Lab – 03: Exploratory Data Analysis (EDA)

1. Introduction

Exploratory Data Analysis (EDA) is the process of examining datasets to summarize their main characteristics, often using visualizations and statistical measures.

EDA helps to:

- Identify missing values or anomalies
- Detect patterns and relationships between features
- Prepare data for modeling

2. Experiments and Observations

2.1 Loading and Inspecting Data

Procedure:

- Loaded the Iris dataset using `pandas`.
- Inspected the dataset using `head()`, `tail()`, `info()`, and `describe()` methods.

Code:

```
import pandas as pd
from sklearn.datasets import load_iris

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

# Inspecting the data
print("First 5 records:")
print(df.head())

print("\nLast 5 records:")
print(df.tail())

print("\nInfo about dataset:")
print(df.info())

print("\nStatistical summary:")
print(df.describe())
```

Observation:

```
First 5 records:
   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 5 records:
     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
```

```
Info about dataset:
<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

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

       petal width (cm)      target
count        150.000000  150.000000
mean           1.199333    1.000000
std            0.762238    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

2.2 Statistical Summary of Data

Procedure:

Calculated statistical measures: mean, median, mode, standard deviation, variance, correlation.

Code :

```
# Mean, median, standard deviation
print("Mean:\n", df.mean())
print("\nMedian:\n", df.median())
print("\nMode:\n", df.mode().iloc[0])
print("\nStandard Deviation:\n", df.std())
print("\nVariance:\n", df.var())

# Correlation
print("\nCorrelation matrix:\n", df.corr())
```

Observation :

```
Mean:
 sepal length (cm)    5.843333
sepal width (cm)     3.057333
petal length (cm)    3.758000
petal width (cm)     1.199333
target               1.000000
dtype: float64

Median:
 sepal length (cm)    5.80
sepal width (cm)     3.00
petal length (cm)    4.35
petal width (cm)     1.30
target               1.00
dtype: float64

Mode:
 sepal length (cm)    5.0
sepal width (cm)     3.0
petal length (cm)    1.4
petal width (cm)     0.2
target               0.0
Name: 0, dtype: float64

Standard Deviation:
 sepal length (cm)    0.828066
sepal width (cm)     0.435866
petal length (cm)    1.765298
petal width (cm)     0.762238
target               0.819232
dtype: float64
```


```
Standard Deviation:
 sepal length (cm)    0.828066
sepal width (cm)     0.435866
petal length (cm)    1.765298
petal width (cm)     0.762238
target               0.819232
dtype: float64
```


```
Variance:
 sepal length (cm)    0.685694
sepal width (cm)     0.189979
petal length (cm)    3.116278
petal width (cm)     0.581006
target               0.671141
dtype: float64

Correlation matrix:
                    sepal length (cm)  sepal width (cm)  petal length (cm)  \
sepal length (cm)           1.000000         -0.117570           0.871754
sepal width (cm)           -0.117570          1.000000          -0.428440
petal length (cm)           0.871754         -0.428440           1.000000
petal width (cm)            0.817941         -0.366126           0.962865
target                      0.782561         -0.426658           0.949035

                   petal width (cm)    target
sepal length (cm)          0.817941  0.782561
sepal width (cm)          -0.366126 -0.426658
petal length (cm)          0.962865  0.949035
petal width (cm)           1.000000  0.956547
target                     0.956547  1.000000
```

2.3 Handling Missing Values and Outliers

Procedure:

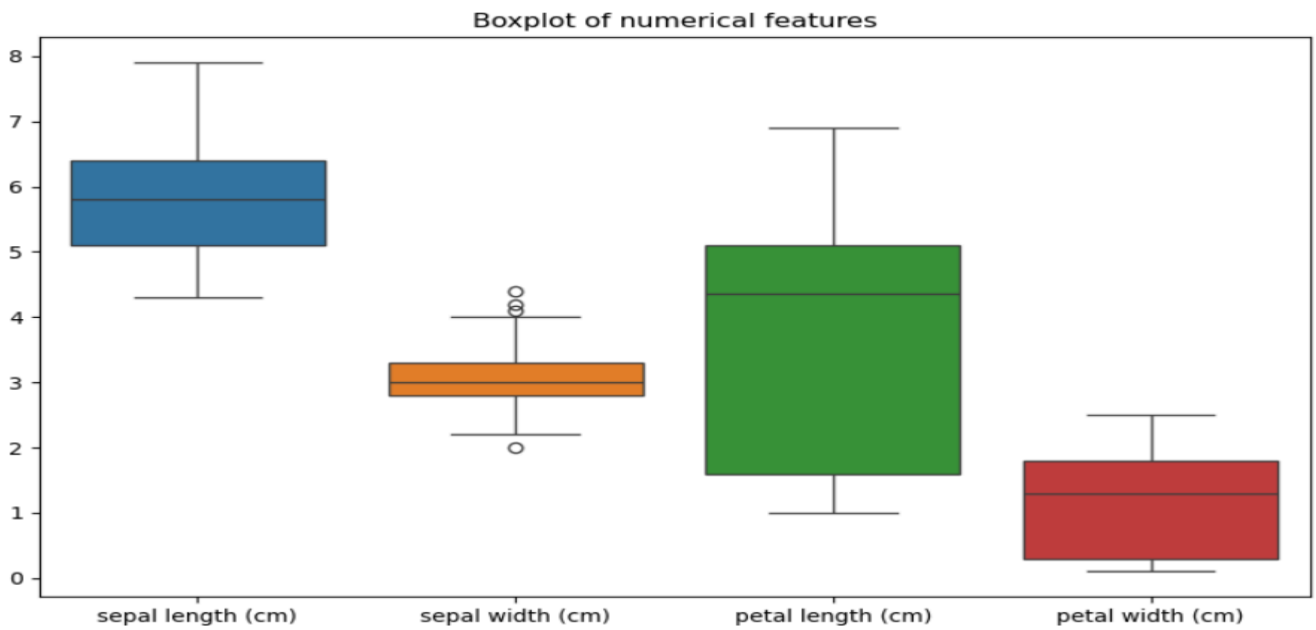
Checked for missing values using `isnull().sum()`.
Visualized outliers using boxplots and handled them.

Code :

```
import matplotlib.pyplot as plt

import seaborn as sns

# Check missing values

print("Missing values:\n", df.isnull().sum())

# Boxplot to visualize outliers

plt.figure(figsize=(10,6))

sns.boxplot(data=df.drop('target', axis=1))

plt.title("Boxplot of numerical features")

plt.show()
```

Observation :

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



2.4 Data Visualization

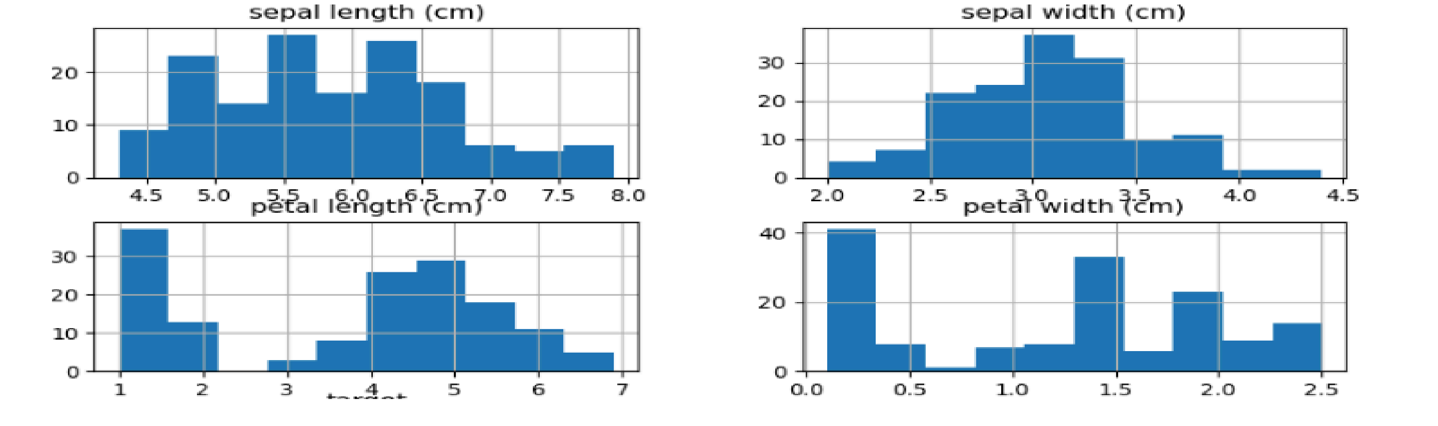
Procedure:

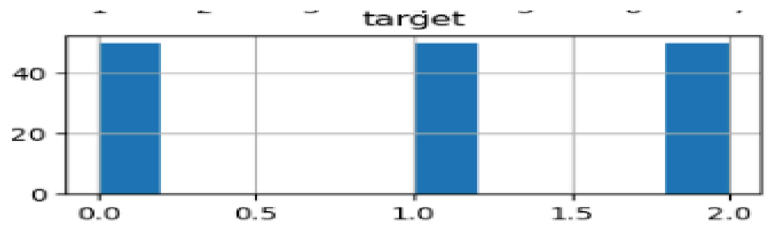
●Created histograms, scatter plots, bar charts, pairplots, and correlation heatmaps.

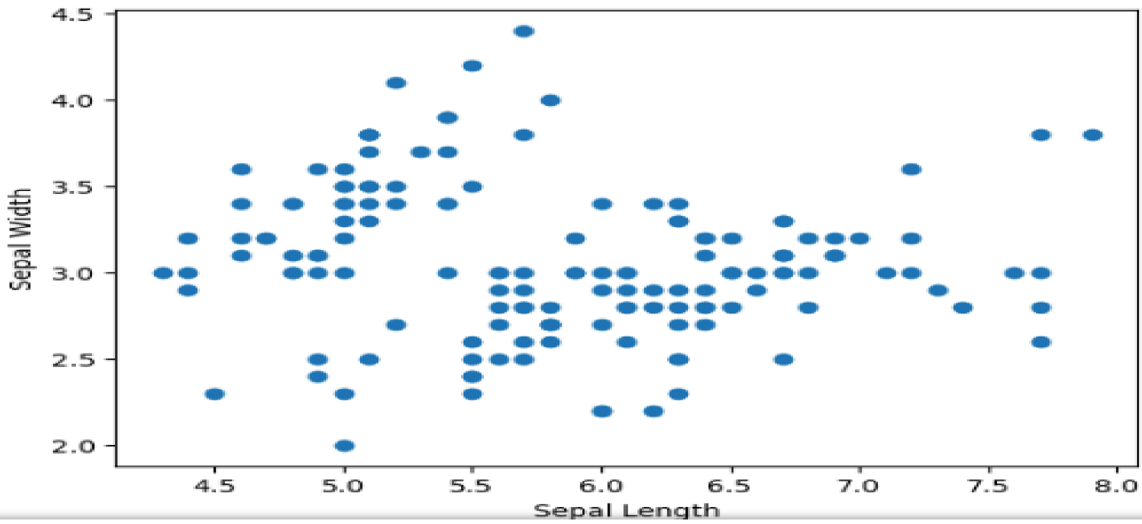
Code:

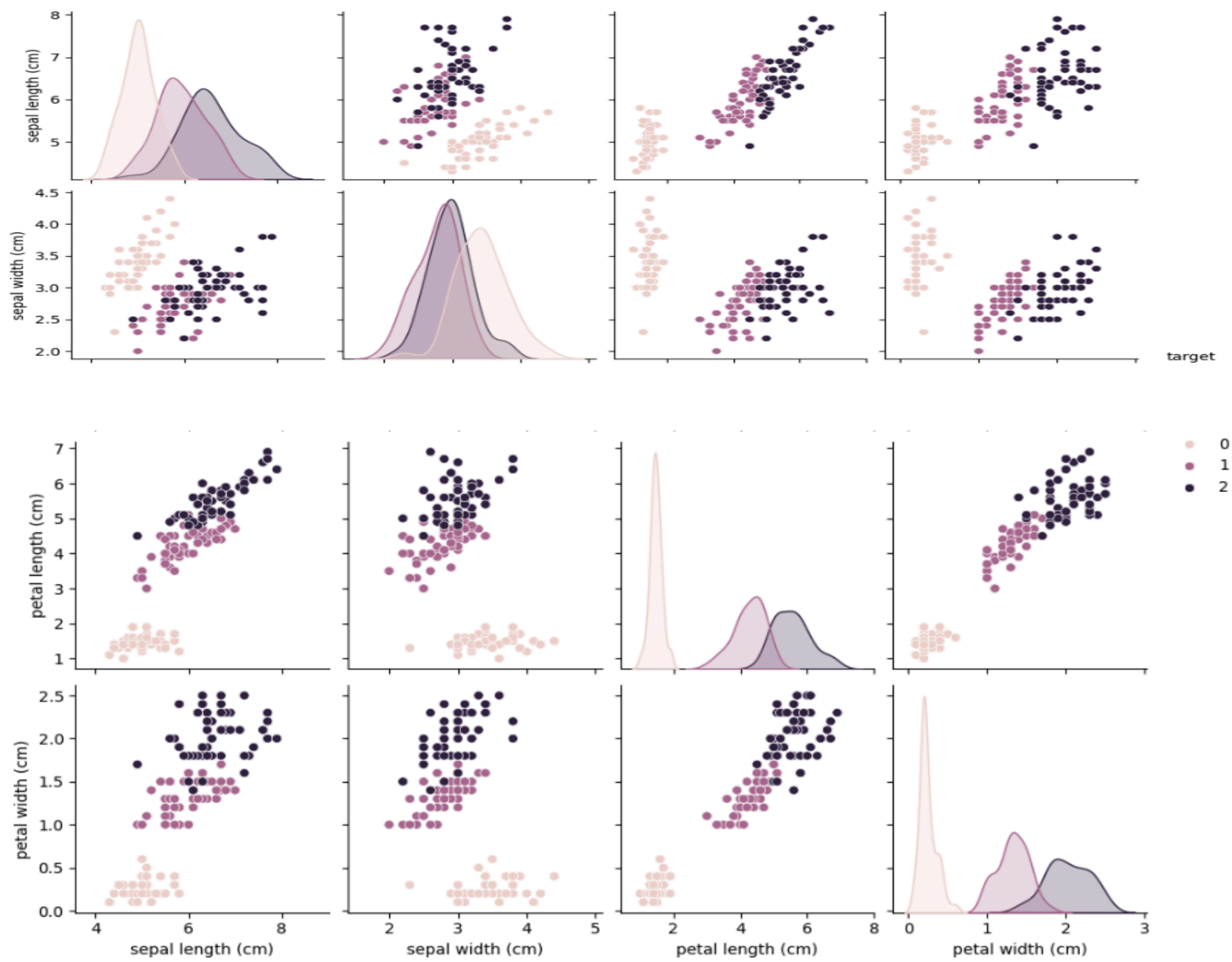
```
# Histogram

df.hist(figsize=(10,6))

plt.show()

# Scatter plot

plt.scatter(df['sepal length (cm)'], df['sepal width (cm)'])

plt.xlabel("Sepal Length")

plt.ylabel("Sepal Width")

plt.show()

# Pairplot

sns.pairplot(df, hue='target')

plt.show()

# Correlation heatmap

plt.figure(figsize=(8,6))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

plt.show()
```

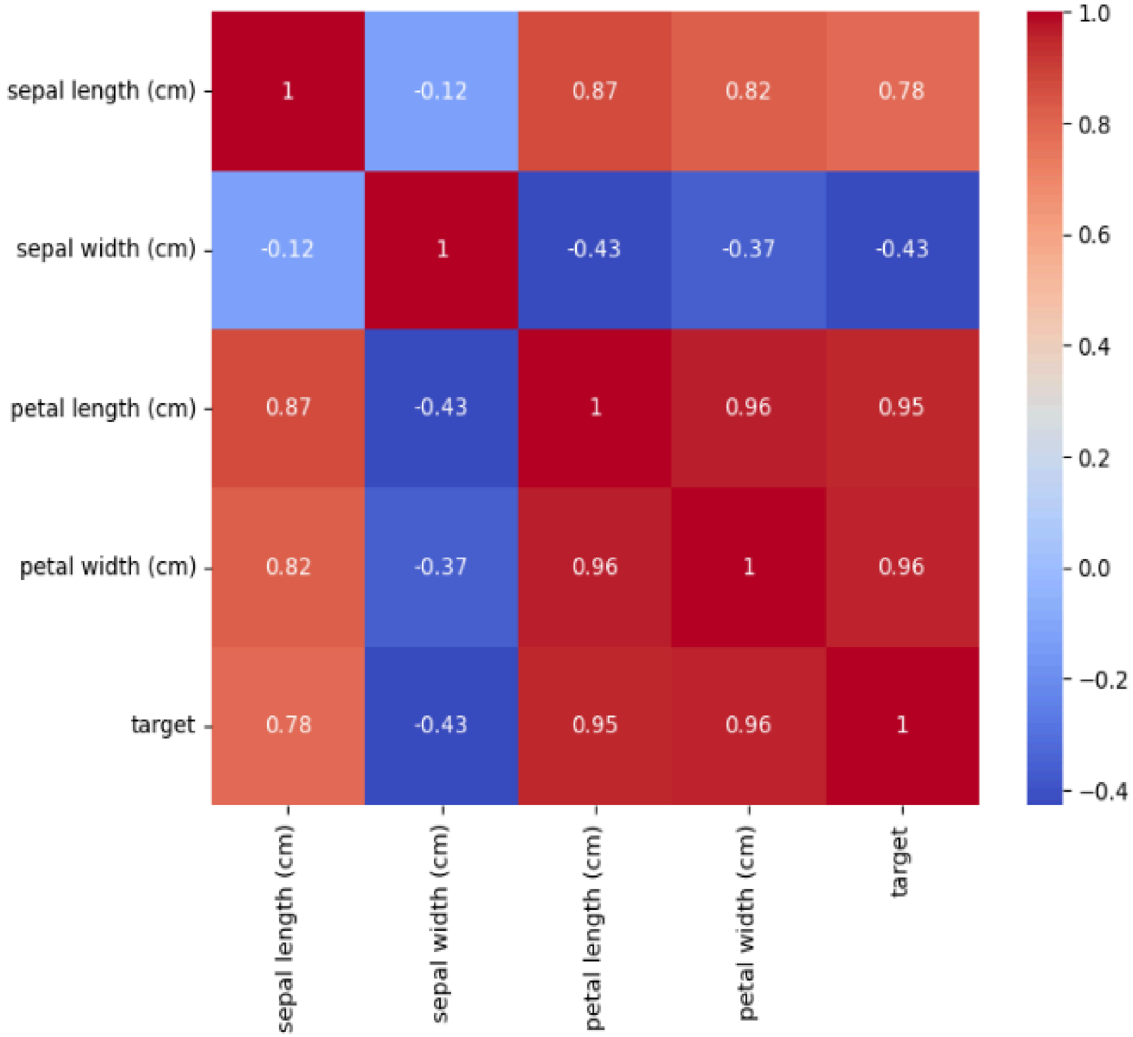
Observation:




target
40
20
0
0.0
0.5
1.0
1.5
2.0


4.5
4.0
3.5
3.0
2.5
2.0
Sepal Width
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
Sepal Length


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
target
0
1
2
sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)



2.5 Feature Analysis

Procedure:

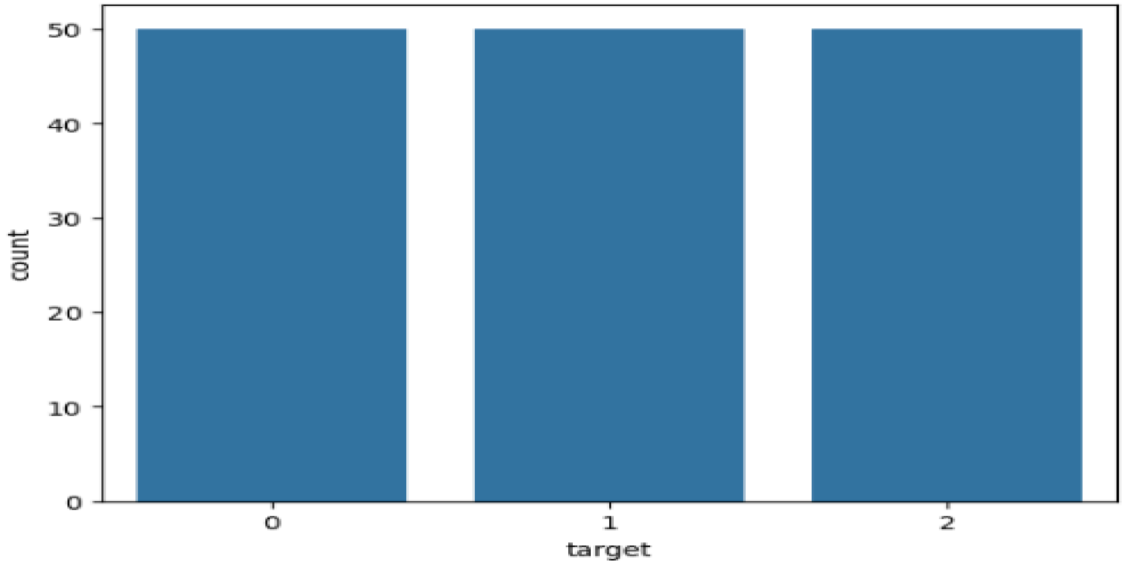
Analyzed categorical target variable using `value_counts()` and bar plots.
Examined numerical features using density plots and boxplots.

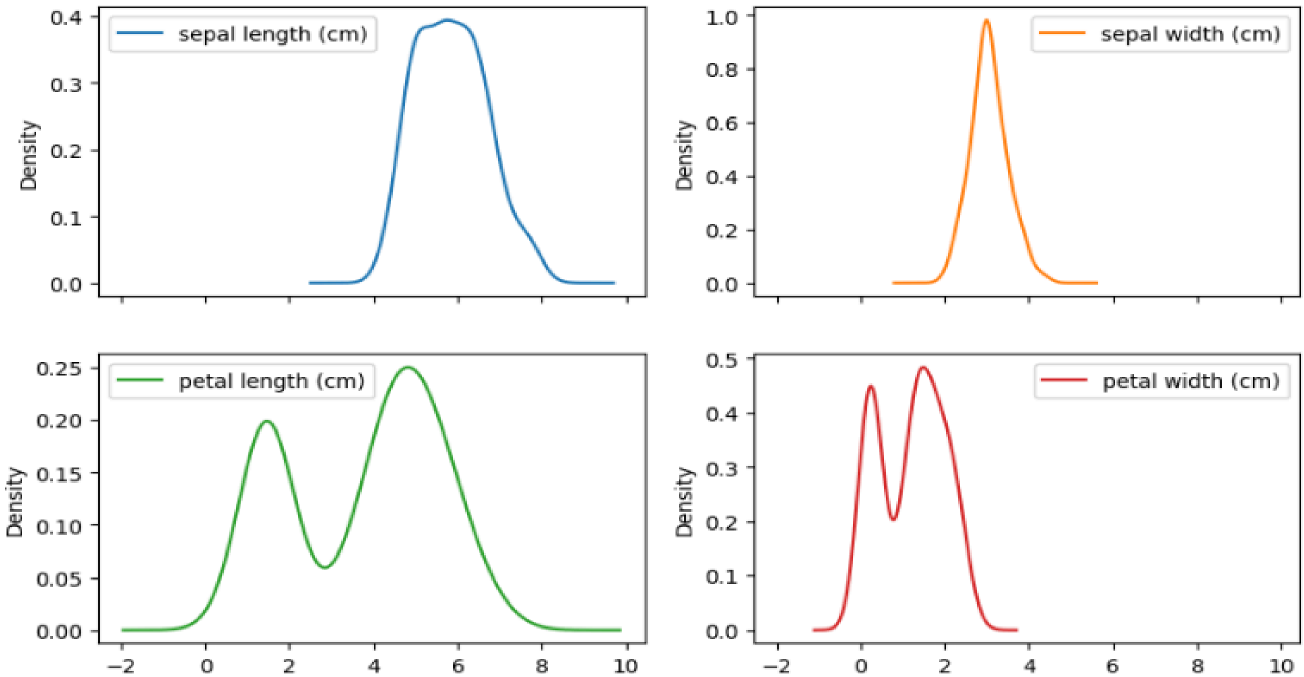
` **Code :**

```
# Categorical variable analysis
print("Target value counts:\n", df['target'].value_counts())
sns.countplot(x='target', data=df)
plt.show()

# Density plots for numerical features
df.drop('target', axis=1).plot(kind='density', subplots=True, layout=(2,2), figsize=(10,6))
plt.show()
```

Observation :

```
Target value counts:
 target
0     50
1     50
2     50
Name: count, dtype: int64
```





3. Conclusion

- EDA helps understand dataset structure, identify missing values, outliers, and feature distributions.
- Visualizations provide insights into feature relationships and correlations.
- Proper analysis of features is essential before applying machine learning algorithms.