Iris_Flowers_Classification

May 30, 2024

1 Objective : Iris Flowers Classification

2 Exploratory Data Analysis (EDA) - Python

3 Insights - Patterns

4 Classification (Using the ML)

Iris Versicolor Iris Setosa Iris Virginica

5 1. Load Python Modules

[1]:
```
#  Use Python's import statement to load modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from tabulate import tabulate

from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import LabelEncoder
```

```python
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.naive_bayes import CategoricalNB
from sklearn.naive_bayes import GaussianNB

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier


from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

6 2. Read the Dataset from CSV file - Using Pandas

[2]:
```python
file_path=r"iris_dataset.csv"
iris_df=pd.read_csv(file_path)
iris_df
```

[2]:
```
     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 target_names
0         0       setosa
```

```
1          0       setosa
2          0       setosa
3          0       setosa
4          0       setosa
..       …          …
145        2    virginica
146        2    virginica
147        2    virginica
148        2    virginica
149        2    virginica

[150 rows x 6 columns]
```

[3]:
```
#rename the columns names
iris_df.columns = ['sepal_length', 'sepal_width', 'petal_length',␣
 ↪'petal_width', 'species_labels','species']
```

7 3. Basic Inspection on given dataset

[4]:
```
def basic_inspection_dataset(table):
    print("Top 5 Records of dataset")
    print(table.head())
    print()

    print("Bottom Records of dataset")
    print(table.tail())
    print()

    print("Column/features/Variable  - Names of Given dataset")
    print(table.columns)
    print()

    print("Shape(rows x columns) - of Given dataset")
    print(table.shape)
    print()

    print("Data types - Given Column Names")
    print(table.dtypes)
    print()

    print("Summry of dataset")
    print(table.info())
    print()

    print("To see the count of null/nan values in columns of dataset")
    print(table.isnull().value_counts())
```

```
    print()

    print("Dataset Summary ")
    print(table.describe())
    print()

basic_inspection_dataset(iris_df)
```

```
Top 5 Records of dataset
   sepal_length  sepal_width  petal_length  petal_width  species_labels  \
0           5.1          3.5           1.4          0.2               0
1           4.9          3.0           1.4          0.2               0
2           4.7          3.2           1.3          0.2               0
3           4.6          3.1           1.5          0.2               0
4           5.0          3.6           1.4          0.2               0

  species
0  setosa
1  setosa
2  setosa
3  setosa
4  setosa

Bottom Records of dataset
     sepal_length  sepal_width  petal_length  petal_width  species_labels  \
145           6.7          3.0           5.2          2.3               2
146           6.3          2.5           5.0          1.9               2
147           6.5          3.0           5.2          2.0               2
148           6.2          3.4           5.4          2.3               2
149           5.9          3.0           5.1          1.8               2

       species
145  virginica
146  virginica
147  virginica
148  virginica
149  virginica

Column/features/Variable  - Names of Given dataset
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species_labels', 'species'],
      dtype='object')

Shape(rows x columns) - of Given dataset
(150, 6)

Data types - Given Column Names
sepal_length      float64
```

```
sepal_width       float64
petal_length      float64
petal_width       float64
species_labels      int64
species            object
dtype: object

Summry of dataset
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   sepal_length    150 non-null    float64
 1   sepal_width     150 non-null    float64
 2   petal_length    150 non-null    float64
 3   petal_width     150 non-null    float64
 4   species_labels  150 non-null    int64
 5   species         150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
None

To see the count of null/nan values in columns of dataset
sepal_length  sepal_width  petal_length  petal_width  species_labels  species
False         False        False         False        False           False
150
Name: count, dtype: int64

Dataset Summary
       sepal_length  sepal_width  petal_length  petal_width  species_labels
count    150.000000   150.000000    150.000000   150.000000      150.000000
mean       5.843333     3.057333      3.758000     1.199333        1.000000
std        0.828066     0.435866      1.765298     0.762238        0.819232
min        4.300000     2.000000      1.000000     0.100000        0.000000
25%        5.100000     2.800000      1.600000     0.300000        0.000000
50%        5.800000     3.000000      4.350000     1.300000        1.000000
75%        6.400000     3.300000      5.100000     1.800000        2.000000
max        7.900000     4.400000      6.900000     2.500000        2.000000
```

7.1 Observations on Iris Dataset

1. **Dataset Overview**
    - Number of Records: 150
    - Features/Variables: 5
    - Independent Features/Variables: sepal_length, sepal_width, petal_length, petal_width (Numerical/Continuous)

- Dependent Feature/Variable: species (Categorical)

2. **Data Integrity**
   - No missing values are observed in any of the columns.
3. **Data Types**
   - Independent features' data types: float/Real numbers
   - Output variable: Categorical Variable
4. **Summary Statistics**
   - **sepal_length**: min: 4.3, 25%: 5.1, mean: 5.8, 50%: 5.8, 75%: 6.4, max: 7.9,
   - **sepal_width**: min: 2.0, 25%: 2.8, mean: 3.1, 50%: 3.0, 75%: 3.3, max: 4.4,
   - **petal_length**: min: 1.0, 25%: 1.6, mean: 3.8, 50%: 4.3, 75%: 5.1, max: 6.9,
   - **petal_width**: min: 0.1, 25%: 0.3, mean: 1.2, 50%: 1.3, 75%: 1.8, max: 2.5,
5. **Observations on Spread and Range**
   - The spread is more pronounced in petal_length.
   - The range is broader in petal_length compared to other features.
6. **Lowest Mean/Median**
   - The lowest mean and median are observed in petal_width.
7. **Petal Width Distribution**
   - The range (25% - 75%) for petal_width is between 0.3 to 1.8 cm, with a median of 1.3 cm.
8. **Petal Length Distribution**
   - The range (25% - 75%) for petal_length is between 1.6 to 5.1 cm, with a median of 4.3 cm.
9. **Sepal Width Distribution**
   - The range (25% - 75%) for sepal_width is between 2.8 to 3.3 cm, with a median of 3.0 cm.
10. **Sepal Length Distribution**
    - The range (25% - 75%) for sepal_length is between 5.1 to 6.4 cm, with a median of 5.8 cm.

8 4. Handling Missing Values - Cat - Variables

[5]:
```
iris_df.isnull().sum()
```

[5]:
```
sepal_length      0
sepal_width       0
petal_length      0
petal_width       0
species_labels    0
species           0
dtype: int64
```

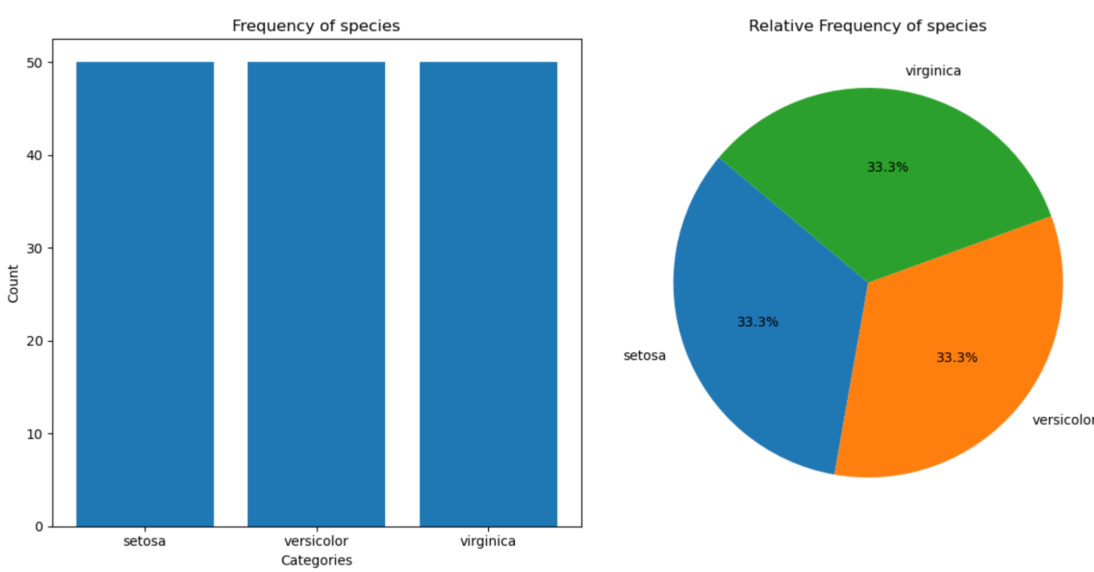
9 5. Categorical- UniVariable - Analysis -Using Pipeline

[6]:
```
class BarPieChartTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        df=X.copy()
        # get cat columns
        cat_cols = df.select_dtypes(include='object').columns
        for cat_name in cat_cols:
            value_counts = df[cat_name].value_counts().reset_index()
            # Rename the columns
            value_counts.columns = ['Class', 'Frequency']

            # Print the result as a table
            print(f"{cat_name} frequency table")
            print(tabulate(value_counts, headers='keys', tablefmt='pretty'))

            # Calculate relative frequency
            total_count = value_counts['Frequency'].sum()
            value_counts['Relative Frequency %'] =␣
↪round((value_counts['Frequency'] / total_count)*100,2)

            # Print the result as a table
            print(f"{cat_name} Relative frequency table")
            print(tabulate(value_counts, headers='keys', tablefmt='pretty'))

            # Extract the values and index from value counts
            value_counts = df[cat_name].value_counts()
            values = value_counts.values
            labels = value_counts.index

            fig, axs = plt.subplots(1, 2, figsize=(12, 6))  # 1 row, 2 columns
            # Create a bar graph
            axs[0].bar(labels, values)
            axs[0].set_title(f'Frequency of {cat_name}')
            axs[0].set_xlabel('Categories')  # Set x-label
            axs[0].set_ylabel('Count')       # Set y-label

            axs[1].pie(value_counts.values, labels=value_counts.index,␣
↪autopct='%1.1f%%', startangle=140)
            axs[1].set_title(f'Relative Frequency of {cat_name}')
            plt.tight_layout()
```

```
    # Show the plot
    plt.show()
```

[7]:
```
pipeline_cat_var = Pipeline([
    ('cat_univaraite_analysis', BarPieChartTransformer())
])

# Fit and transform your data using the pipeline
processed_data = pipeline_cat_var.fit_transform(iris_df)
```

```
species frequency table
+---+------------+-----------+
|   |   Class    | Frequency |
+---+------------+-----------+
| 0 |   setosa   |    50     |
| 1 | versicolor |    50     |
| 2 | virginica  |    50     |
+---+------------+-----------+
species Relative frequency table
+---+------------+-----------+----------------------+
|   |   Class    | Frequency | Relative Frequency % |
+---+------------+-----------+----------------------+
| 0 |   setosa   |    50     |        33.33         |
| 1 | versicolor |    50     |        33.33         |
| 2 | virginica  |    50     |        33.33         |
+---+------------+-----------+----------------------+
```



10 6. Handling Missing Values in Numerical Columns

[8]:
```
iris_df.isnull().sum()
```

[8]:
```
sepal_length      0
sepal_width       0
petal_length      0
petal_width       0
species_labels    0
species           0
dtype: int64
```

[9]:
```
iris_df.describe()
```

[9]:

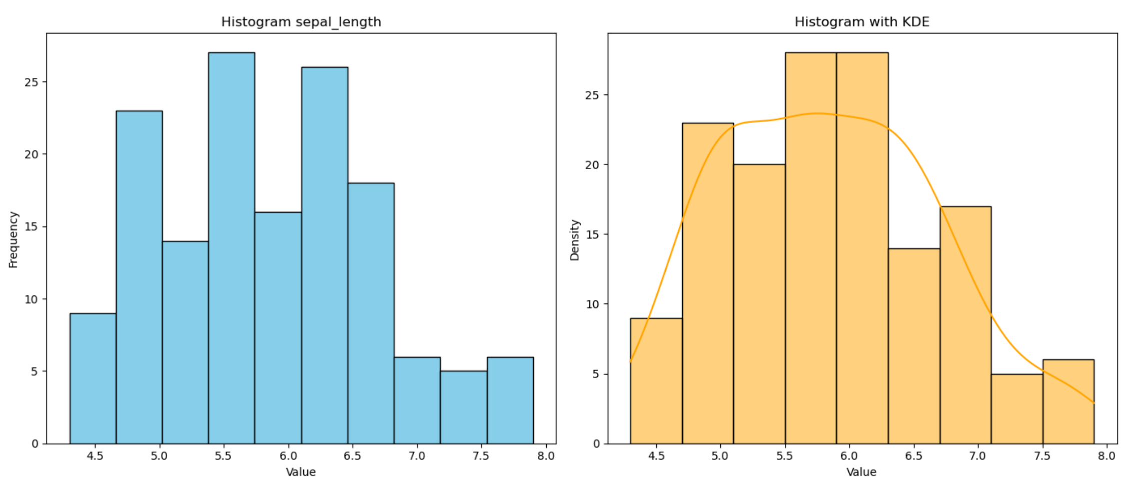
	sepal_length	sepal_width	petal_length	petal_width	species_labels
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

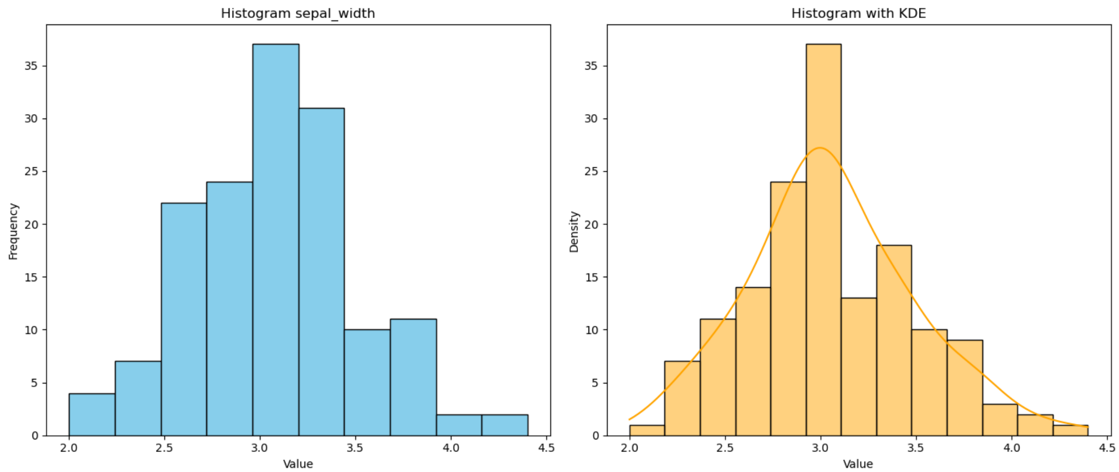
11 7. Numerical - UniVariable - Analysis - Using -Pipeline

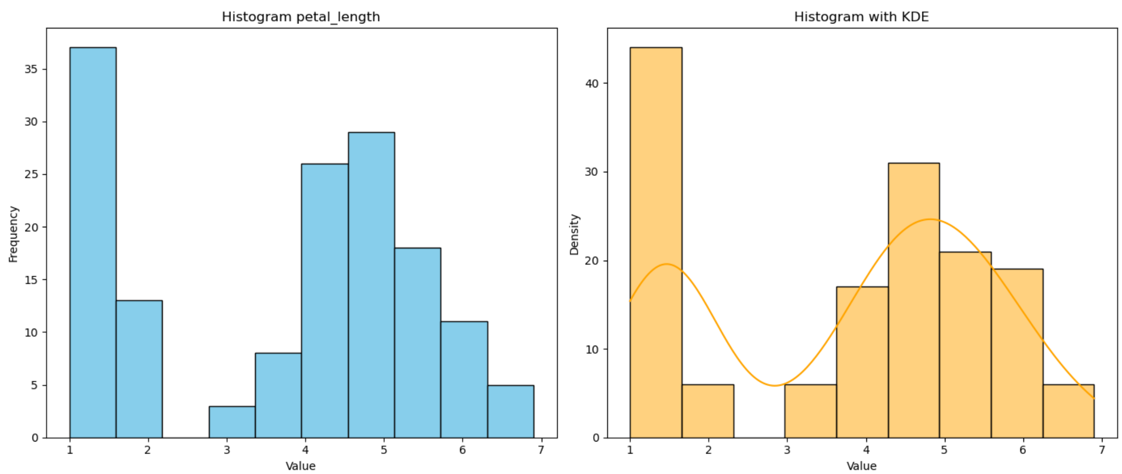
[10]:
```
class HistBoxChartTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        df=X.copy()
        # getting num cols
        num_cols = df.select_dtypes(exclude='object').columns
        for con_var in num_cols:


            # Create a figure and axes object
            fig, axes = plt.subplots(1, 2, figsize=(14, 6))

            # Plot histogram without KDE on the left
            axes[0].hist(df[con_var],  color='skyblue', edgecolor='black')
            axes[0].set_xlabel('Value')
            axes[0].set_ylabel('Frequency')
```

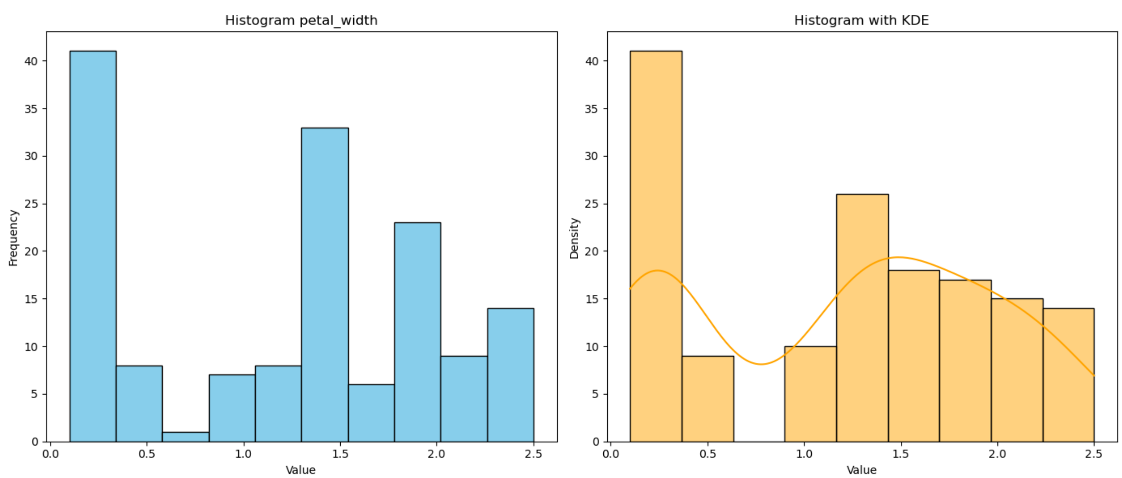
```
            axes[0].set_title(f'Histogram {con_var}')

            # Plot histogram with KDE on the right
            sns.histplot(data=df, x=con_var, kde=True, color='orange',␣
 ↪edgecolor='black', ax=axes[1])
            axes[1].set_xlabel('Value')
            axes[1].set_ylabel('Density')
            axes[1].set_title('Histogram with KDE')

            # Adjust layout
            plt.tight_layout()

            # Show the combined plot
            plt.show()
```

[11]:
```
pipeline_num_var = Pipeline([
    ('num_uni_variate_analysis', HistBoxChartTransformer())
])

iris_num_df = iris_df[['sepal_length', 'sepal_width', 'petal_length',␣
 ↪'petal_width']].copy()
# Fit and transform your data using the pipeline
processed_data = pipeline_num_var.fit_transform(iris_num_df)
```




Histogram sepal_width
Frequency
Value
0
5
10
15
20
25
30
35
2.0
2.5
3.0
3.5
4.0
4.5
Histogram with KDE
Density
Value


Histogram petal_length
Frequency
Value
1
2
3
4
5
6
7
Histogram with KDE
Density
Value
0
10
20
30
40


Histogram petal_width
Frequency
Value
0.0
0.5
1.0
1.5
2.0
2.5
Histogram with KDE
Density
Value

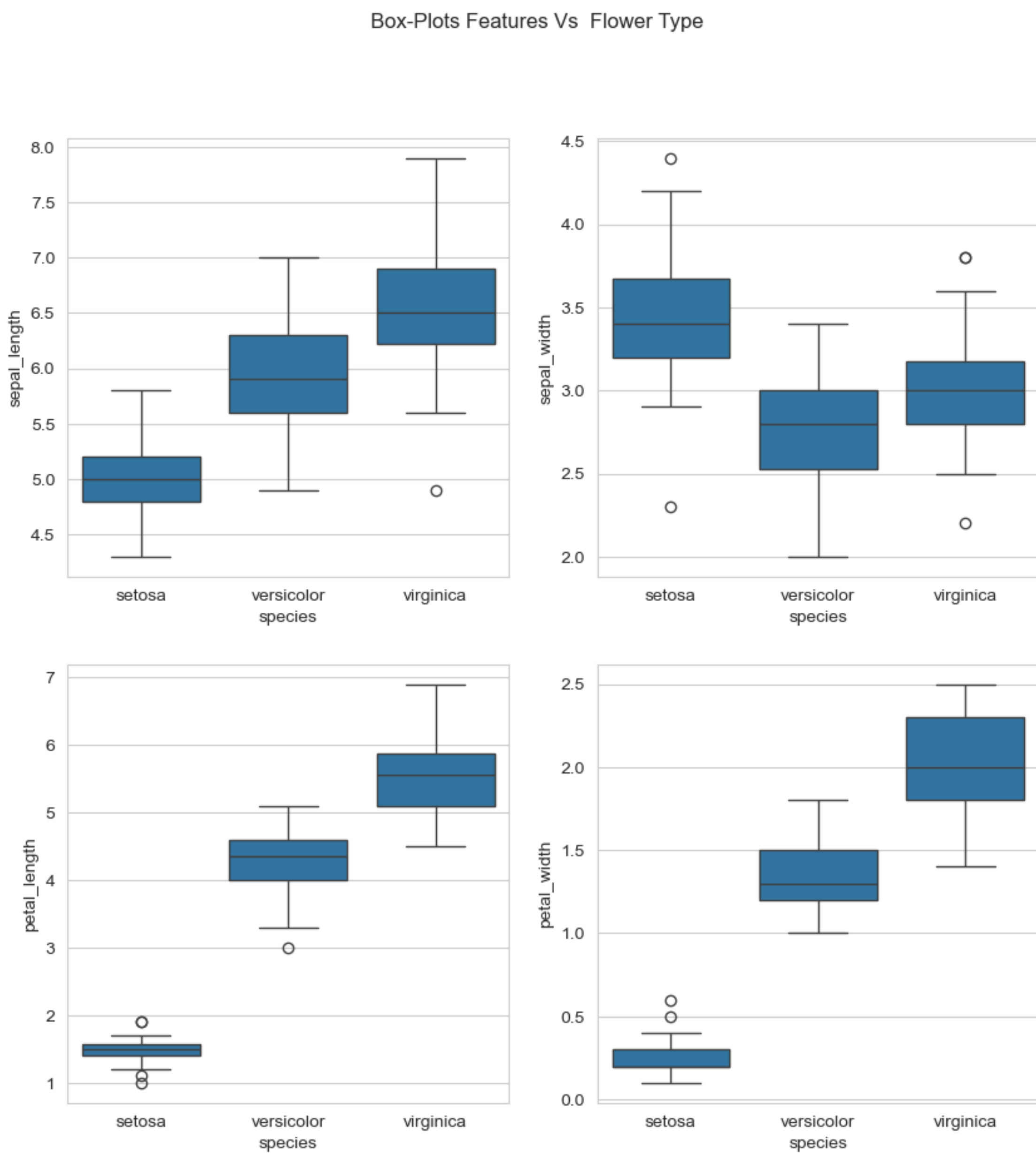
12 8. Numerical - Variables -Outliers Analysis

13 9. Bi Variate Analysis

13.1 9.1 Num Vs cat(target)

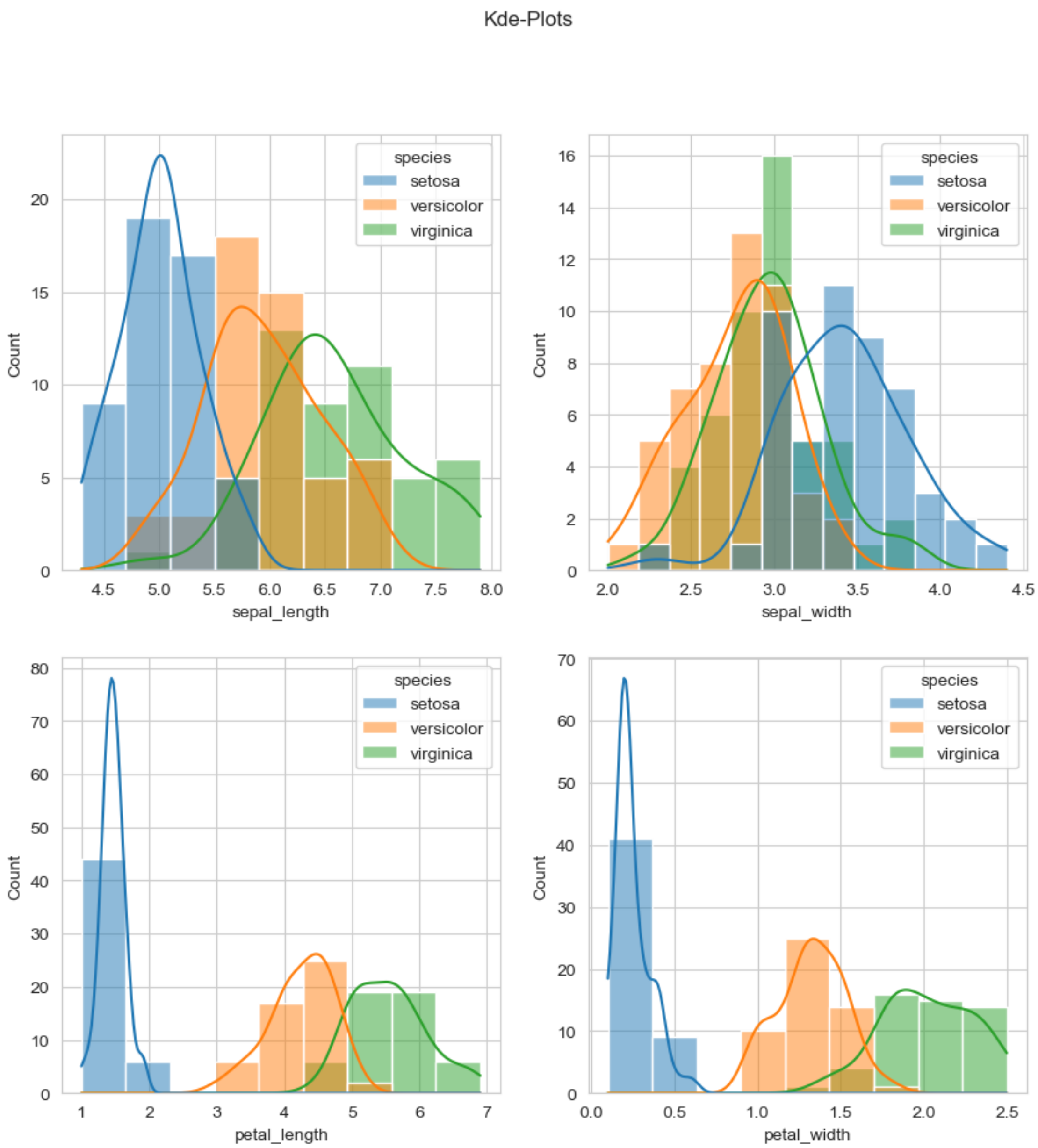
[12]:
```
output_var='species'
```

[13]:
```
sns.set_style("whitegrid")
fig, axes = plt.subplots(2, 2, figsize=(10, 10))
fig.suptitle('Box-Plots Features Vs  Flower Type')
sns.boxplot(ax=axes[0, 0], x=output_var, y='sepal_length', data=iris_df)
sns.boxplot(ax=axes[0, 1], x=output_var, y='sepal_width', data=iris_df)
sns.boxplot(ax=axes[1, 0], x=output_var, y='petal_length', data=iris_df)
sns.boxplot(ax=axes[1, 1], x=output_var, y='petal_width', data=iris_df)
plt.show()
```




```
[14]: sns.set_style("whitegrid")
fig, axes = plt.subplots(2, 2, figsize=(10, 10))
fig.suptitle('Kde-Plots')
sns.histplot(ax=axes[0, 0], hue=output_var, x='sepal_length', data=iris_df,kde=True)
sns.histplot(ax=axes[0, 1], hue=output_var, x='sepal_width', data=iris_df,kde=True)
sns.histplot(ax=axes[1, 0], hue=output_var, x='petal_length', data=iris_df,kde=True)
```

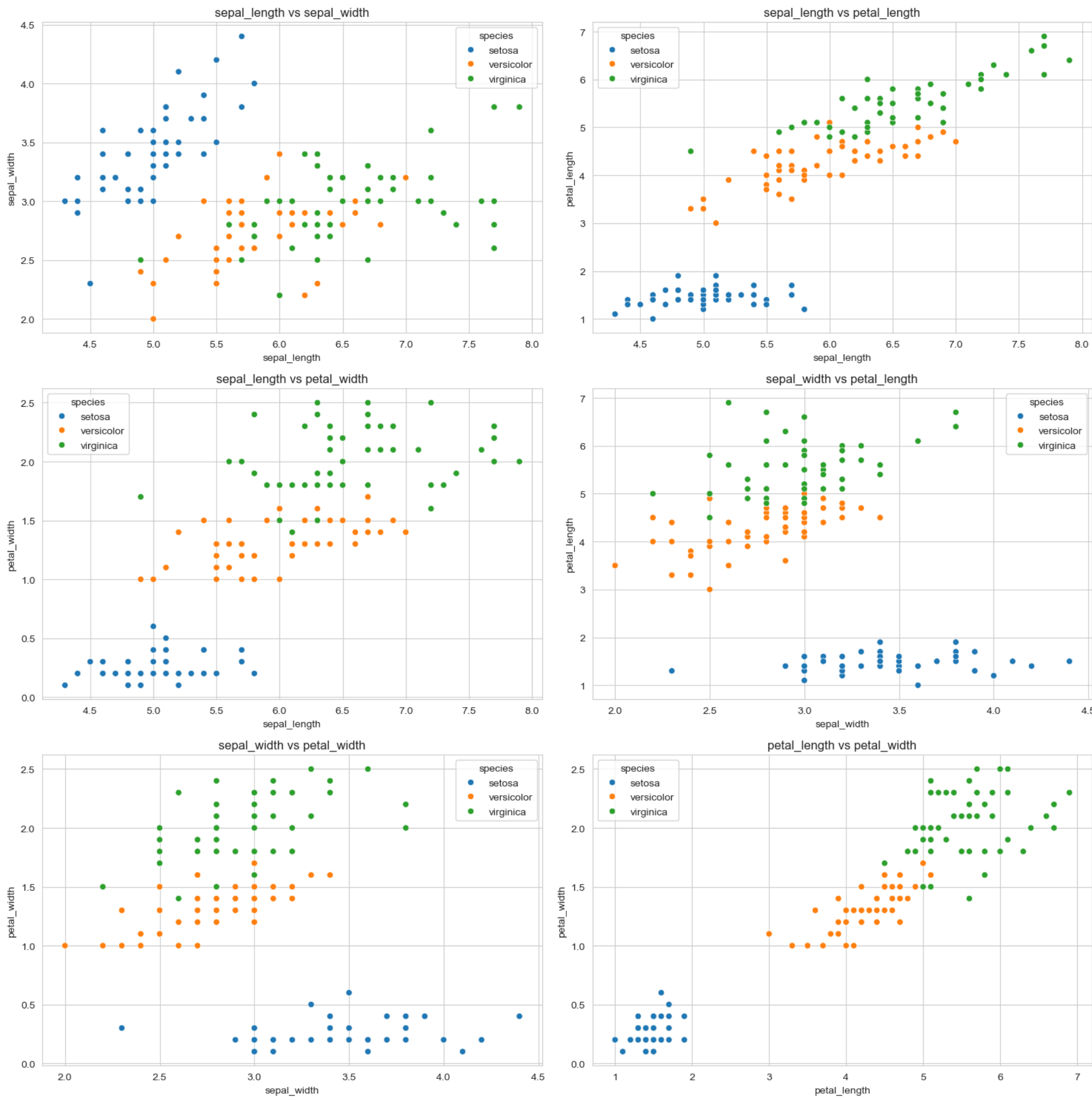
```
sns.histplot(ax=axes[1, 1], hue=output_var, x='petal_width', data=iris_df,kde=True)
plt.show()
```



13.2 9.2 Num Vs Num

[15]:
```
# Selecting only numerical columns
numerical_columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
```

```
# Creating unique scatter plots
num_cols_count = len(numerical_columns)
num_plots = num_cols_count * (num_cols_count - 1) // 2

# Setting up subplots
fig, axes = plt.subplots(num_plots // 2, 2, figsize=(15, 15))

plot_index = 0
for i in range(num_cols_count):
    for j in range(i+1, num_cols_count):
        row = plot_index // 2
        col = plot_index % 2

        # Scatter plot
        sns.scatterplot(x=numerical_columns[i], y=numerical_columns[j],␣
 ↪hue=output_var,data=iris_df, ax=axes[row, col])
        axes[row, col].set_title(f'{numerical_columns[i]} vs␣
 ↪{numerical_columns[j]}')

        plot_index += 1

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

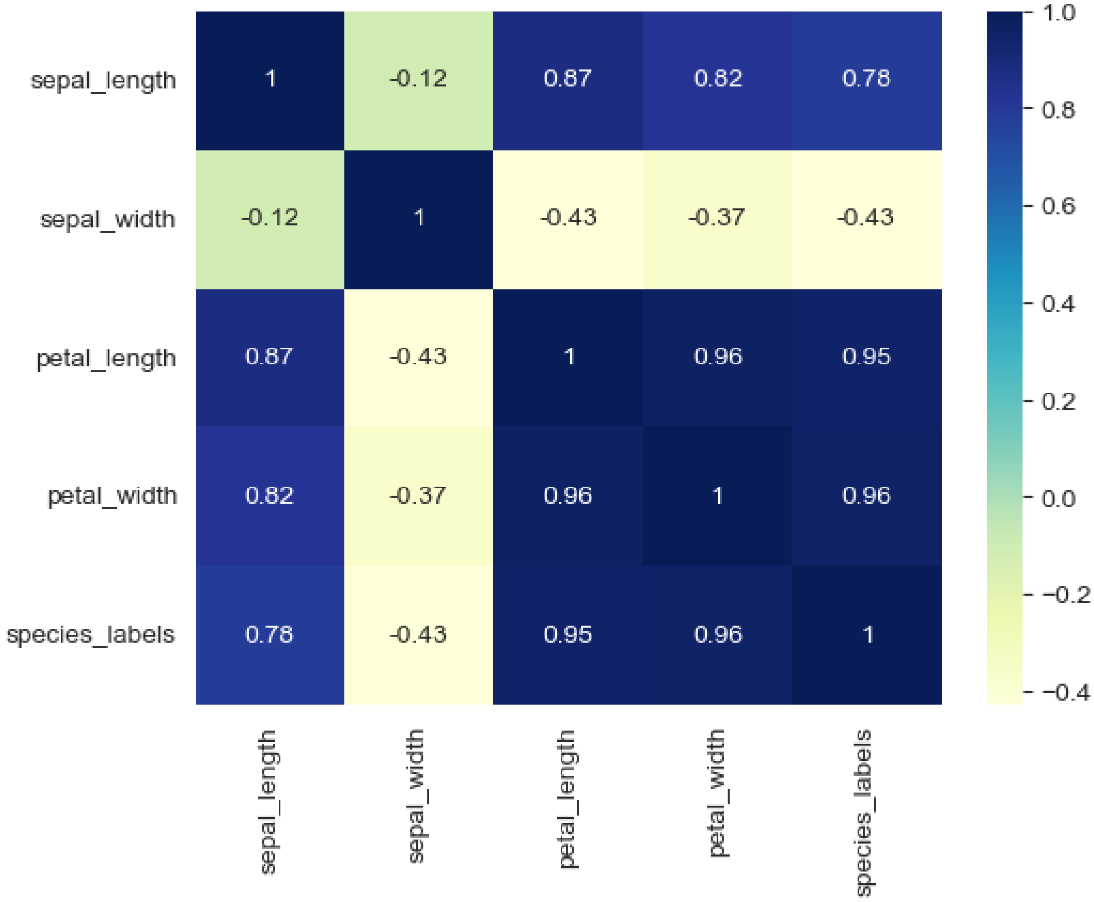


13.3 9.3 Correaltion Numerical Columns

[16]:
```
print(iris_df.corr(numeric_only=True))
sns.heatmap(iris_df.corr(numeric_only=True), cmap="YlGnBu", annot=True)
plt.show()
```

```
               sepal_length  sepal_width  petal_length  petal_width  \
sepal_length       1.000000    -0.117570      0.871754     0.817941
sepal_width       -0.117570     1.000000     -0.428440    -0.366126
petal_length       0.871754    -0.428440      1.000000     0.962865
petal_width        0.817941    -0.366126      0.962865     1.000000
species_labels     0.782561    -0.426658      0.949035     0.956547
```

```
                species_labels
sepal_length          0.782561
sepal_width          -0.426658
petal_length          0.949035
petal_width           0.956547
species_labels        1.000000
```



14 10. Data Transformation

15 11. Standization

[17]:
```
scaler = StandardScaler()
mean_list = []
std_list = []
for var in ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']:
    mean_list.append(iris_df[var].mean())
    std_list.append(iris_df[var].std())
```

```
print(mean_list)
print(std_list)

# Fit and transform the scaler on the selected columns
scaled_columns = scaler.fit_transform(iris_df[['sepal_length', 'sepal_width',␣
↪'petal_length', 'petal_width']])

# Replace the original columns with the scaled columns
iris_df[['sepal_length_stand', 'sepal_width_stand', 'petal_length_stand',␣
↪'petal_width_stand']] = scaled_columns

print(iris_df)
```

```
[5.843333333333334, 3.0573333333333337, 3.7580000000000005, 1.1993333333333336]
[0.8280661279778629, 0.435866284936698, 1.7652982332594667, 0.7622376689603465]
     sepal_length  sepal_width  petal_length  petal_width  species_labels  \
0             5.1          3.5           1.4          0.2               0
1             4.9          3.0           1.4          0.2               0
2             4.7          3.2           1.3          0.2               0
3             4.6          3.1           1.5          0.2               0
4             5.0          3.6           1.4          0.2               0
..            …            …             …            …               …
145           6.7          3.0           5.2          2.3               2
146           6.3          2.5           5.0          1.9               2
147           6.5          3.0           5.2          2.0               2
148           6.2          3.4           5.4          2.3               2
149           5.9          3.0           5.1          1.8               2

       species  sepal_length_stand  sepal_width_stand  petal_length_stand  \
0       setosa           -0.900681           1.019004           -1.340227
1       setosa           -1.143017          -0.131979           -1.340227
2       setosa           -1.385353           0.328414           -1.397064
3       setosa           -1.506521           0.098217           -1.283389
4       setosa           -1.021849           1.249201           -1.340227
..         …                 …                  …                  …
145  virginica            1.038005          -0.131979            0.819596
146  virginica            0.553333          -1.282963            0.705921
147  virginica            0.795669          -0.131979            0.819596
148  virginica            0.432165           0.788808            0.933271
149  virginica            0.068662          -0.131979            0.762758

     petal_width_stand
0            -1.315444
1            -1.315444
2            -1.315444
3            -1.315444
4            -1.315444
```

```
..                …
145          1.448832
146          0.922303
147          1.053935
148          1.448832
149          0.790671

[150 rows x 10 columns]
```

16 12. Convert Cat - to - Numerical Columns

17 ML Models

[18]:
```
Y=iris_df["species_labels"]
X=iris_df[['sepal_length_stand', 'sepal_width_stand', 'petal_length_stand',␣
 ↪'petal_width_stand']]
print(len(Y),len(X))
```

```
150 150
```

[19]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.
 ↪2,random_state = 42)
print(len(X_train),len(X_test))
```

```
120 30
```

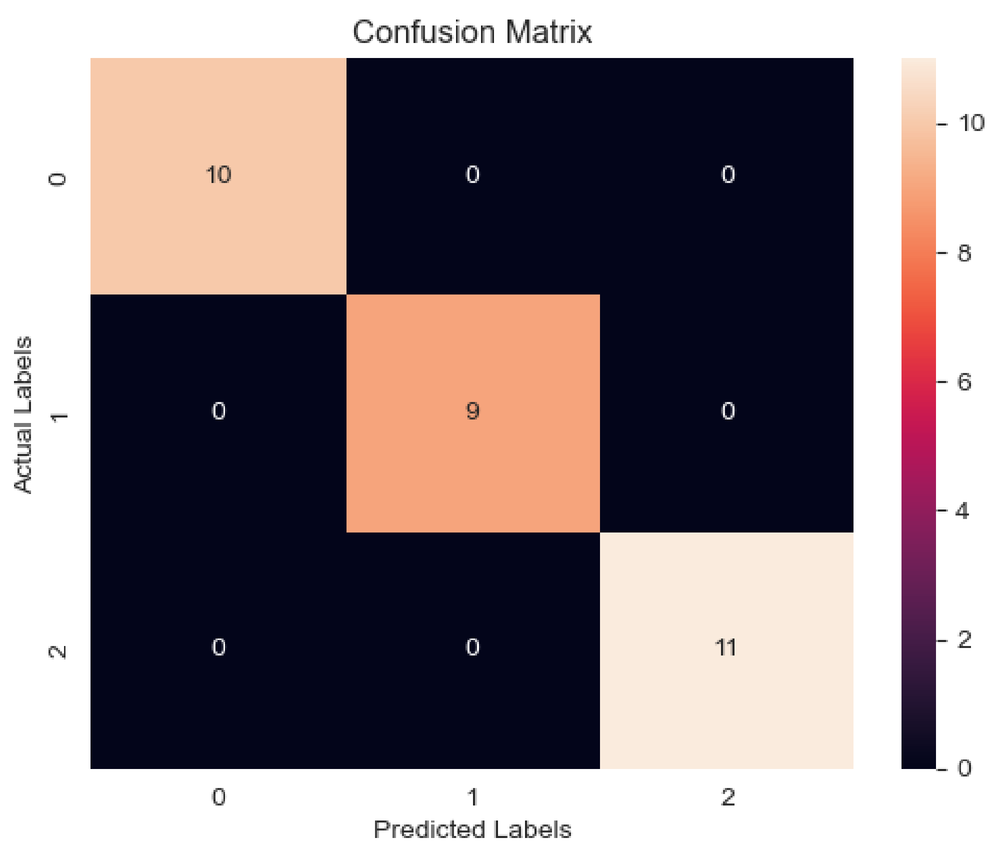
[20]:
```
def draw_heatmap(conf_matrix):
    sns.heatmap(conf_matrix, annot=True)
    plt.xlabel('Predicted Labels')
    plt.ylabel('Actual Labels')
    plt.title('Confusion Matrix')
    plt.show()
```

17.1 Logistic Regression

[21]:
```
lg_model = LogisticRegression(solver='saga', max_iter=500, random_state=42)
lg_model.fit(X_train, Y_train)

print("Model - Logistic Regression")
score = lg_model.score(X_train, Y_train)
print('accuracy train score overall :', score)
score = lg_model.score(X_test, Y_test)
print('accuracy test score overall :', score)

y_pred = lg_model.predict(X_test)
print(classification_report(Y_test, y_pred))
print(confusion_matrix(Y_test, y_pred))
conf_matrix = confusion_matrix(Y_test, y_pred)
```

```
draw_heatmap(conf_matrix)
```

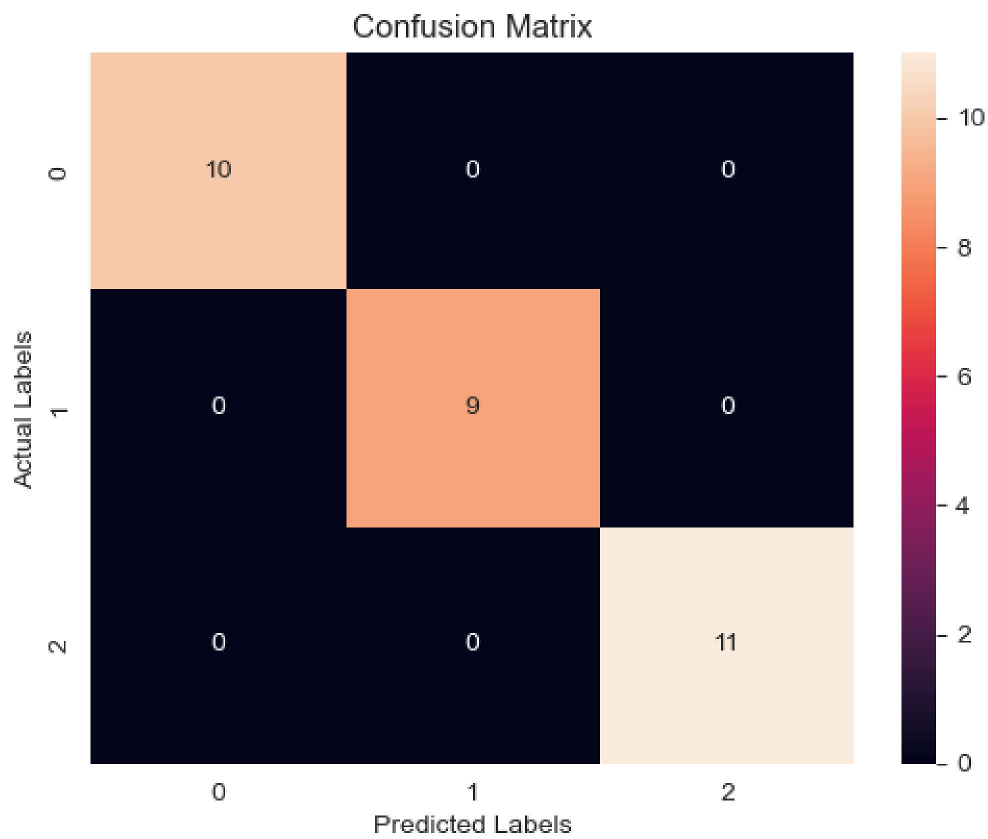
```
Model - Logistic Regression
accuracy train score overall : 0.9666666666666667
accuracy test score overall : 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

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

[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```



17.2 GaussianNB

[22]:
```
from sklearn.naive_bayes import GaussianNB, CategoricalNB
gnb_model = GaussianNB()
gnb_model.fit(X_train,Y_train)

print("Model-GaussianNB")
print("train score",gnb_model.score(X_train,Y_train))
print("test score",gnb_model.score(X_test,Y_test))

y_pred = gnb_model.predict(X_test)
print(classification_report(Y_test, y_pred))
print(confusion_matrix(Y_test, y_pred))
conf_matrix = confusion_matrix(Y_test, y_pred)
draw_heatmap(conf_matrix)
```

```
Model-GaussianNB
train score 0.95
test score 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

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

[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```



17.2.1 Save the model

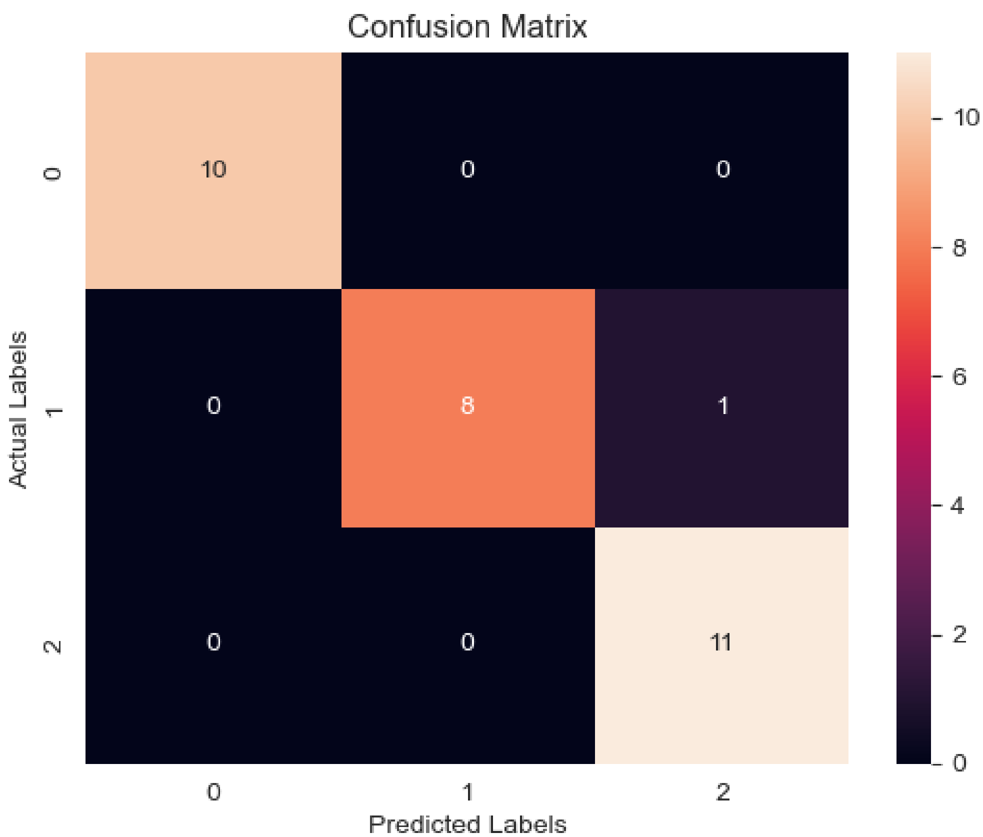
[23]:
```
import pickle
pickle.dump(gnb_model, open('iris-model.pkl', 'wb'))
```

18 Suport Vector Machine - Classifier

[24]:
```
from sklearn.svm import SVC
# Initialize the SVM classifier
svm_linear_classifier = SVC(kernel='linear', random_state=42)

# Train the SVM classifier
svm_linear_classifier.fit(X_train, Y_train)
print("model-Suport Vector Machine - kernel - linear -Classifier")

y_pred = svm_linear_classifier.predict(X_train)
# Calculate the accuracy of the model
accuracy = accuracy_score(Y_train, y_pred)
print("Train Accuracy:", accuracy)
```

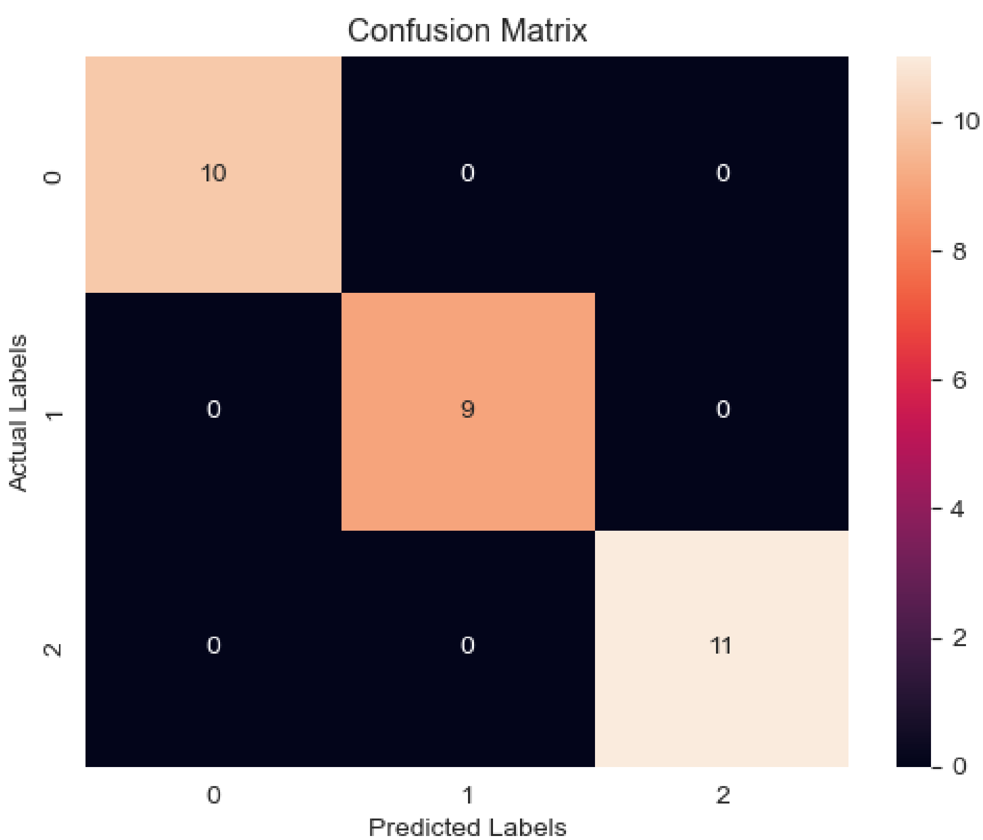
```python
# Predict the classes for test set
y_pred = svm_linear_classifier.predict(X_test)
# Calculate the accuracy of the model
accuracy = accuracy_score(Y_test, y_pred)
print("Test Accuracy:", accuracy)

print(classification_report(Y_test, y_pred))
print(confusion_matrix(Y_test, y_pred))
conf_matrix = confusion_matrix(Y_test, y_pred)
draw_heatmap(conf_matrix)
```

```
model-Suport Vector Machine - kernel - linear -Classifier
Train Accuracy: 0.9833333333333333
Test Accuracy: 0.9666666666666667
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.89      0.94         9
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.96      0.97        30
weighted avg       0.97      0.97      0.97        30

[[10  0  0]
 [ 0  8  1]
 [ 0  0 11]]
```



[25]:
```
svm_rbf_classifier = SVC(kernel='rbf', random_state=42)

# Train the SVM classifier
svm_rbf_classifier.fit(X_train, Y_train)
print("model-Suport Vector Machine - Kernel -rbf - Classifier")
y_pred = svm_rbf_classifier.predict(X_train)
# Calculate the accuracy of the model
accuracy = accuracy_score(Y_train, y_pred)
print("Train Accuracy:", accuracy)

# Predict the classes for test set
y_pred = svm_rbf_classifier.predict(X_test)
# Calculate the accuracy of the model
accuracy = accuracy_score(Y_test, y_pred)
print("Test Accuracy:", accuracy)

print(classification_report(Y_test, y_pred))
print(confusion_matrix(Y_test, y_pred))
conf_matrix = confusion_matrix(Y_test, y_pred)
draw_heatmap(conf_matrix)
```

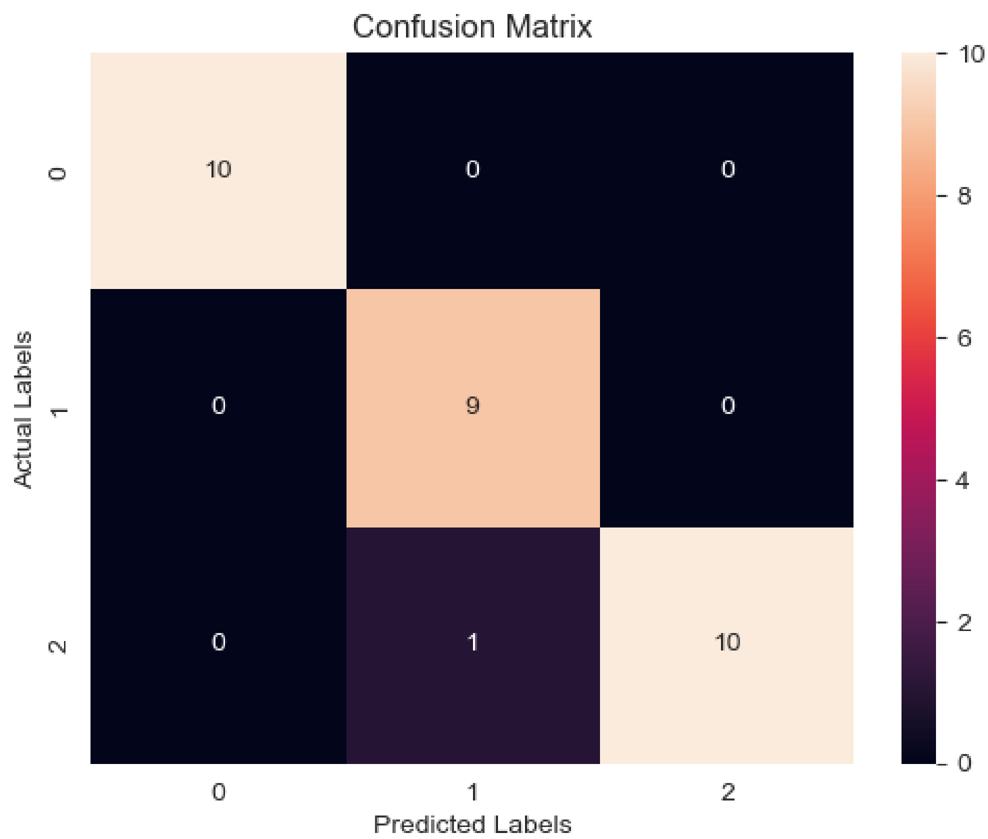
```
model-Suport Vector Machine - Kernel -rbf - Classifier
Train Accuracy: 0.975
Test Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

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

[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```



[26]:
```
svm_poly_classifier = SVC(kernel='poly', random_state=42)

# Train the SVM classifier
```

```python
svm_poly_classifier.fit(X_train, Y_train)
print("model-Suport Vector Machine - Kernel -poly - Classifier")

y_pred = svm_poly_classifier.predict(X_train)
# Calculate the accuracy of the model
accuracy = accuracy_score(Y_train, y_pred)
print("Train Accuracy:", accuracy)

# Predict the classes for test set
y_pred = svm_poly_classifier.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(Y_test, y_pred)
print("Test Accuracy:", accuracy)

print(classification_report(Y_test, y_pred))
print(confusion_matrix(Y_test, y_pred))
conf_matrix = confusion_matrix(Y_test, y_pred)
draw_heatmap(conf_matrix)
```

```
model-Suport Vector Machine - Kernel -poly - Classifier
Train Accuracy: 0.9333333333333333
Test Accuracy: 0.9666666666666667
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.90      1.00      0.95         9
           2       1.00      0.91      0.95        11

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30

[[10  0  0]
 [ 0  9  0]
 [ 0  1 10]]
```

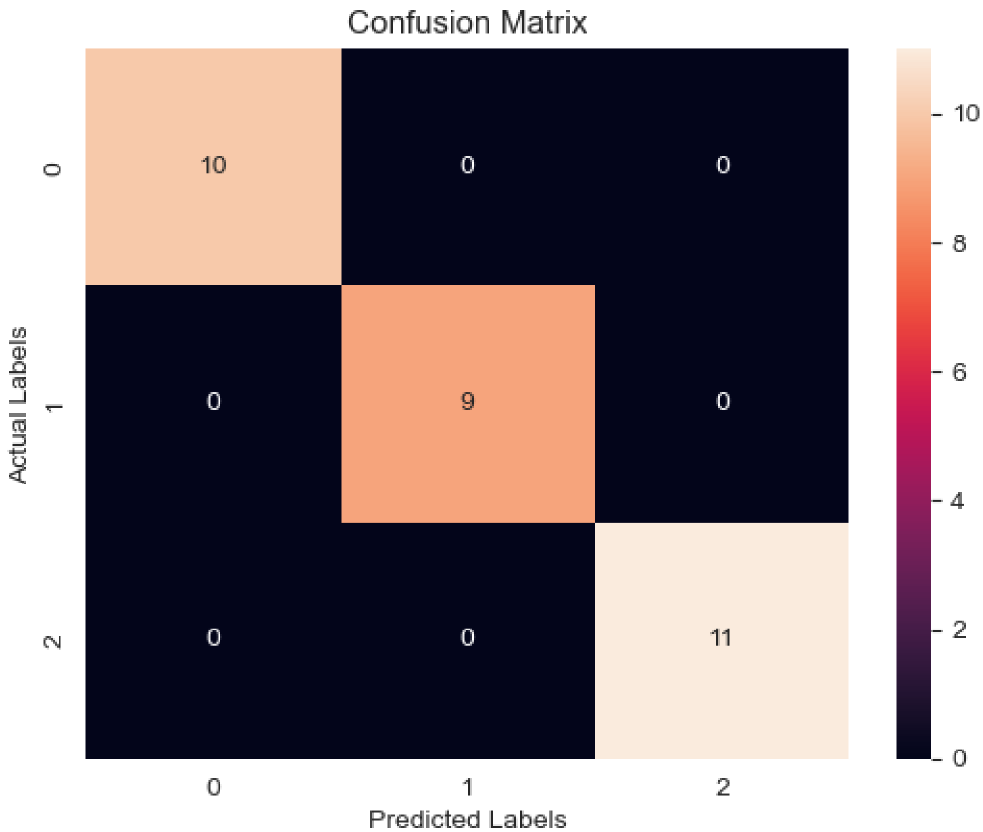


18.1 Decision Tree

[27]:
```
dt_clf = DecisionTreeClassifier(max_leaf_nodes=20,random_state=42)
dt_clf.fit(X_train, Y_train)
print("Model-Decion Tree")

accuracy=dt_clf.score(X_train, Y_train)
print(f"train score: {accuracy}")

accuracy=dt_clf.score(X_test, Y_test)
print(f"test score: {accuracy}")

y_pred=dt_clf.predict(X_test)
print(classification_report(Y_test, y_pred))
print(confusion_matrix(Y_test, y_pred))
conf_matrix = confusion_matrix(Y_test, y_pred)
draw_heatmap(conf_matrix)
```

```
Model-Decion Tree
train score: 1.0
test score: 1.0
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

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

[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

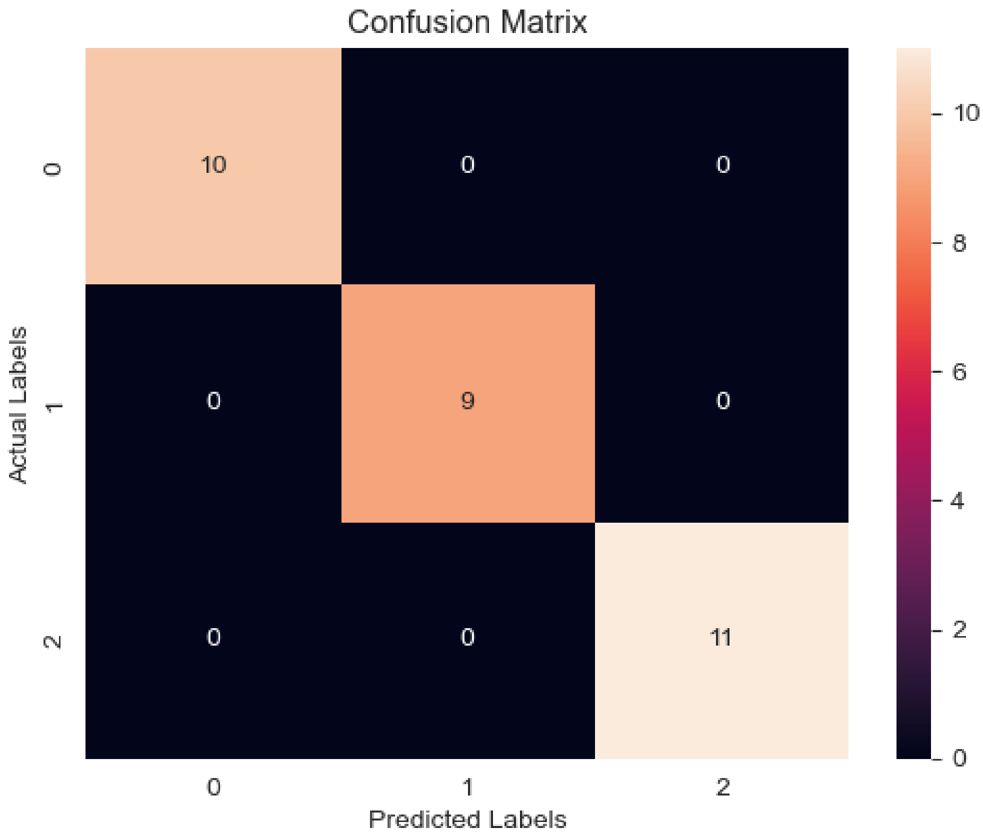


18.2 Random Forest

[28]:
```
rf_clf= RandomForestClassifier(n_estimators = 1000, random_state = 42,␣
    ↪max_leaf_nodes=20)
rf_clf.fit(X_train, Y_train)
print("Model- Random Forest Tree")
```

```python
accuracy=rf_clf.score(X_train, Y_train)
print(f"train score: {accuracy}")

accuracy=rf_clf.score(X_test, Y_test)
print(f"test score: {accuracy}")

y_pred=rf_clf.predict(X_test)
print(classification_report(Y_test, y_pred))
print(confusion_matrix(Y_test, y_pred))
conf_matrix = confusion_matrix(Y_test, y_pred)
draw_heatmap(conf_matrix)
```

```
Model- Random Forest Tree
train score: 1.0
test score: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

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

[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

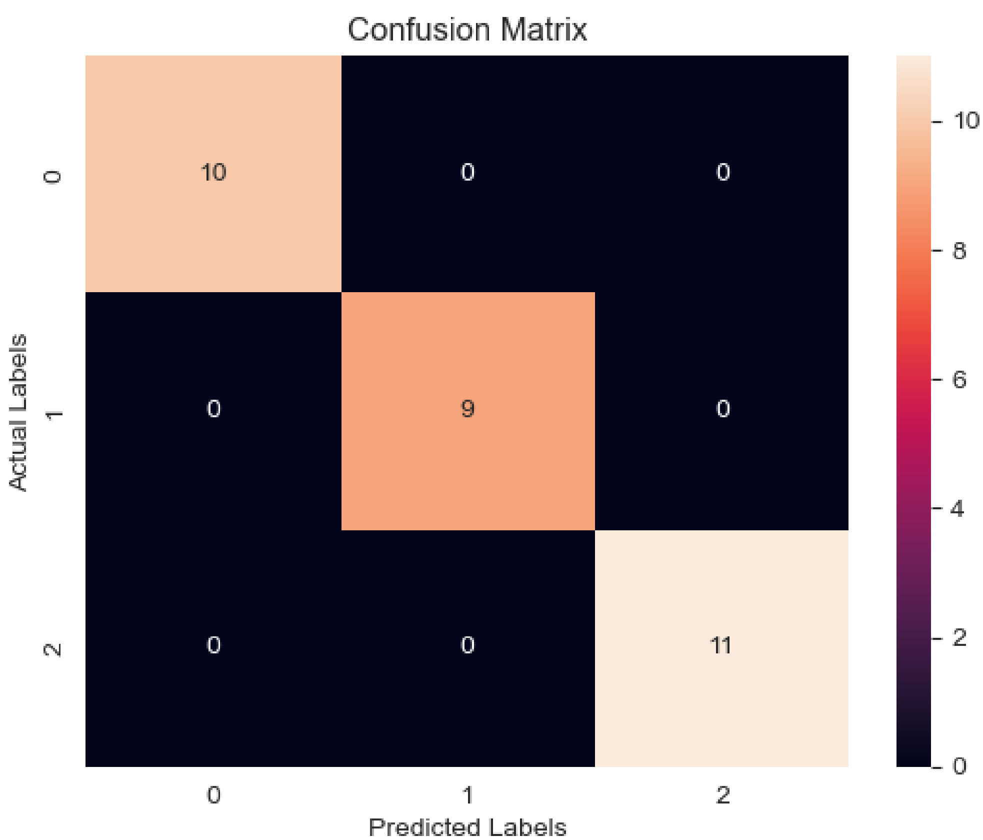


18.3 AdaBoost

[29]:
```
base_classifier = DecisionTreeClassifier(max_depth=1)
adaboost_clf = AdaBoostClassifier( n_estimators=50, random_state=42)

# Train the AdaBoost classifier
adaboost_clf.fit(X_train, Y_train)

print("Model-AdaBoost")
print("train score",adaboost_clf.score(X_train, Y_train))

# Predict on the test set
y_pred = adaboost_clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(Y_test, y_pred)
print(f"test score: {accuracy}")

print(classification_report(Y_test, y_pred))
print(confusion_matrix(Y_test, y_pred))
```

```
conf_matrix = confusion_matrix(Y_test, y_pred)
draw_heatmap(conf_matrix)
```

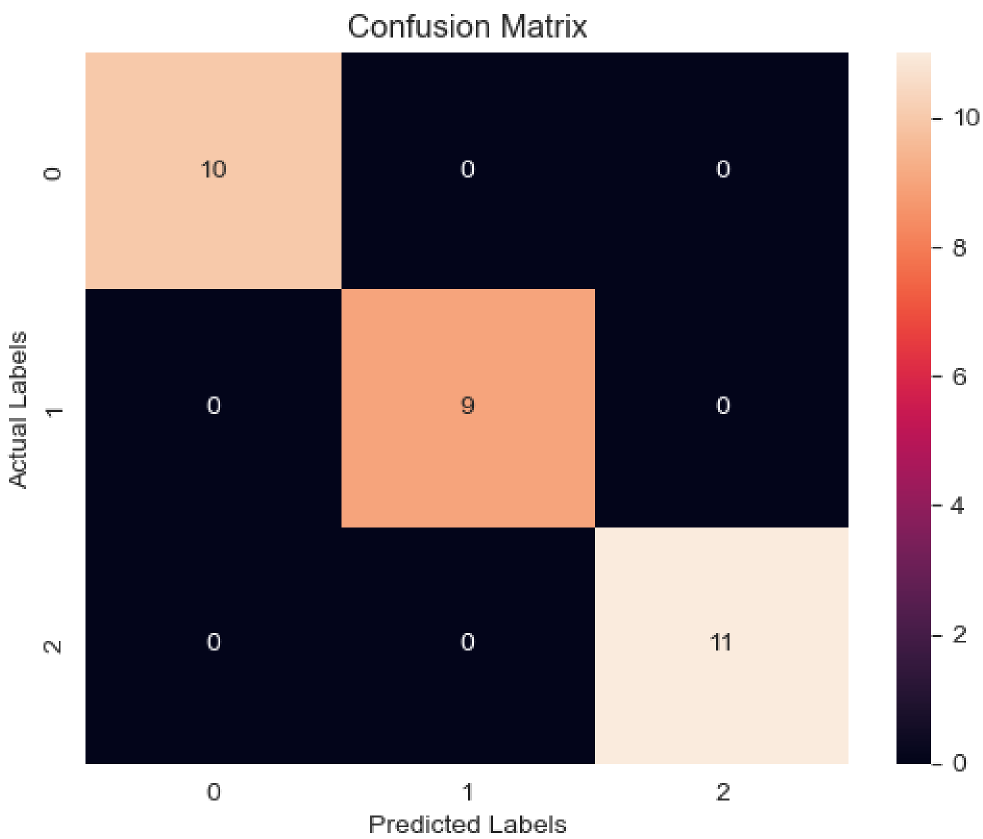
```
Model-AdaBoost
train score 0.9666666666666667
test score: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

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

[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```



18.4 GradientBoostingClassifier

[30]:
```
gdb_clf = GradientBoostingClassifier(n_estimators=20, learning_rate=0.1,
 ↪max_depth=1, random_state=42)
gdb_clf.fit(X_train, Y_train)
print("model-Gradient Boosting Classifier")

accuracy = gdb_clf.score(X_train, Y_train)
print("Train Accuracy:", accuracy)

accuracy = gdb_clf.score(X_test, Y_test)
print("Test Accuracy:", accuracy)

print(classification_report(Y_test, y_pred))
print(confusion_matrix(Y_test, y_pred))
conf_matrix = confusion_matrix(Y_test, y_pred)
draw_heatmap(conf_matrix)
```

```
model-Gradient Boosting Classifier
Train Accuracy: 0.9583333333333334
Test Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

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

[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

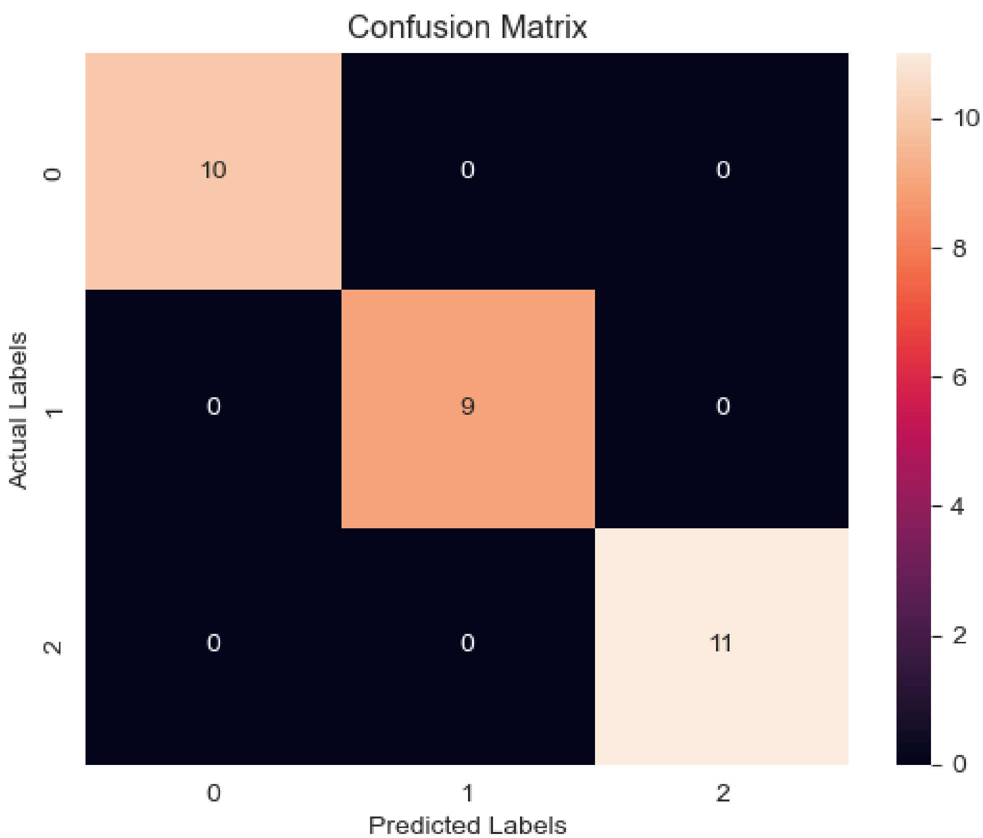


18.5 XGBClassifier

[31]:
```
from xgboost import XGBClassifier
xgmodel = XGBClassifier()
xgmodel.fit(X_train, Y_train)


print("model- XGB Classifier")
# Make predictions on the test set
y_pred = xgmodel.predict(X_train)
accuracy = accuracy_score(Y_train, y_pred)
print("Test Accuracy:", accuracy)
# Evaluate the model

# Make predictions on the test set
y_pred = xgmodel.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
print("Test Accuracy:", accuracy)

print(classification_report(Y_test, y_pred))
```

```python
print(confusion_matrix(Y_test, y_pred))
conf_matrix = confusion_matrix(Y_test, y_pred)
draw_heatmap(conf_matrix)
```

```
model- XGB Classifier
Test Accuracy: 1.0
Test Accuracy: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

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

[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```



[ ]:

```

```