
```

Explanation:

- `df[df['SepalLengthCm'] > 5.0]` filters the DataFrame to include only the rows where the value in the `SepalLengthCm` column is greater than 5.0. This is a basic form of data selection and is very useful in exploratory data analysis (EDA).

---

15. Normalizing Data

Normalization is the process of scaling numeric columns to a specific range, often between 0 and 1. This is particularly useful when preparing data for machine learning.

In [55]:
```
# Normalize the 'sepal_length' column to a range of 0 to 1
df['sepal_length_normalized'] = (df['SepalLengthCm'] - df['SepalLengthCm'
```

In [59]:
```
# Print 'SepalLengthCm' and 'sepal_length_normalized' columns side by sid
print(df[['SepalLengthCm', 'sepal_length_normalized']])
```

```
     SepalLengthCm  sepal_length_normalized
0              5.1                 0.222222
1              4.9                 0.166667
2              4.7                 0.111111
3              4.6                 0.083333
4              5.0                 0.194444
..             ...                      ...
145            6.7                 0.666667
146            6.3                 0.555556
147            6.5                 0.611111
148            6.2                 0.527778
149            5.9                 0.444444

[150 rows x 2 columns]
```

Explanation:

- `df[['SepalLengthCm', 'sepal_length_normalized']]` : Selects both the SepalLengthCm and sepal_length_normalized columns from the DataFrame.

and

- `df['sepal_length_normalized'] = (df['sepal_length'] - df['sepal_length'].min()) / (df['sepal_length'].max() -`

`df['sepal_length'].min())` -> scales the sepal_length column to a range between 0 and 1.

This technique helps when comparing features with different scales, especially in machine learning models.

Interactive Exercises and Summary

Quiz

1. What function is used to load a CSV file in Pandas?
   - a) `read_csv()`
   - b) `load_csv()`
   - c) `import_csv()`
2. What is the purpose of `dropna()`?
   - a) To drop columns
   - b) To remove missing values
   - c) To sort data

   Correct Answer:

   A. a) `read_csv()`
   B. b) To remove missing values *italicised text*

Great Job! You've Made Tremendous Progress!

- You've learned key **Pandas** functions for data manipulation, essential in scientific and medical research.
- This tutorial covered everything from basic operations to more advanced techniques like plotting, filtering, normalizing, and calculating correlations.
- Keep practicing with real-world datasets to solidify your understanding.
- Explore the official Pandas Documentation to deepen your knowledge and tackle more complex data analysis tasks.

```
Congratulations!                                    You've
completed the tutorial.                                   Happy
learning!
```

Project 1: Classification of Iris Flowers

- **Input:** Iris.csv data set
- **Project:** Building different classification models, validation and performance evaluation of models

---

Step 1: Import all necessary libraries

The following libraries to be imported in this project:

- pandas: Used to read and manipulate CSV data.
- Numpy: For fast and efficient processing of data
- sklearn.dataset: To load data from the Sci-Kit-Learn repository
- sklearn.train_test_split: From scikit-learn, used to split data into training and testing sets.
- sklearn.preprocessing: For feature scaling/normalization
- sklearn.LogisticRegression: A common classification algorithm from scikit-learn.
- sklearn.SVC: Support Vector Machine Classifier
- sklearn.RandomeForest: Random Forest Classification
- sklearn.KNeighborsClassifier: k-Nearest Neighbour classifier
- sklearn.DecissionTreeClassifier: Decision Tree Classifier
- sklearn.MLPClassifier: Multi-Layer Perceptron classifier
- sklearn.GradientBoostingClassifier: Gradient Boosting classifier
- sklearn.accuracy_score: To calculate model accuracy.

In [9]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris

# Step 1: Load the Iris dataset
iris = load_iris()
data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
data['target'] = iris.target

# Download the data from "Iris.csv"   locally
X, y = iris.data, iris.target

# Convert to DataFrame for better processing
df = pd.DataFrame(data=X, columns=iris.feature_names)
df['target'] = y

# Preview the dataset: It si required as a customary step!
#print("Top 5 rows of the dataset:")
#print(df.head())
#print("Bottom 5 rows of the dataset:")
#print(df.tail())
```

```
#print("The columns present in the data frame
#print(df.columns)
#print("The information about the attributes
print(df.info())
#print("To check if the null entries are there")
#print(df.isnull())
#print("The statistical information about the data")
# print(df.describe())
```

```
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, 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])
<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
```

Step 2: Split the data set into two parts: "Training set" and "Test set"

The following library is used

- import train_test_split from sklearn.model_selection
- "Training set" is used to train a model and "Test set" is used to test a model

In [8]:
```
from sklearn.model_selection import train_test_split
print("Import of \"Train-Test-Split-Selection\" library is successful")

# Split the dataset into training and testing sets: 67% for training and
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
# Note 1: Data (i.e., Data-attributes and Target-column are kept as separ
# Note 2: Here, random_state=42 is chosen as a seed value and popularly i

print("\nTrain and test data shapes:")
print("X_train:", X_train.shape, "X_test:", X_test.shape)
```

```
Import of "Train-Test-Split-Selection" library is successful

Train and test data shapes:
X_train: (100, 4) X_test: (50, 4)
```

Step 3: Preprocessing

The preprocessing task includes

- (a)Handling null-entries, if applicable
- (b) Scaling (to put all values in a normalize scale

- For scaling there are many methods: StandardScalar, MinMaxScalar, Normalizer, PolynomialFeatures, etc. Use any one.**

In [57]:
```
### Tutorial to learn the basic of scalar-based normolization
# Create a DataFrame
data1 = {'A': [2, 4, 5, 6, 7, 8, 9], 'B': [60, 70, 90, 10, 30, 40, 50]}
data2 = {'A':[1, 6, 3], 'B':[80, 40, 20]}
X_train_ = pd.DataFrame(data1)
X_test_ = pd.DataFrame(data2)

# Create a StandardScaler object
scaler = StandardScaler()

# Fit and transform the training data
X_train_scaled_ = scaler.fit_transform(X_train_)
print("Normalized training data set...\n")
display(X_train_scaled_)

# Transform the training data; it uses the parameters already leraned by
X_test_scaled_ = scaler.fit_transform(X_test_)
print("\n Normalized testing data set...\n")
display(X_test_scaled_)
```

```
Normalized training data set...

array([[-1.72849788,  0.40824829],
       [-0.83223972,  0.81649658],
       [-0.38411064,  1.63299316],
       [ 0.06401844, -1.63299316],
       [ 0.51214752, -0.81649658],
       [ 0.9602766 , -0.40824829],
       [ 1.40840568,  0.        ]])
 Normalized testing data set...

array([[-1.13554995,  1.33630621],
       [ 1.29777137, -0.26726124],
       [-0.16222142, -1.06904497]])
```

In [10]:
```
# Handling missing values: There are no missing values

# Normalization of training and testing data
'''
 Note: For normalization, sklearn provides two methods: fit_transform() a
       fit_transform() is applied to training data, whereas transform() i
       fit_transform() is a combination of fit() (to calculate the necess
       transformation based on the training data, such as, min, max, mean
       transform() applies the transformation to the data using the param

       The two methods applicable to all normalization methods defined in
'''
# Import scaling methods for normalization
from sklearn.preprocessing import StandardScaler

'''
```

```
# Import other normalization methods and use them, if necessary
#from sklearn.preprocessing import MinMaxScaler
#from sklearn.preprocessing import Normalizer
#from sklearn.preprocessing import PolynomialFeatures
'''

# Let's use the standard scaling in this project
scaler = StandardScaler()                              # Let StandardScalar(
X_train_scaled = scaler.fit_transform(X_train)         # Apply fit_transform
X_test_scaled = scaler.transform(X_test)               # Apply transform() t
print("Standard Scaled Data (First 5 rows):\n", X_train_scaled[:5])  # Sh

'''
# Min-Max scaling
minmax_scaler = MinMaxScaler()    # Let MinMaxScalar() be mimax_scalar
X_train_minmax = minmax_scaler.fit_transform(X_train)      # Apply fit_tr
X_test_minmax = minmax_scaler.transform(X_test)            # Apply transf
print("\nMin-Max Scaled Data (First 5 rows):\n", X_train_minmax[:5])

# Normalization
normalizer = Normalizer()        # Let Normalizer() be normalizer
X_train_normalized = normalizer.fit_transform(X_train)
X_test_normalized = normalizer.transform(X_test)
print("\nNormalized Data (First 5 rows):\n", X_train_normalized[:5])

# Polynomial-features scaling
poly = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
print("\nPolynomial Features (First 5 rows):\n", X_train_poly[:5])
'''
```

```
Standard Scaled Data (First 5 rows):
 [[-0.13835603 -0.26550845  0.22229072  0.10894943]
 [ 2.14752625 -0.02631165  1.61160773  1.18499319]
 [-0.25866563 -0.02631165  0.39595535  0.37796037]
 [-0.8602136   1.16967238 -1.39857913 -1.37061074]
 [ 2.26783585 -0.50470526  1.66949594  1.05048772]]
```

Out[10]:
```
'\n# Min-Max scaling\nminmax_scaler = MinMaxScaler()    # Let MinMaxScal
ar() be mimax_scalar\nX_train_minmax = minmax_scaler.fit_transform(X_tra
in)      # Apply fit_transform() to training data set\nX_test_minmax = m
inmax_scaler.transform(X_test)            # Apply transform() to testing
data set\nprint("\nMin-Max Scaled Data (First 5 rows):\n", X_train_minma
x[:5])      # Show top 5 training data\n\n# Normalization\nnormalizer =
Normalizer()        # Let Normalizer() be normalizer\nX_train_normalized
= normalizer.fit_transform(X_train)\nX_test_normalized = normalizer.tran
sform(X_test)\nprint("\nNormalized Data (First 5 rows):\n", X_train_norm
alized[:5])\n\n# Polynomial-features scaling\npoly = PolynomialFeatures
(degree=2, include_bias=False)\nX_train_poly = poly.fit_transform(X_trai
n)\nX_test_poly = poly.transform(X_test)\nprint("\nPolynomial Features
(First 5 rows):\n", X_train_poly[:5])\n'
```

Step 4: Dimensionality reduction

There are several methods defined in sklearn:

- PCA (Principal Component Analysis), IDA (Independent Component Analysis), LDA (Linear Discrimnant Analysis), NMF (Non-negative Matrix Factorization, SVD

(Singular Value Decomposition), etc. are a few popular dimensionality reduction techniques
- This project follows PCA
- **Note:** Dimnsionality reduction method is optional and does not necessarily yield good results.

In [70]:
```
# Using PCA for dimensionality reduction

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Display PCA results
print("\nPCA Reduced Training data shape (2 components):", X_train_pca.sh
print("\nPCA Reduced Testing data shape (2 components):", X_test_pca.shap
```

```
PCA Reduced Training data shape (2 components): (100, 2)

PCA Reduced Testing data shape (2 components): (50, 2)
```

Step 5: Building Classification Models

There are several ML algorithms that can be followed to build classification models. In this project, we shall follow the following ML algorithms followed by the performance evaluation of each.

- Support Vector Machine (SVM) classifier
- Random Forest classifier
- Decision Tree classifier
- Logistic Regression classifier
- XGBoost classifier
- Gradient boosting classifier

SVM Classifier

- Building model with SVM classifier

In [14]:
```
# Import SVM from sklearn package
from sklearn.svm import SVC  # Import Support Vector Machine (SVM) classi

# Support Vector Classifier
svm_model = SVC()          # Initialize the  classification method

svm_model.fit(X_train_scaled, y_train)      # Fit the model with scalar-n
svm_predictions = svm_model.predict(X_test_scaled)     # Get the predicti
y_pred = svm_predictions                               # Predicted result
print("\nSVM Predictions (First 10):", y_pred[:10])

#Note: We didn't use the result of dimensionality reduction in this proje
#      The result may be different if the data after dimensionality reduc
```

```
SVM Predictions (First 10): [1 0 2 1 1 0 1 2 1 1]
```

Evaluation of the performance of SVM classifier

- Evaluation with the **simple validation method**

In [ ]:
```
# Import evaluation metrics
from sklearn.metrics import confusion_matrix, accuracy_score, precision_s
from sklearn.metrics import roc_auc_score, classification_report

import seaborn as sns                    # It is a Python data visualiz
import matplotlib.pyplot as plt          #For graph-plotting
print("Import of packages for performance evaluation is successful\n")

#Define how to plot a confusion matrix with the result of validation
# Function to plot the confusion matrix: We shall use the same method in
def plot_confusion_matrix(y_true, y_pred, title):
    conf_matrix = confusion_matrix(y_true, y_pred)          # y_test is
    plt.figure(figsize=(5, 4))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabe
    plt.xlabel('Prediction labels')
    plt.ylabel('True lables')
    plt.title(title)
    plt.show()

plot_confusion_matrix(y_test, y_pred, "SVM Confusion Matrix")
```

In [1]:
```
# Import evaluation metrics
from sklearn.metrics import confusion_matrix, accuracy_score, precision_s
from sklearn.metrics import roc_auc_score, classification_report

# Now, let's get the result of SVM evaluation
#Accuracy:               (TP+TN)/(TP+TN+FP+FN)
svm_accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", svm_accuracy)

# Precision: TP/(TP+FP): The ratio of TP to the total predicted positives
svm_precision = precision_score(y_test, y_pred, average="weighted")
print("\nPrecision: ", svm_precision)

# Recall: TP/(TP+FN): Tha ratio of true positives to the total actual pot
svm_recall = recall_score(y_test, y_pred, average="weighted")
print("\nReacll : ", svm_recall)

#F1-score: Harmonic mean of `Precision` and `Recall'
svm_f1 = f1_score(y_test, y_pred, average="weighted")
print("\nF1 score: ", svm_f1)

# Specificity calculation
cm = confusion_matrix(y_test, y_pred)
specificity = []
for i in range(len(cm)):
    tn = np.sum(cm) - np.sum(cm[i, :]) - np.sum(cm[:, i]) + cm[i, i]
    fp = np.sum(cm[:, i]) - cm[i, i]
    specificity.append(tn / (tn + fp))
    svm_specificity = np.mean(specificity)
print("\nSpecificity: ", svm_specificity)

# Report the summary of all evaluation:
```

```
print("\nSVM Classification Report:", classification_report(y_test, y_pre
print("Classification with SVM is done!")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call las
t)
Cell In[1], line 7
      3 from sklearn.metrics import roc_auc_score, classification_report
      5 # Now, let's get the result of SVM evaluation
      6 #Accuracy:              (TP+TN)/(TP+TN+FP+FN)
----> 7 svm_accuracy = accuracy_score(y_test, y_pred)
      8 print("\nAccuracy:", svm_accuracy)
     10 # Precision: TP/(TP+FP): The ratio of TP to the total predicted po
sitives

NameError: name 'y_test' is not defined
```

Random Forest classifier

- Building model with Random Forest classifier

In [39]:
```
# Import Random Forest from sklean package
from sklearn.ensemble import RandomForestClassifier

# Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)          # Initialize

rf_model.fit(X_train_scaled, y_train)      # Fit the model with scalar-no
rf_predictions = svm_model.predict(X_test_scaled)     # Get the predictio
y_pred = rf_predictions                               # Predicted result
print("\nRandom Forest Predictions (First 10):", y_pred[:10])
```

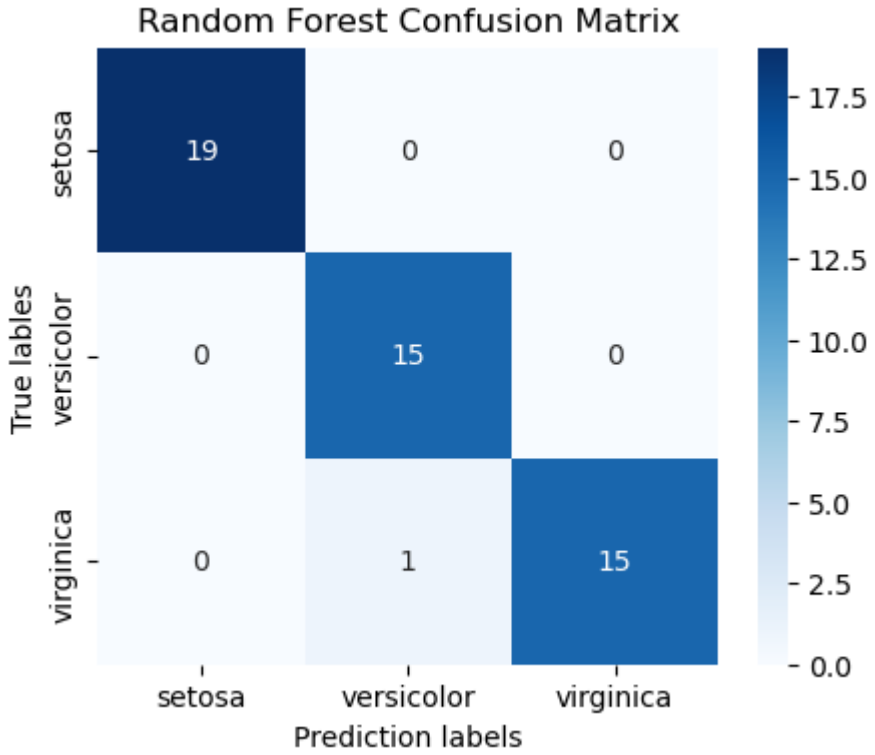
```
Random Forest Predictions (First 10): [1 0 2 1 1 0 1 2 1 1]
```

Evaluation of the performance of Random Forest classifier

- Evaluation with the **simple validation method**

In [42]:
```
plot_confusion_matrix(y_test, y_pred, "Random Forest Confusion Matrix")

# Now, let's get the result of RF evaluation
#Accuracy:              (TP+TN)/(TP+TN+FP+FN)
rf_accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", rf_accuracy)

# Precision: TP/(TP+FP): The ratio of TP to the total predicted positives
rf_precision = precision_score(y_test, y_pred, average="weighted")
print("\nPrecision: ", rf_precision)

# Recall: TP/(TP+FN): Tha ratio of true positives to the total actual pot
rf_recall = recall_score(y_test, y_pred, average="weighted")
print("\nReacll : ", rf_recall)

#F1-score: Harmonic mean of `Precision` and `Recall'
rf_f1 = f1_score(y_test, y_pred, average="weighted")
print("\nF1 score: ", rf_f1)
```

```
# Specificity calculation
cm = confusion_matrix(y_test, y_pred)
specificity = []
for i in range(len(cm)):
    tn = np.sum(cm) - np.sum(cm[i, :]) - np.sum(cm[:, i]) + cm[i, i]
    fp = np.sum(cm[:, i]) - cm[i, i]
    specificity.append(tn / (tn + fp))
    rf_specificity = np.mean(specificity)
print("\nSpecificity: ", rf_specificity)

# Report the summary of all evaluation:
print("\nRandom Forest Classification Report:", classification_report(y_t
print("Classification with Random Forest is done!")
```




```
Accuracy: 0.98

Precision:  0.98125

Reacll :  0.98

F1 score:  0.98

Specificity:  0.9904761904761905

Random Forest Classification Report:              precision    recall  f1
-score   support

           0       1.00      1.00      1.00        19
           1       0.94      1.00      0.97        15
           2       1.00      0.94      0.97        16

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50

Classification with Random Forest is done!
```

Decision Tree classifier

- Building model with Decision Tree classifier

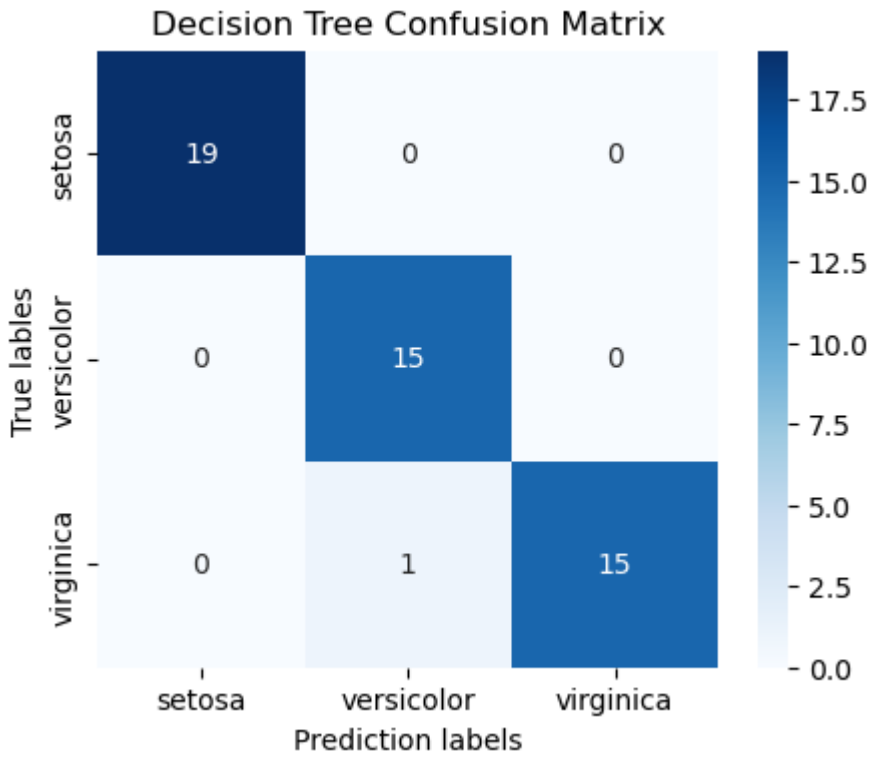
In [56]:
```
# Import Random Forest from sklean package
from sklearn.tree import DecisionTreeClassifier

# Decision Tree Classifier
dt_model = DecisionTreeClassifier(random_state=42)          # Initialize

dt_model.fit(X_train_scaled, y_train)      # Fit the model with scalar-no
dt_predictions = dt_model.predict(X_test_scaled)     # Get the prediction
y_pred = dt_predictions                                 # Predicted result
print("\nDecision Tree Predictions (First 10):", y_pred[:10])
```

```
Decision Tree Predictions (First 10): [1 0 2 1 1 0 1 2 1 1]
```

Evaluation of the performance of Decision Tree classifier

- Evaluation with the **simple validation method**

In [58]:
```
plot_confusion_matrix(y_test, y_pred, "Decision Tree Confusion Matrix")

# Now, let's get the result of RF evaluation
#Accuracy:               (TP+TN)/(TP+TN+FP+FN)
dt_accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", dt_accuracy)

# Precision: TP/(TP+FP): The ratio of TP to the total predicted positives
dt_precision = precision_score(y_test, y_pred, average="weighted")
print("\nPrecision: ", dt_precision)

# Recall: TP/(TP+FN): Tha ratio of true positives to the total actual pot
dt_recall = recall_score(y_test, y_pred, average="weighted")
print("\nReacll : ", dt_recall)

#F1-score: Harmonic mean of `Precision` and `Recall'
dt_f1 = f1_score(y_test, y_pred, average="weighted")
print("\nF1 score: ", dt_f1)

# Specificity calculation
cm = confusion_matrix(y_test, y_pred)
specificity = []
for i in range(len(cm)):
    tn = np.sum(cm) - np.sum(cm[i, :]) - np.sum(cm[:, i]) + cm[i, i]
    fp = np.sum(cm[:, i]) - cm[i, i]
    specificity.append(tn / (tn + fp))
    dt_specificity = np.mean(specificity)
print("\nSpecificity: ", dt_specificity)

# Report the summary of all evaluation:
print("\nDecision Tree Classification Report:", classification_report(y_t
print("Classification with Decision Tree is done!")
```




```
Accuracy: 0.98

Precision:  0.98125

Reacll :  0.98

F1 score:  0.98

Specificity:  0.9904761904761905

Decision Tree Classification Report:              precision    recall  f1
-score   support

           0       1.00      1.00      1.00        19
           1       0.94      1.00      0.97        15
           2       1.00      0.94      0.97        16

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50

Classification with Decision Tree is done!
```

Logistic Regression classifier

- Building model with Logistic Regression classifier

In [60]:
```
# Import Logistic Regression from sklearn package
from sklearn.linear_model import LogisticRegression  # Import Logistic Re

# Logistic Regession Classifier
lr_model = LogisticRegression(random_state=42, max_iter=200)          # I
# Note: Here, max-iter is the maximum number of iterations that the optim

lr_model.fit(X_train_scaled, y_train)      # Fit the model with scalar-no
```

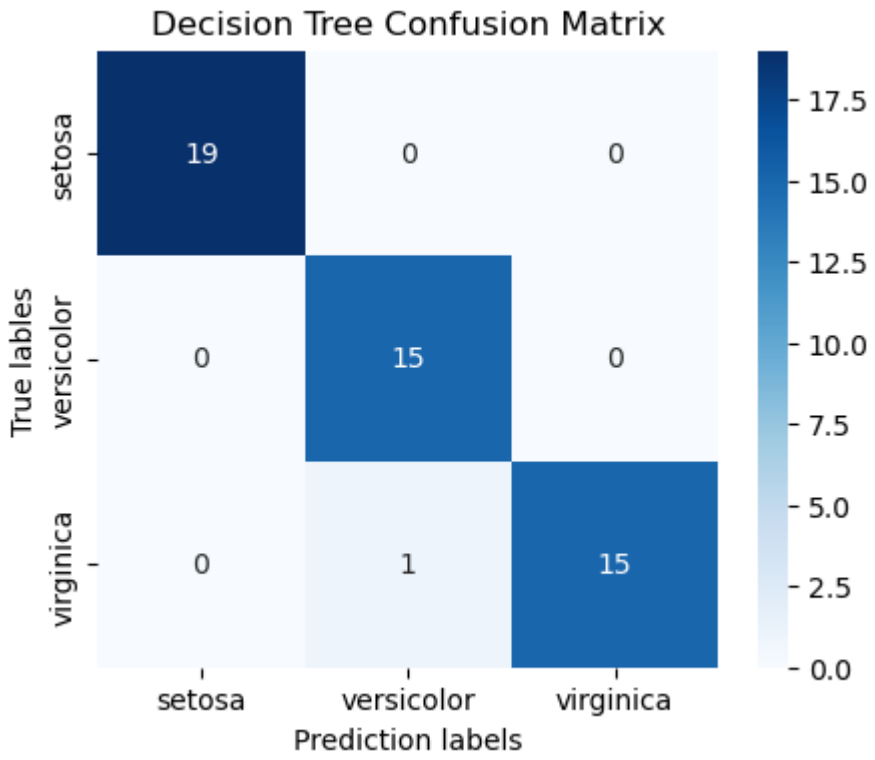
```
lr_predictions = lr_model.predict(X_test_scaled)     # Get the prediction
y_pred = lr_predictions                               # Predicted result
print("\nLogistic Regression Predictions (First 10):", y_pred[:10])
```

```
Logistic Regression Predictions (First 10): [1 0 2 1 1 0 1 2 1 1]
```

Evaluation of the performance of Logistic Regression classifier

- Evaluation with the **simple validation method**

In [64]:
```
plot_confusion_matrix(y_test, y_pred, "Ligistic Regression Confusion Matr

# Now, let's get the result of RF evaluation
#Accuracy:               (TP+TN)/(TP+TN+FP+FN)
lr_accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", lr_accuracy)

# Precision: TP/(TP+FP): The ratio of TP to the total predicted positives
lr_precision = precision_score(y_test, y_pred, average="weighted")
print("\nPrecision: ", lr_precision)

# Recall: TP/(TP+FN): Tha ratio of true positives to the total actual pot
lr_recall = recall_score(y_test, y_pred, average="weighted")
print("\nReacll : ", lr_recall)

#F1-score: Harmonic mean of `Precision` and `Recall'
lr_f1 = f1_score(y_test, y_pred, average="weighted")
print("\nF1 score: ", lr_f1)

# Specificity calculation
cm = confusion_matrix(y_test, y_pred)
specificity = []
for i in range(len(cm)):
    tn = np.sum(cm) - np.sum(cm[i, :]) - np.sum(cm[:, i]) + cm[i, i]
    fp = np.sum(cm[:, i]) - cm[i, i]
    specificity.append(tn / (tn + fp))
    lr_specificity = np.mean(specificity)
print("\nSpecificity: ", lr_specificity)

# Report the summary of all evaluations:
print("\nLogistic Regression Classification Report:", classification_repo
print("Classification with Logistic Regression is done!")
```




```
Accuracy: 0.98

Precision:  0.98125

Reacll :  0.98

F1 score:  0.98

Specificity:  0.9904761904761905

Logistic Regression Classification Report:                precision    reca
ll  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.94      1.00      0.97        15
           2       1.00      0.94      0.97        16

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50

Classification with Logistic Regression is done!
```

XGBoost classifier

- Building model with XGBoost classifier

In [82]:
```
# Import XGBoost from sklearn package
from xgboost import XGBClassifier         # Import XGBoost classification

# XGBoost Classifier
xgb_model = XGBClassifier(random_state=42, eval_metric='mlogloss')    # I
# Note: For details about the paramters see the sklearn manual

xgb_model.fit(X_train_scaled, y_train)      # Fit the model with scalar-n
```

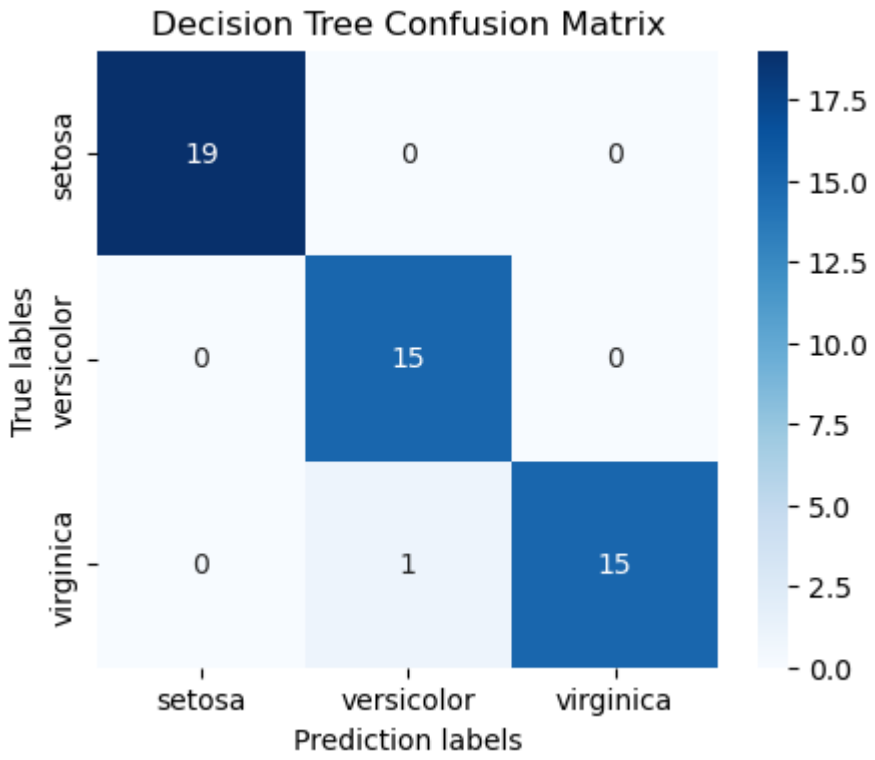
```
xgb_predictions = xgb_model.predict(X_test_scaled)     # Get the predicti
y_pred = xgb_predictions                               # Predicted result
print("\nXGBoost Predictions (First 10):", y_pred[:10])
```

```
XGBoost Predictions (First 10): [1 0 2 1 1 0 1 2 1 1]
```

Evaluation of the performance of XGBoost classifier

- Evaluation with the **simple validation method**

In [85]:
```
plot_confusion_matrix(y_test, y_pred, "XGBoost Confusion Matrix")

# Now, let's get the result of RF evaluation
#Accuracy:               (TP+TN)/(TP+TN+FP+FN)
xgb_accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", xgb_accuracy)

# Precision: TP/(TP+FP): The ratio of TP to the total predicted positives
xgb_precision = precision_score(y_test, y_pred, average="weighted")
print("\nPrecision: ", xgb_precision)

# Recall: TP/(TP+FN): Tha ratio of true positives to the total actual pot
xgb_recall = recall_score(y_test, y_pred, average="weighted")
print("\nReacll : ", xgb_recall)

#F1-score: Harmonic mean of `Precision` and `Recall'
xgb_f1 = f1_score(y_test, y_pred, average="weighted")
print("\nF1 score: ", xgb_f1)

# Specificity calculation
cm = confusion_matrix(y_test, y_pred)
specificity = []
for i in range(len(cm)):
    tn = np.sum(cm) - np.sum(cm[i, :]) - np.sum(cm[:, i]) + cm[i, i]
    fp = np.sum(cm[:, i]) - cm[i, i]
    specificity.append(tn / (tn + fp))
    xgb_specificity = np.mean(specificity)
print("\nSpecificity: ", xgb_specificity)

# Report the summary of all evaluation:
print("\nLogistic XGBoost Classification Report:", classification_report(
print("Classification with XGBoost is done!")
```




```
Accuracy: 0.98

Precision:  0.98125

Reacll :  0.98

F1 score:  0.98

Specificity:  0.9904761904761905

Logistic XGBoost Classification Report:               precision    recall
f1-score   support

           0       1.00      1.00      1.00        19
           1       0.94      1.00      0.97        15
           2       1.00      0.94      0.97        16

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50

Classification with XGBoost is done!
```

Gradient Boosting classifier

- Building model with Gradient Boosting classifier

In [91]:
```
#Import  Gradient Boosting classifier
from sklearn.ensemble import GradientBoostingClassifier

# Gradient Boosting Classifier
gb_model = GradientBoostingClassifier()    # Initialize the  classificati

gb_model.fit(X_train_scaled, y_train)      # Fit the model with scalar-no
gb_predictions = gb_model.predict(X_test_scaled)     # Get the prediction
```

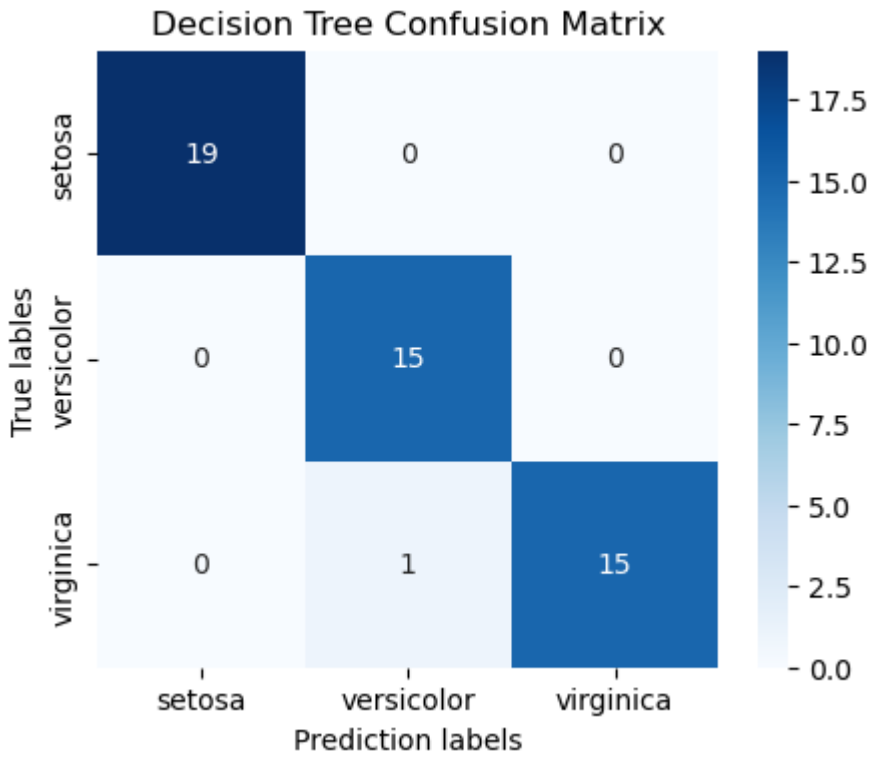
```
y_pred = gb_predictions                                    # Predicted result
print("\nGradient Boosting Predictions (First 10):", y_pred[:10])
```

```
Gradient Boosting Predictions (First 10): [1 0 2 1 1 0 1 2 1 1]
```

Evaluation of the performance of Gradient Boosting classifier

- Evaluation with the **simple validation method**

In [95]:
```
plot_confusion_matrix(y_test, y_pred, "Gradient Boosting Confusion Matrix

# Now, let's get the result of RF evaluation
#Accuracy:               (TP+TN)/(TP+TN+FP+FN)
gb_accuracy = accuracy_score(y_test, y_pred)
print("\nAccuracy:", gb_accuracy)

# Precision: TP/(TP+FP): The ratio of TP to the total predicted positives
gb_precision = precision_score(y_test, y_pred, average="weighted")
print("\nPrecision: ", gb_precision)

# Recall: TP/(TP+FN): Tha ratio of true positives to the total actual pot
gb_recall = recall_score(y_test, y_pred, average="weighted")
print("\nReacll : ", gb_recall)

#F1-score: Harmonic mean of `Precision` and `Recall'
gb_f1 = f1_score(y_test, y_pred, average="weighted")
print("\nF1 score: ", gb_f1)

# Specificity calculation
cm = confusion_matrix(y_test, y_pred)
specificity = []
for i in range(len(cm)):
    tn = np.sum(cm) - np.sum(cm[i, :]) - np.sum(cm[:, i]) + cm[i, i]
    fp = np.sum(cm[:, i]) - cm[i, i]
    specificity.append(tn / (tn + fp))
    gb_specificity = np.mean(specificity)
print("\nSpecificity: ", gb_specificity)

# Report the summary of all evaluation:
print("\nLogistic Gradient Boosting Classification Report:", classificati
print("Classification with Gradient Boosting is done!")
```



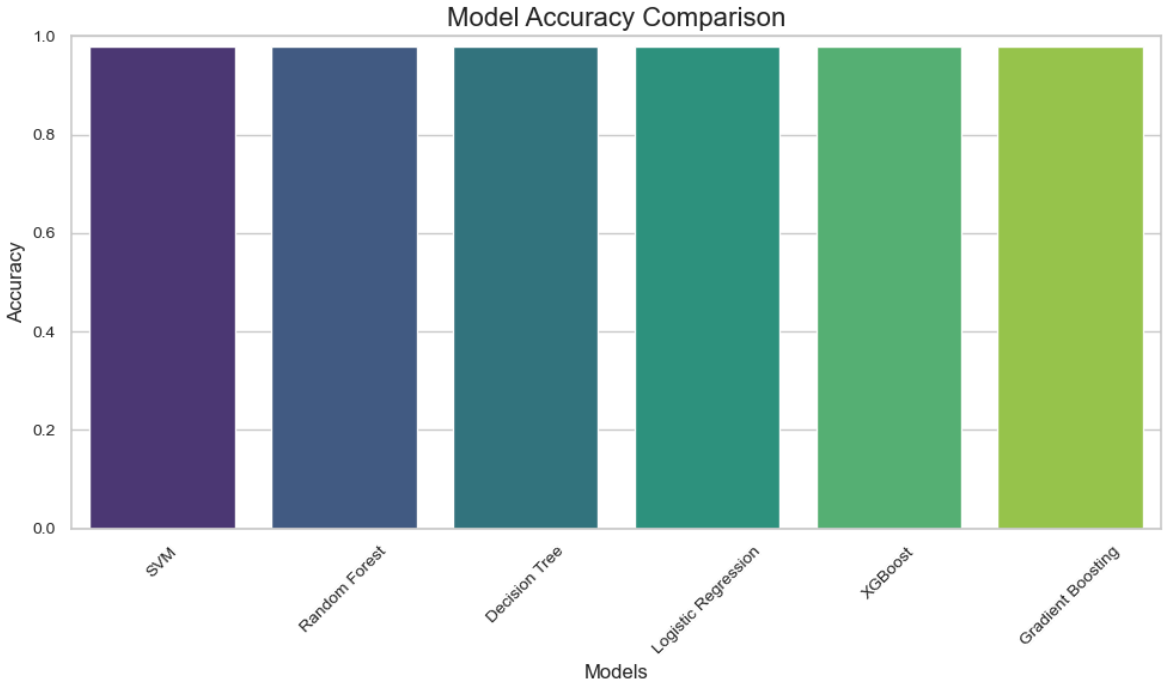

```
Accuracy: 0.98

Precision:  0.98125

Reacll :  0.98

F1 score:  0.98

Specificity:  0.9904761904761905

Logistic Gradient Boosting Classification Report:                precision
recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.94      1.00      0.97        15
           2       1.00      0.94      0.97        16

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50

Classification with Gradient Boosting is done!
```

Comparative study on the Performance of Different classifiers

In [131…

```
# Plot model accuracies

# Model names and their corresponding accuracies
models = ['SVM', 'Random Forest', 'Decision Tree', 'Logistic Regression',
accuracies = [svm_accuracy, rf_accuracy, dt_accuracy, lr_accuracy, xgb_ac

# Plotting
plt.figure(figsize=(10, 6))
# Set Seaborn style and color palette
```

```
sns.set_theme(style="whitegrid")
colors = sns.color_palette("viridis", len(models))
#sns.set_palette("viridis")  # Set the palette globally
sns.barplot(x=models, y=accuracies, palette=colors, hue=models, dodge=Fal
plt.ylim(0, 1)
plt.title('Model Accuracy Comparison', fontsize=16)
plt.xlabel('Models', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.xticks(rotation=45, fontsize=10)
plt.yticks(fontsize=10)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



The Project "Iris Classification" is over!

In [ ]:

Project 2: Clustering of Iris Flowers

- **Input:** Iris.csv data set
- **Project:** Clustering models using K-Means, DBSCAN, etc. unsupervised Machine Learning algorithms

---

Step 1: Import all necessary libraries

The following libraries to be imported in this project:

- pandas: Used to read and manipulate CSV data.
- Numpy: For fast and efficient processing of data
- sklearn.dataset: To load data from the Sci-Kit-Learn repository
- sklearn.train_test_split: From scikit-learn, used to split data into training and testing sets.
- sklearn.preprocessing: For feature scaling/normalization
- sklearn.cluster: A package containing different clustering methods: KMeans, DBSCAN, AgglomerativeClustering.
- sklearn.mixture: A package containing Gaussian mixture model for clustering
- sklearn.accuracy_score: To calculate model accuracy.

In [56]:
```
import pandas as pd
import numpy as np

## The rest of the libraries will be loaded as and when required.
from sklearn.datasets import load_iris          # A special method to do

print("Import of all necessary packages is successful")
```

```
Import of all necessary packages is successful
```

Step 2: Load and check the input dataset

In [7]:
```
# Load the Iris dataset
iris = load_iris()                    # This is a library method defined i
X, y = iris.data, iris.target         # Take it as two-component: Data and

# Convert to DataFrame for better processing
df = pd.DataFrame(data=X, columns=iris.feature_names)
df['target'] = y

# Preview the dataset: It si required as a customary step!
#print("Top 5 rows of the dataset:")
#print(df.head())
#print("Bottom 5 rows of the dataset:")
#print(df.tail())
```

```
#print("The columns present in the data frame
#print(df.columns)
#print("The information about the attributes
print(df.info())
#print("To check if the null entries are there")
#print(df.isnull())
#print("The statistical information about the data")
# print(df.describe())
```

```
<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
```

Step 3: Split the data set into two parts: "Training set" and "Test set"

The following library is used

- import train_test_split from sklearn.model_selection
- "Training set" is used to train a model and "Test set" is used to test a model

In [58]:
```
from sklearn.model_selection import train_test_split
print("Import of \"Train-Test-Split-Selection\" library is successful")

# Split the dataset into training and testing sets: 67% for training and
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,
# Note 1: Data (i.e., Data-attributes and Target-column are kept as separ
# Note 2: Here, random_state=42 is chosen as a seed value and popularly i

print("\nTrain and test data shapes:")
print("X_train:", X_train.shape, "X_test:", X_test.shape)
```

```
Import of "Train-Test-Split-Selection" library is successful

Train and test data shapes:
X_train: (100, 4) X_test: (50, 4)
```

Step 4: Preprocessing

The preprocessing task includes

- (a)Handling null-entries, if applicable
- (b) Scaling (to put all values in a normalize scale

- For scaling there are many methods: StandardScalar, MinMaxScalar, Normalizer, PolynomialFeatures, etc. Use any one.**

In [70]:
```
# Normalization of training and testing data

'''
 Note: For normalization, sklearn provides two methods: fit_transform() a
       fit_transform() is applied to training data, whereas transform() i
       fit_transform() is a combination of fit() (to calculate the necess
       transformation based on the training data, such as, min, max, mean
       transform() applies the transformation to the data using the param

       The three methods applicable to normalization are defined in sklea
'''
# Import scaling methods for normalization
from sklearn.preprocessing import StandardScaler

# Import other normalization methods and use them, if necessary
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import PolynomialFeatures


# Let's use the standard scaling in this project
scaler = StandardScaler()                          # Let StandardScalar(
X_train_scaled = scaler.fit_transform(X_train)     # Apply fit_transform
X_test_scaled = scaler.transform(X_test)           # Apply transform() t
#print("Standard Scaled Data (First 5 rows):\n", X_train_scaled[:5])  # S


# Min-Max scaling
minmax_scaler = MinMaxScaler()    # Let MinMaxScalar() be mimax_scalar
X_train_minmax = minmax_scaler.fit_transform(X_train)      # Apply fit_tr
X_test_minmax = minmax_scaler.transform(X_test)            # Apply transf
#print("\nMin-Max Scaled Data (First 5 rows):\n", X_train_minmax[:5])

# Normalization
normalizer = Normalizer()        # Let Normalizer() be normalizer
X_train_normalized = normalizer.fit_transform(X_train)
X_test_normalized = normalizer.transform(X_test)
#print("\nNormalized Data (First 5 rows):\n", X_train_normalized[:5])

# Polynomial-features scaling
poly = PolynomialFeatures(degree=2, include_bias=False)
X_train_poly = poly.fit_transform(X_train)
X_test_poly = poly.transform(X_test)
#print("\nPolynomial Features (First 5 rows):\n", X_train_poly[:5])
```

Step 5: Dimensionality reduction

There are several methods defined in sklearn:

- PCA (Principal Component Analysis), IDA (Independent Component Analysis), LDA (Linear Discrimnant Analysis), NMF (Non-negative Matrix Factorization, SVD (Singular Value Decomposition), etc. are a few popular dimensionality reduction techniques

- This project follows PCA
- **Note:** Dimnsionality reduction method is optional and does not necessarily yield good results.

In [98]:
```
# Using PCA for dimensionality reduction

from sklearn.decomposition import PCA, NMF
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as L
from sklearn.decomposition import TruncatedSVD

#PCA (Principal Component Analysis)
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Display PCA results
print("\nPCA Reduced Training data shape (2 components):", X_train_pca.sh
print("\nPCA Reduced Testing data shape (2 components):", X_test_pca.shap

# LDA (Linear Discriminant Analysis)
lda = LDA(n_components=2)
X_train_lda = lda.fit_transform(X_train_scaled, y_train)
X_test_lda = lda.transform(X_test_scaled)

# Display LDA results
print("\nLDA Reduced Training data shape (2 components):", X_train_lda.sh
print("\nLDA Reduced Testing data shape (2 components):", X_test_lda.shap

# NMF (Non-Negative Matrix Factorization)
nmf = NMF(n_components=2, init='random', random_state=42)
X_train_nmf = nmf.fit_transform(np.abs(X_train_scaled))          # Ensure
X_test_nmf = nmf.transform(np.abs(X_test_scaled))

# Display NMF results
print("\nNMF Reduced Training data shape (2 components):", X_train_nmf.sh
print("\nNMF Reduced Testing data shape (2 components):", X_test_nmf.shap

# SVD (Singular Value Decomposition)
svd = TruncatedSVD(n_components=2)
X_train_svd = svd.fit_transform(X_train_scaled)
X_test_svd = svd.transform(X_test_scaled)

# Display SVD results
print("\nSVD Reduced Training data shape (2 components):", X_train_svd.sh
print("\nLDA Reduced Testing data shape (2 components):", X_test_svd.shap
```

```
PCA Reduced Training data shape (2 components): (100, 2)

PCA Reduced Testing data shape (2 components): (50, 2)

LDA Reduced Training data shape (2 components): (100, 2)

LDA Reduced Testing data shape (2 components): (50, 2)

NMF Reduced Training data shape (2 components): (100, 2)

NMF Reduced Testing data shape (2 components): (50, 2)

SVD Reduced Training data shape (2 components): (100, 2)

LDA Reduced Testing data shape (2 components): (50, 2)
```

Step 6: Building Clustering Models

There are several ML algorithms that can be followed to build clusters. In this project, we shall follow the following ML algorithms.

- k-Means clustering
- DBSCAN clustering
- Agglomerative clustering
- Gaussian Mixture clustering

Clustering methods intiations:

```
km = KMeans(n_clusters=3, random_state=42)
db = DBSCAN(eps=0.5, min_samples=3)
am = AgglomerativeClustering(n_clusters=3)
gm = GaussianMixture(n_components=3, random_state=42)
```

k_means Clustering

- Clustering with partition-based clustering algorithm

In [74]:
```
# KMeans clustering algorithms

# Import clustering models
from sklearn.cluster import KMeans

# Clustering: build clustering with "training data set"
km = KMeans(n_clusters=3, random_state=42)
km.fit(X_train_scaled)                              # Learn the clustering
km_labels = km.predict(X_test_scaled)
print("\nKMeans Cluster Labels (First 10):", km_labels[:10])
```

```
KMeans Cluster Labels (First 10): [1 0 2 1 1 0 1 2 1 1]
```

DBSCAN Clustering

- Clustering with density-based clustering algorithm

In [76]:
```
from sklearn.cluster import DBSCAN

# Clustering: build clustering with "training data set"
db = DBSCAN(eps=0.5, min_samples=3)
db.fit(X_train_scaled)          # DBSCAN is density-based clustering, unl

# Using the test dataset to assign cluster labels (DBSCAN does not have '
db_labels = db.fit_predict(X_test_scaled)     # Predict the cluster label

print("\nDBSCAN Cluster Labels (First 10):", db_labels[:10])
```

```
DBSCAN Cluster Labels (First 10): [-1  1 -1 -1 -1  0 -1  3 -1 -1]
```

Agglomerative Clustering

- Clustering with hierachical clustering algorithm

In [78]:
```
# Agglomerative clustering algorithms

# Import clustering models
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score, davies_bouldin_score, calin

# Clustering: build clustering with "training data set"
am = AgglomerativeClustering(n_clusters=3)
am.fit(X_train_scaled)                              # Learn the clustering
am_labels = am.fit_predict(X_test_scaled)
print("\nKMeans Cluster Labels (First 10):", am_labels[:10])
```

```
KMeans Cluster Labels (First 10): [2 0 1 2 1 0 2 1 2 2]
```

Gaussian Mixture Model of Clustering

- Clustering baed on Gausian mixture model (GMM)

In [80]:
```
# Gaussian Mixture clustering

# Import the packages
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score, davies_bouldin_score, calin

# Clustering: build clustering with "training data set"
gmm = GaussianMixture(n_components=3)
gmm.fit(X_train_scaled)
gmm_labels = gmm.predict(X_test_scaled)
print("\nGaussian Mixture Cluster Labels (First 10):", gmm_labels[:10])
```

```
Gaussian Mixture Cluster Labels (First 10): [2 1 0 2 2 1 2 0 2 2]
```

Step 7: Evaluation of Clustering Performance

The following metrics are popular to validate clusuer quality

A. Silhouette Score
   - Higher score is preferable
B. Davies-Bouldin Index

- Lower score is preferable

C. Calinski-Harabasz Index
- Higher score is preferable

In [82]:
```
# Import the packages for the evaluation metrics
from sklearn.metrics import silhouette_score, davies_bouldin_score, calin

# Evaluation of k-Means clustering performance: Use the testing data set
km_silhouette = silhouette_score(X_test_scaled, km_labels)
km_davies_bouldin = davies_bouldin_score(X_test_scaled, km_labels)
km_calinski_harabasz = calinski_harabasz_score(X_test_scaled, km_labels)

print("\nPerformance of k-Means clustering :")
print("Silhouette score: ", km_silhouette)
print("Davies_Bouldin Index: ",  km_davies_bouldin)
print("Calinski_Harabasz Index: ", km_calinski_harabasz)


# Evaluation of DBSACN clustering performance: Use the testing data set
# Only calculate metrics if there is more than one cluster
if len(set(db_labels)) > 1:
    db_silhouette = silhouette_score(X_test_scaled, db_labels)
    db_davies_bouldin = davies_bouldin_score(X_test_scaled, db_labels)
    db_calinski_harabasz = calinski_harabasz_score(X_test_scaled, db_labe

    print("\nPerformance of DBSCAN clustering :")
    print("Silhouette score: ", db_silhouette)
    print("Davies_Bouldin Index: ", db_davies_bouldin)
    print("Calinski_Harabasz Index: ", db_calinski_harabasz)
else:
    print("\nDBSCAN clustering resulted in a single cluster or noise. Per


# Evaluation of Agglomerative clustering performance: Use the testing dat
am_silhouette = silhouette_score(X_test_scaled, am_labels)
am_davies_bouldin = davies_bouldin_score(X_test_scaled, am_labels)
am_calinski_harabasz = calinski_harabasz_score(X_test_scaled, am_labels)

print("\nPerformance of Agglomerative clustering :")
print("Silhouette score: ", am_silhouette)
print("Davies_Bouldin Index: ",  am_davies_bouldin)
print("Calinski_Harabasz Index: ", am_calinski_harabasz)

# Evaluation of GMM clustering performance: Use the testing data set
gmm_silhouette = silhouette_score(X_test_scaled, gmm_labels)
gmm_davies_bouldin = davies_bouldin_score(X_test_scaled, gmm_labels)
gmm_calinski_harabasz = calinski_harabasz_score(X_test_scaled, gmm_labels

print("\nPerformance of Gaussian Mixture Model clustering :")
print("Silhouette score: ", gmm_silhouette)
print("Davies_Bouldin Index: ",  gmm_davies_bouldin)
print("Calinski_Harabasz Index: ", gmm_calinski_harabasz)
```

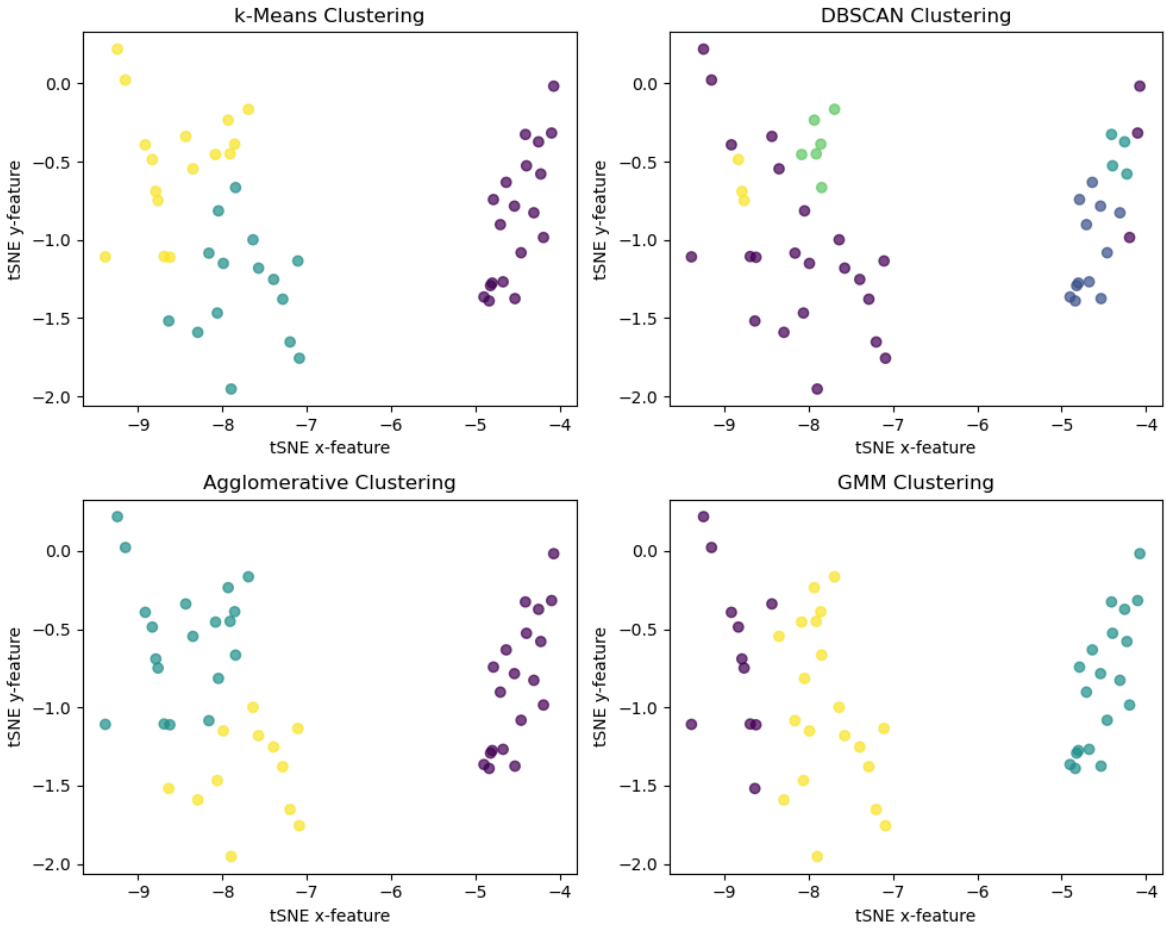
```
Performance of k-Means clustering :
Silhouette score:  0.4210004812765778
Davies_Bouldin Index:  0.9393158190308629
Calinski_Harabasz Index:  79.17686608604731

Performance of DBSCAN clustering :
Silhouette score:  0.029476423715931174
Davies_Bouldin Index:  2.0506227240487127
Calinski_Harabasz Index:  13.126853015201641

Performance of Agglomerative clustering :
Silhouette score:  0.4317277722434559
Davies_Bouldin Index:  0.8981491987778765
Calinski_Harabasz Index:  79.45002644134509

Performance of Gaussian Mixture Model clustering :
Silhouette score:  0.42290334959270026
Davies_Bouldin Index:  0.9882792690749124
Calinski_Harabasz Index:  75.74409477521915
```

Step 8: Visualization of clusters using tSNE plot

In [111…

```python
# Import the necessary package...for tSNE (t-distributed Stochastic Neigh

from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# Get your data ready...
tsne = TSNE(n_components=2, random_state=42)    # Initialize tSNE
X_tsne = tsne.fit_transform(X_test_scaled)   # Input data set to tSNE to

plt.figure(figsize=(10, 8))       # Define the size of your figure....
# Visualization with t-SNE: k-Means graph
plt.subplot(2,2,1)
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c = km_labels, cmap="viridis", al
plt.title("k-Means Clustering")
plt.xlabel('tSNE x-feature')
plt.ylabel('tSNE y-feature')

# Visualization with t-SNE: DBSCAN graph
plt.subplot(2,2,2)
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c = db_labels, cmap="viridis", al
plt.title("DBSCAN Clustering")
plt.xlabel('tSNE x-feature')
plt.ylabel('tSNE y-feature')

# Visualization with t-SNE: Agglomerative graph
plt.subplot(2,2,3)
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c = am_labels, cmap="viridis", al
plt.title("Agglomerative Clustering")
plt.xlabel('tSNE x-feature')
plt.ylabel('tSNE y-feature')

# Visualization with t-SNE: Agglomerative graph
plt.subplot(2,2,4)
plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c = gmm_labels, cmap="viridis", a
plt.title("GMM Clustering")
plt.xlabel('tSNE x-feature')
plt.ylabel('tSNE y-feature')
```

```
plt.tight_layout()
plt.show()
```



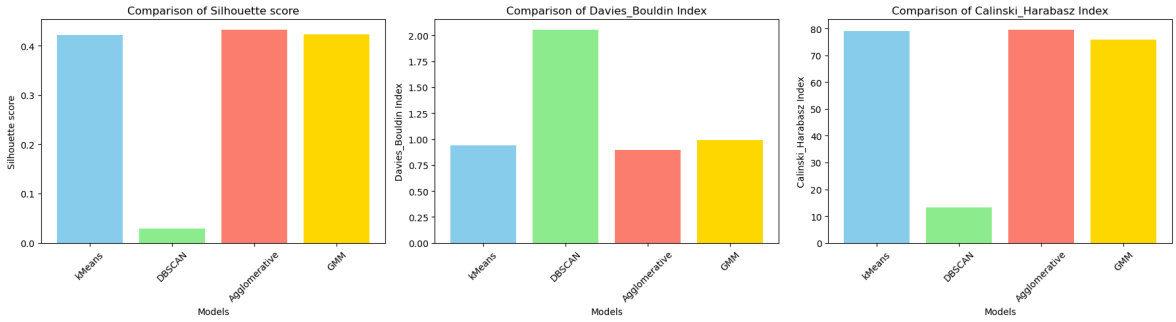
Step 9: Comparing the cluster performance

In [107…

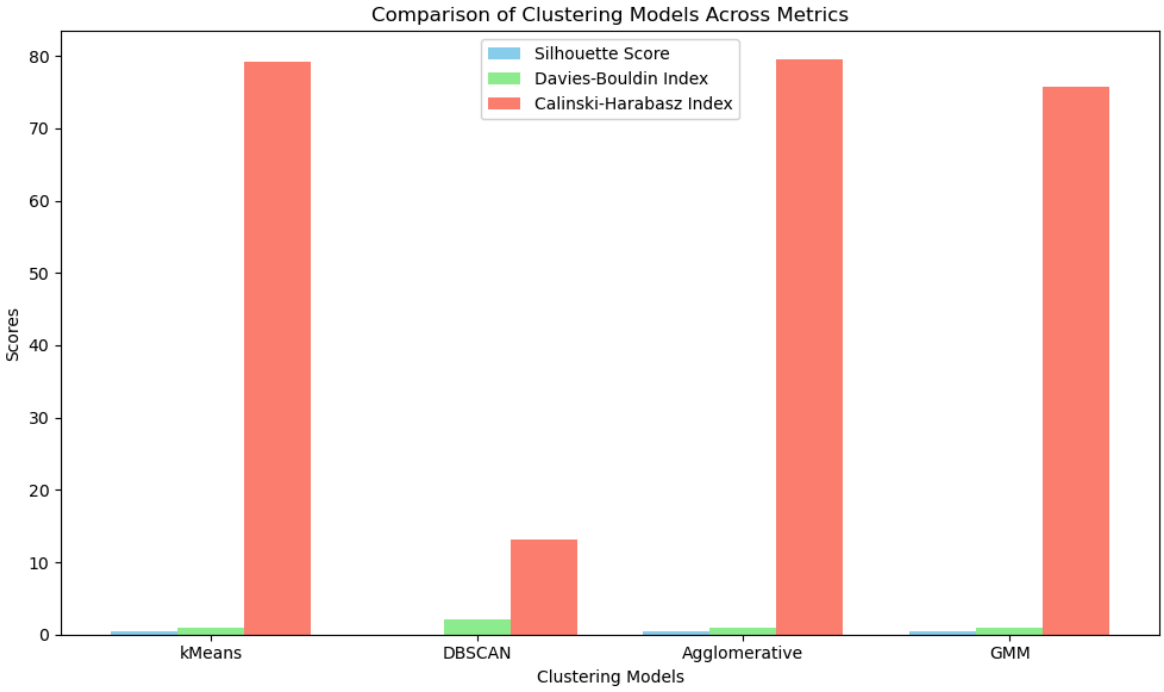
```
import matplotlib.pyplot as plt

# Create your data sets put into the data frame
eval_data = {"Models":['kMeans', 'DBSCAN', 'Agglomerative', 'GMM'], "Silh

             "Davies_Bouldin Index":[km_davies_bouldin, db_davies_bouldin
             "Calinski_Harabasz Index":[km_calinski_harabasz, db_calinski
edf = pd.DataFrame(eval_data)
display(edf)

# Plot the graph of comparison
#plt.figure(figsize=(10,8)

# Plot bar charts for each metric
metrics = ["Silhouette score", "Davies_Bouldin Index", "Calinski_Harabasz
fig, axes = plt.subplots(1, 3, figsize=(18, 5), sharey=False)

for i, metric in enumerate(metrics):
    ax = axes[i]
    ax.bar(edf["Models"], edf[metric], color=['skyblue', 'lightgreen', 's
    ax.set_title(f'Comparison of {metric}')
    ax.set_ylabel(metric)
    ax.set_xlabel("Models")
    ax.set_xticks(np.arange(len(edf["Models"])))
```

```
    ax.set_xticklabels(edf["Models"], rotation=45)

plt.tight_layout()
plt.show()
```

	Models	Silhouette score	Davies_Bouldin Index	Calinski_Harabasz Index
0	kMeans	0.421000	0.939316	79.176866
1	DBSCAN	0.029476	2.050623	13.126853
2	Agglomerative	0.431728	0.898149	79.450026
3	GMM	0.422903	0.988279	75.744095



In [88]:
```
# Bar chart with grouped bars
x = np.arange(len(edf["Models"]))  # X-axis positions for models
width = 0.25  # Width of each bar

fig, ax = plt.subplots(figsize=(10, 6))

# Plotting each metric
bars1 = ax.bar(x - width, edf["Silhouette score"], width, label="Silhouet
bars2 = ax.bar(x, edf["Davies_Bouldin Index"], width, label="Davies-Bould
bars3 = ax.bar(x + width, edf["Calinski_Harabasz Index"], width, label="C

# Adding labels, title, and legend
ax.set_xlabel("Clustering Models")
ax.set_ylabel("Scores")
ax.set_title("Comparison of Clustering Models Across Metrics")
ax.set_xticks(x)
ax.set_xticklabels(edf["Models"])
ax.legend()

# Display the bar chart
plt.tight_layout()
plt.show()
```



The Project "Iris Clustering" is over!