Iris_DataMining

November 25, 2024

1 1. Know The Data

1.1 Import Libraries

[1]:
```
! pip install xgboost
#jupyter nbconvert --to pdf Iris_DataMining.ipynb
```

```
Requirement already satisfied: xgboost in c:\app\python\lib\site-packages
(2.1.1)
Requirement already satisfied: numpy in c:\app\python\lib\site-packages (from
xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\app\python\lib\site-packages (from
xgboost) (1.13.1)


[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

[2]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,␣
↪recall_score, f1_score, classification_report

from sklearn.preprocessing import LabelEncoder

# Import model selection libraries
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪RandomizedSearchCV, RepeatedStratifiedKFold

# Library used for ML Model implementation
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.naive_bayes import GaussianNB
import xgboost as xgb

# Library used for ignore warnings
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

1.2 Dataset Loading

[3]:
```
df = pd.read_csv('Data/Iris.csv')
```

1.3 Data First View

[4]:
```
# Dataset First Look
# View top 5 rews of the dataset
df.head()
```

[4]:

	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

1.4 Dataset Row & Columns count

[5]:
```
print("Number of rows are: ", df.shape[0])
print("Number of columns are: ", df.shape[1])
```

```
Number of rows are:  150
Number of columns are:  6
```

1.5 Dataset Infnormation

[6]:
```
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
```

1.6 Duplicate Values

[7]:
```python
dup = df.duplicated().sum()
print(f'number of duplicated rows are {dup}')
```

```
number of duplicated rows are 0
```

1.7 Missing Values/Null Values

[8]:
```python
df.isnull().sum()
```

[8]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

1.8 What did i know about the dataset?

2 2. Understanding The Variables

[9]:
```python
df.columns
```

[9]:
```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

[10]:
```python
df.describe(include= 'all').round(2)
```

[10]:
```
            Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
count   150.00         150.00        150.00         150.00        150.00
unique     NaN            NaN           NaN            NaN           NaN
top        NaN            NaN           NaN            NaN           NaN
freq       NaN            NaN           NaN            NaN           NaN
mean     75.50           5.84          3.05           3.76          1.20
std      43.45           0.83          0.43           1.76          0.76
min       1.00           4.30          2.00           1.00          0.10
25%      38.25           5.10          2.80           1.60          0.30
50%      75.50           5.80          3.00           4.35          1.30
75%     112.75           6.40          3.30           5.10          1.80
max     150.00           7.90          4.40           6.90          2.50

            Species
```

```
count            150
unique             3
top      Iris-setosa
freq              50
mean             NaN
std              NaN
min              NaN
25%              NaN
50%              NaN
75%              NaN
max              NaN
```

2.1 Check Unique Values for each variable.

[11]:
```
for i in df.columns.tolist():
    print("No. of unique values in ", i, "is ", df[i].nunique())
```

```
No. of unique values in  Id is  150
No. of unique values in  SepalLengthCm is  35
No. of unique values in  SepalWidthCm is  23
No. of unique values in  PetalLengthCm is  43
No. of unique values in  PetalWidthCm is  22
No. of unique values in  Species is  3
```

3 3. Data Wrangling

3.1 Data Wrangling Code

[12]:
```
data = df.iloc[:, 1:]
```

[13]:
```
data.head()
```

[13]:

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

4 4. Data Vizualization, Storytelling & Experimenting with charts : Understand the relationships between variables

Chart - 1 : Distribution of Numerical Variables

[14]:
```
plt.figure(figsize=(8, 6))
plt.suptitle('Distrbution of Iris Flower Measurements', fontsize = 14)
```

```
plt.subplot(2,2,1)
plt.hist(data['SepalLengthCm'])
plt.title('Sepal length Distribution')

plt.subplot(2,2,2)
plt.hist(data['SepalWidthCm'])
plt.title('Sepal width Distribution')

plt.subplot(2,2,3)
plt.hist(data['PetalLengthCm'])
plt.title('Patal length Distribution')

plt.subplot(2,2,4)
plt.hist(data['PetalWidthCm'])
plt.title('Patal width Distribution')

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

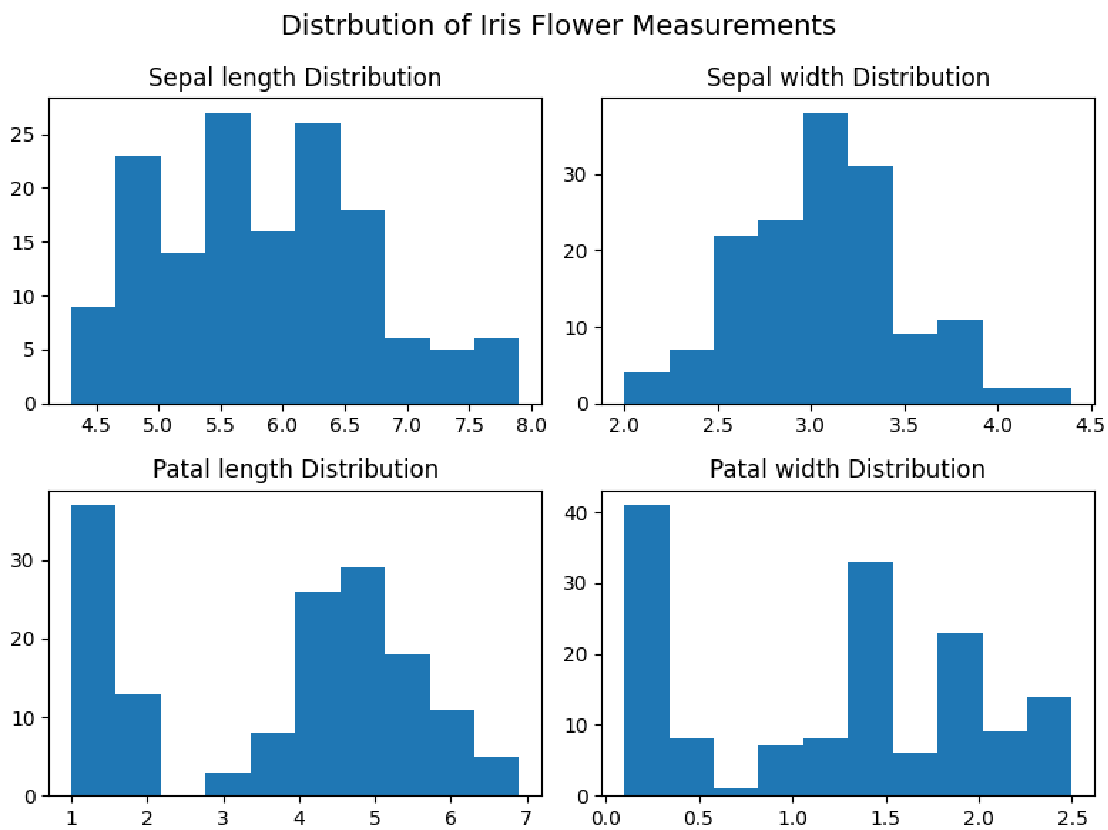


Chart - 2: Sepal Length vs epal Width

[15]:

```
colors = ['red', 'yellow', 'green']
species = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

[16]:

```
# Chart - 2 Scatter plot visualization code for Sepal Length vs Sepal Width.
# Create a scatter plot for Sepal Length vs Sepal Width for each species.
for i in range(3):
    # Select data for the current species.
    x = data[data['Species'] == species[i]]

    # Create a scatter plot with the specified color and label for the current␣
 ↪species.
    plt.scatter(x['SepalLengthCm'], x['SepalWidthCm'], c=colors[i],␣
 ↪label=species[i])

# Add labels to the x and y axes.
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')

# Add a legend to identify species based on colors.
plt.legend()

# Display the scatter plot.
plt.show()
```

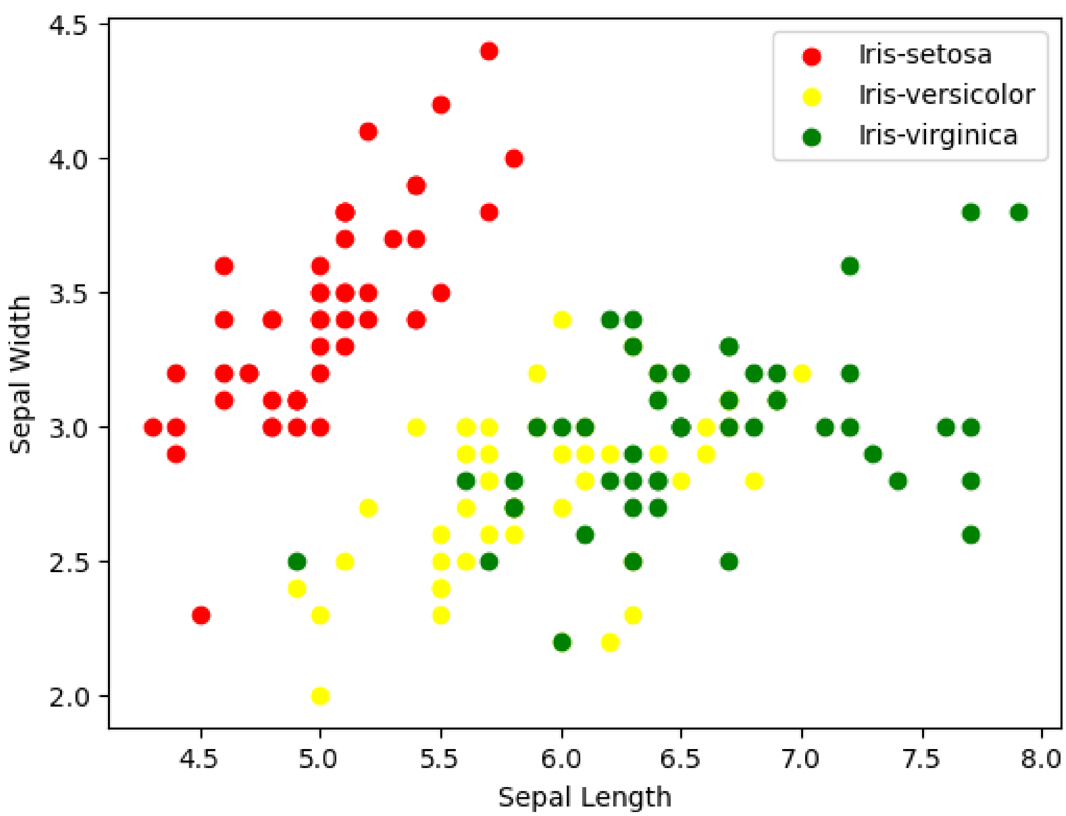


Chart - 3: Petal Length vs Petal Width

[17]:
```
for i in range(3):
    x = data[data['Species'] == species[i]]

    plt.scatter(x['PetalLengthCm'], x['PetalWidthCm'], c = colors[i], label =␣
 ↪species[i])
plt.xlabel("Patal Length")
plt.ylabel("Patal Width")

plt.legend()
plt.show()
```

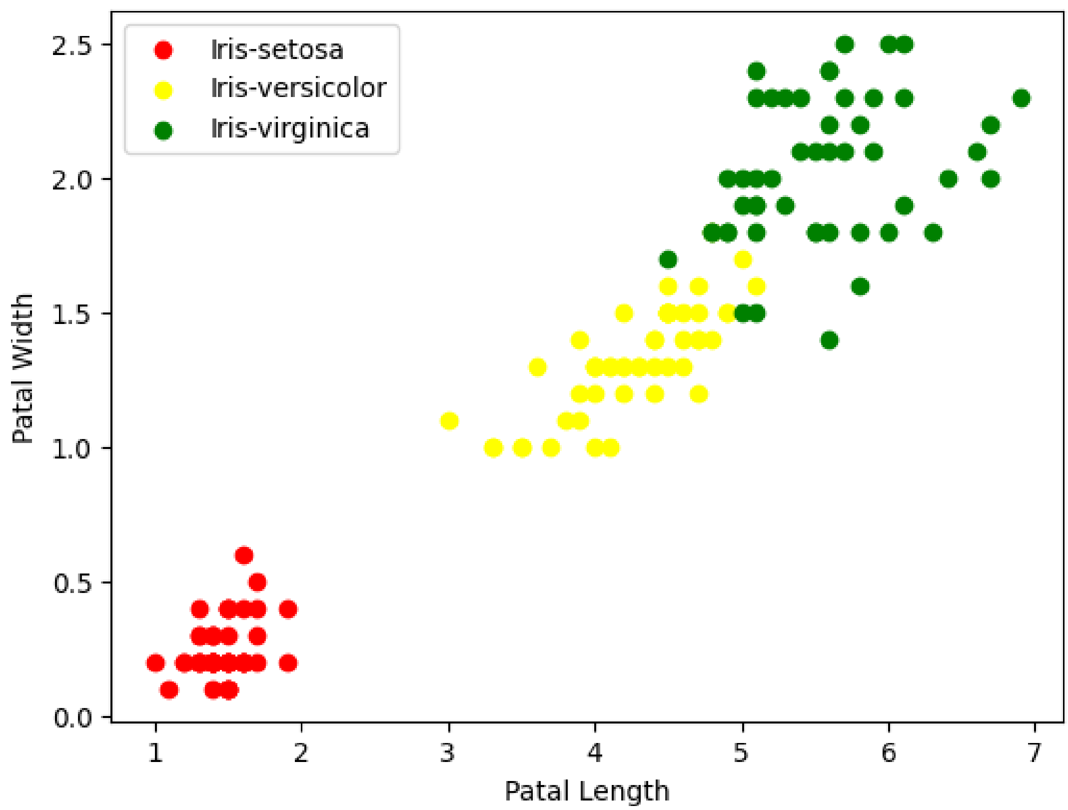


Chart - 4: Sepal Length vs Petal Length

[18]:
```
for i in range(3):

    x = data[data['Species'] == species[i]]

```

```
    plt.scatter(x['SepalLengthCm'], x['PetalLengthCm'], c = colors[i], label =␣
↪species[i])

plt.xlabel('Sepal Length')
plt.ylabel('Petal Length')
plt.legend()
plt.show()
```

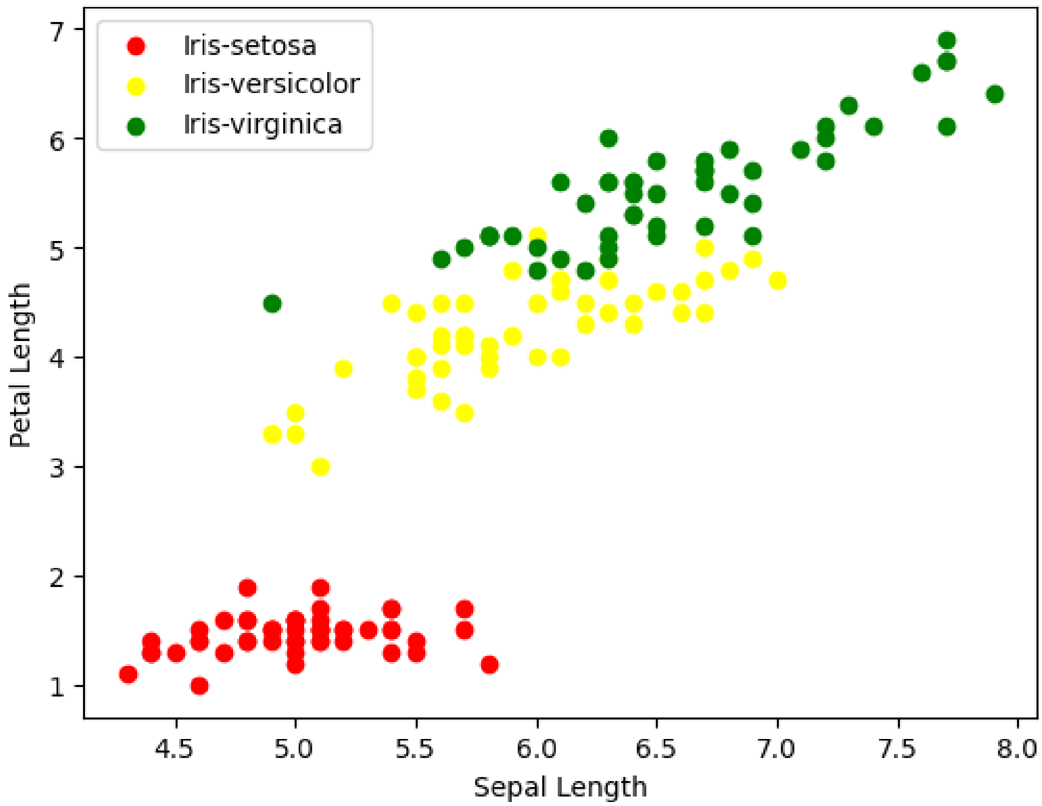


Chart - 5 : Sepal Width vs Petal Width

[19]:
```
for i in range(3):

    x = data[data['Species'] == species[i]]

    plt.scatter(x['SepalWidthCm'], x['PetalWidthCm'], c = colors[i], label =␣
↪species[i])

plt.xlabel('Sepal Width')
plt.ylabel('Petal Width')
plt.legend()
plt.show()
```

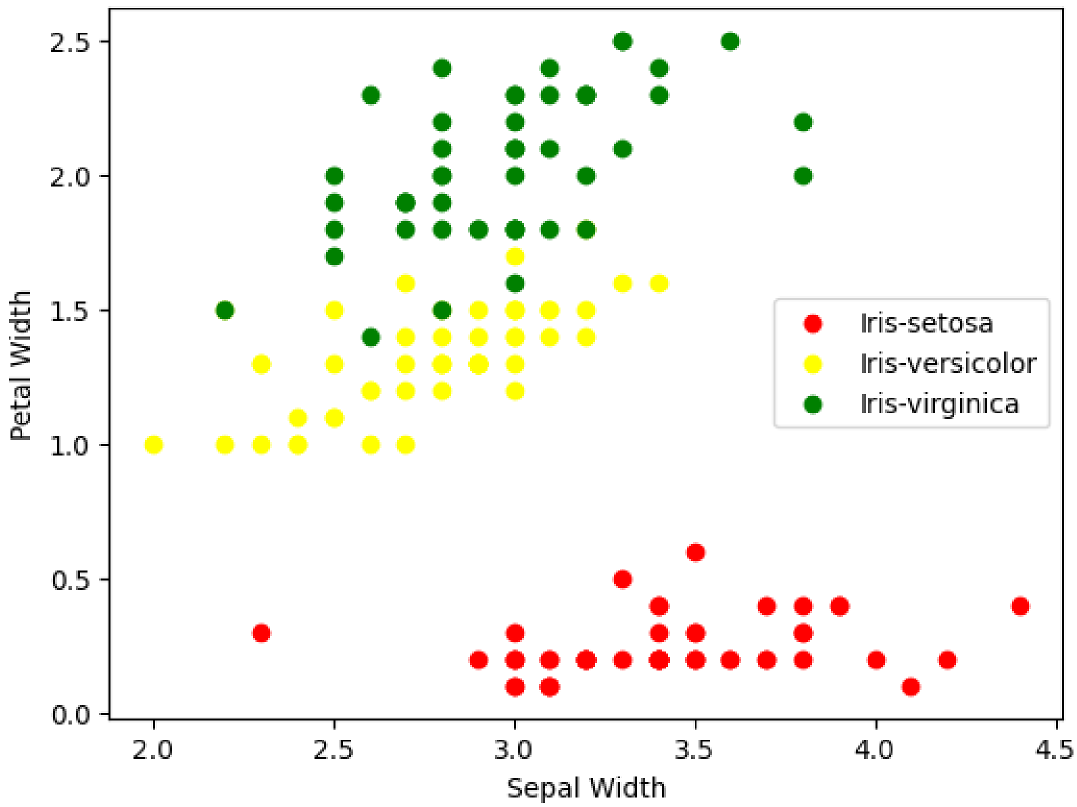


Chart - 6: Correlaton Heatmap

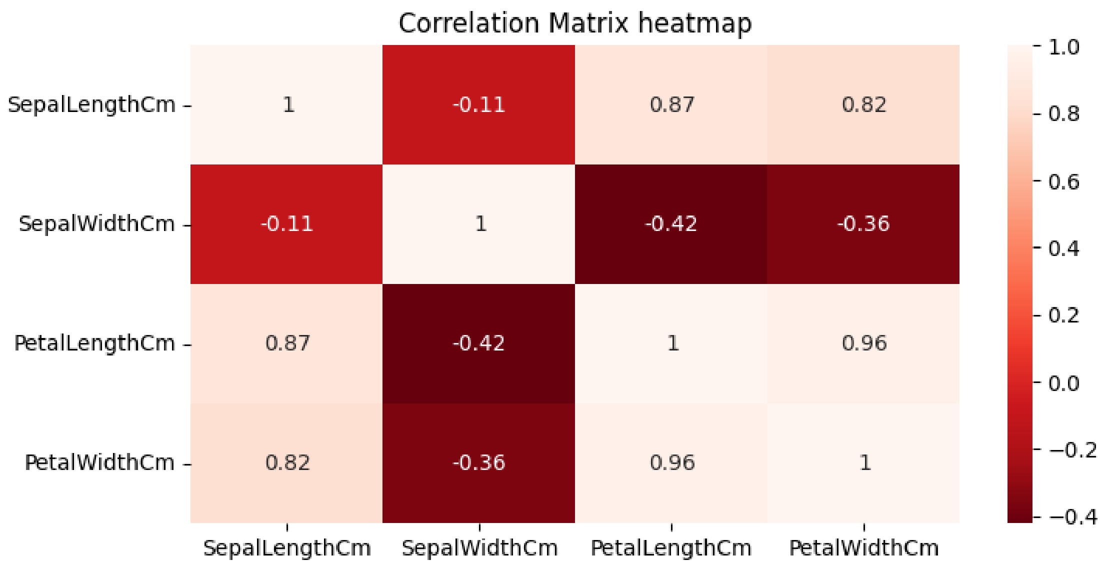
[20]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   SepalLengthCm  150 non-null    float64
 1   SepalWidthCm   150 non-null    float64
 2   PetalLengthCm  150 non-null    float64
 3   PetalWidthCm   150 non-null    float64
 4   Species        150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

[21]:
```
data1 = data.drop('Species', axis= 1)
```

[22]:
```
# Correlation Heatmap Visualization Code
corr_matrix = data1.corr()
```

```python
# Plot Heatmap
plt.figure(figsize=(8, 4))
sns.heatmap(corr_matrix, annot=True, cmap='Reds_r')

# Setting Labels
plt.title('Correlation Matrix heatmap')

# Display Chart
plt.show()
```



5 5. Feature Engineering & Data Pre-processing

5.1 1. Categorical Encoding

[23]:
```python
le = LabelEncoder()
data['Species'] = le.fit_transform(data['Species'])

unique_species = data['Species'].unique()

print("Encoded Species Values: ")
print(unique_species)
```

```
Encoded Species Values:
[0 1 2]
```

5.2 2. Data Scaling

[24]:
```
x=data.drop(columns=['Species'], axis=1)
y=data['Species']
```

5.3 3. Data Splitting

[25]:
```
# Splitting the data to train and test
x_train,x_test,y_train,y_test=train_test_split(x,y, test_size=0.3)
```

[26]:
```
y_train.value_counts()
```

[26]:
```
Species
1    36
2    35
0    34
Name: count, dtype: int64
```

6 6. ML Model Implementation

[27]:
```
def evaluate_model(model, x_train, x_test, y_train, y_test):
    '''The function will take model, x train, x test, y train, y test
    and then it will fit the model, then make predictions on the trained model,
    it will then print roc-auc score of train and test, then plot the roc, auc␣
↪curve,
    print confusion matrix for train and test, then print classification report␣
↪for train and test,
    then plot the feature importances if the model has feature importances,
    and finally it will return the following scores as a list:
    recall_train, recall_test, acc_train, acc_test, F1_train, F1_test
    '''

    # Fit the model to the training data.
    model.fit(x_train, y_train)

    # make predictions on the test data
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)

    # calculate confusion matrix
    cm_train = confusion_matrix(y_train, y_pred_train)
    cm_test = confusion_matrix(y_test, y_pred_test)

    fig, ax = plt.subplots(1, 2, figsize=(11,4))

    print("\nConfusion Matrix:")
```

```python
    sns.heatmap(cm_train, annot=True, xticklabels=['Negative', 'Positive'],␣
↪yticklabels=['Negative', 'Positive'], cmap="Oranges", fmt='.4g', ax=ax[0])
    ax[0].set_xlabel("Predicted Label")
    ax[0].set_ylabel("True Label")
    ax[0].set_title("Train Confusion Matrix")

    sns.heatmap(cm_test, annot=True, xticklabels=['Negative', 'Positive'],␣
↪yticklabels=['Negative', 'Positive'], cmap="Oranges", fmt='.4g', ax=ax[1])
    ax[1].set_xlabel("Predicted Label")
    ax[1].set_ylabel("True Label")
    ax[1].set_title("Test Confusion Matrix")

    plt.tight_layout()
    plt.show()


    # calculate classification report
    cr_train = classification_report(y_train, y_pred_train, output_dict=True)
    cr_test = classification_report(y_test, y_pred_test, output_dict=True)
    print("\nTrain Classification Report:")
    crt = pd.DataFrame(cr_train).T
    print(crt.to_markdown())
    # sns.heatmap(pd.DataFrame(cr_train).T.iloc[:, :-1], annot=True,␣
↪cmap="Blues")
    print("\nTest Classification Report:")
    crt2 = pd.DataFrame(cr_test).T
    print(crt2.to_markdown())
    # sns.heatmap(pd.DataFrame(cr_test).T.iloc[:, :-1], annot=True,␣
↪cmap="Blues")

    precision_train = cr_train['weighted avg']['precision']
    precision_test = cr_test['weighted avg']['precision']

    recall_train = cr_train['weighted avg']['recall']
    recall_test = cr_test['weighted avg']['recall']

    acc_train = accuracy_score(y_true = y_train, y_pred = y_pred_train)
    acc_test = accuracy_score(y_true = y_test, y_pred = y_pred_test)

    F1_train = cr_train['weighted avg']['f1-score']
    F1_test = cr_test['weighted avg']['f1-score']

    model_score = [precision_train, precision_test, recall_train, recall_test,␣
↪acc_train, acc_test, F1_train, F1_test ]
    return model_score
```

```
[28]: # Create a score dataframe
score = pd.DataFrame(index = ['Precision Train', 'Precision Test','Recall␣
↪Train','Recall Test','Accuracy Train', 'Accuracy Test', 'F1 macro Train',␣
↪'F1 macro Test'])
```

6.1 ML Model - 1: Logisic Regression

```
[29]: # ML Model - 1 Implementation
lr_model = LogisticRegression(fit_intercept=True, max_iter=10000)
```

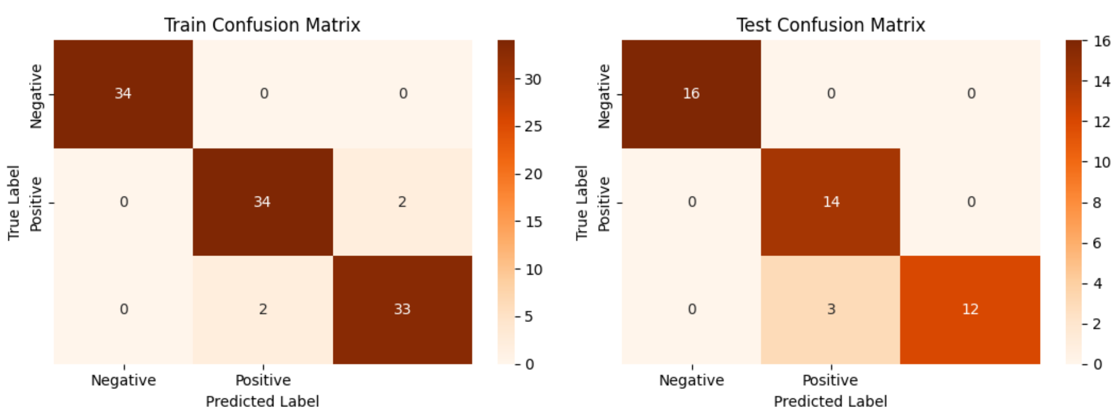
6.1.1 1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

```
[30]: ! pip install tabulate
```

```
Requirement already satisfied: tabulate in c:\app\python\lib\site-packages
(0.9.0)


[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
[31]: # Visualizing evaluation Metric Score chart
lr_score = evaluate_model(lr_model, x_train, x_test, y_train, y_test)
```

```
Confusion Matrix:
```




```
Train Classification Report:
|              |   precision |   recall |   f1-score |   support |
|:-------------|------------:|---------:|-----------:|----------:|
| 0            |     1       | 1        |   1        |  34       |
```

```
| 1            |     0.944444 | 0.944444 |   0.944444 |   36       |
| 2            |     0.942857 | 0.942857 |   0.942857 |   35       |
| accuracy     |     0.961905 | 0.961905 |   0.961905 |    0.961905 |
| macro avg    |     0.962434 | 0.962434 |   0.962434 | 105        |
| weighted avg |     0.961905 | 0.961905 |   0.961905 | 105        |

Test Classification Report:
|              |   precision |    recall |   f1-score |   support |
|:-------------|------------:|----------:|-----------:|----------:|
| 0            |    1        | 1        |   1        | 16        |
| 1            |    0.823529 | 1        |   0.903226 | 14        |
| 2            |    1        | 0.8      |   0.888889 | 15        |
| accuracy     |    0.933333 | 0.933333 |   0.933333 |  0.933333 |
| macro avg    |    0.941176 | 0.933333 |   0.930705 | 45        |
| weighted avg |    0.945098 | 0.933333 |   0.932855 | 45        |
```

[32]:
```python
# Updated Evaluation metric Score Chart
score['Logistic regression'] = lr_score
score
```

[32]:
```
                 Logistic regression
Precision Train             0.961905
Precision Test              0.945098
Recall Train                0.961905
Recall Test                 0.933333
Accuracy Train              0.961905
Accuracy Test               0.933333
F1 macro Train              0.961905
F1 macro Test               0.932855
```

6.1.2 2. Cross- Validation & Hyperparameter Tuning

[33]:
```python
# ML Model - 1 Implementation with hyperparameter optimization techniques (i.e.
↪, GridSearch CV, RandomSearch CV, Bayesian Optimization etc.)
# Define the hyperparameter grid
param_grid = {'C': [100,10,1,0.1,0.01,0.001,0.0001],
              'penalty': ['l1', 'l2'],
              'solver':['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']}

# Initializing the logistic regression model
logreg = LogisticRegression(fit_intercept=True, max_iter=10000, random_state=0)

# Repeated stratified kfold
rskf = RepeatedStratifiedKFold(n_splits=3, n_repeats=4, random_state=0)

# Using GridSearchCV to tune the hyperparameters using cross-validation
grid = GridSearchCV(logreg, param_grid, cv=rskf)
```

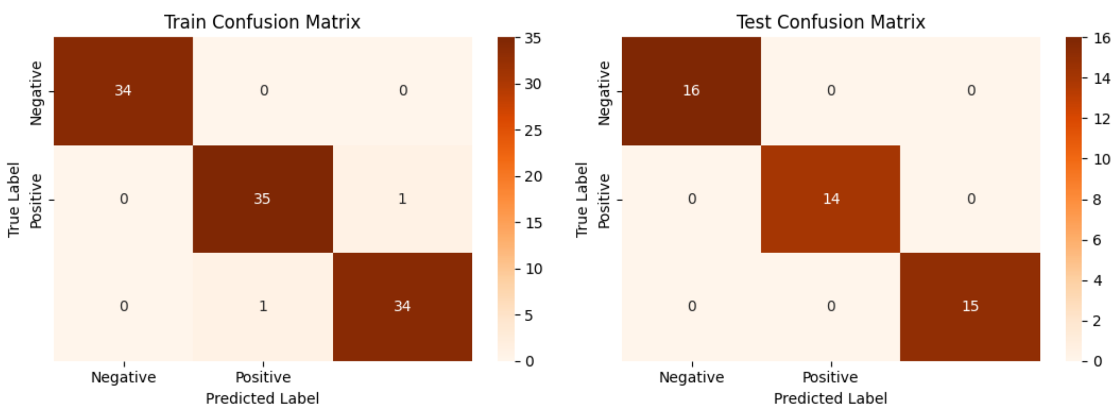
```python
grid.fit(x_train, y_train)

# Select the best hyperparameters found by GridSearchCV
best_params = grid.best_params_
print("Best hyperparameters: ", best_params)
```

```
Best hyperparameters:  {'C': 10, 'penalty': 'l2', 'solver': 'sag'}
```

[34]:
```python
# Initiate model with best parameters
lr_model2 = LogisticRegression(C=best_params['C'],
                               penalty=best_params['penalty'],
                               solver=best_params['solver'],
                               max_iter=10000, random_state=0)
```

[35]:
```python
# Visualizing evaluation Metric Score chart
lr_score2 = evaluate_model(lr_model2, x_train, x_test, y_train, y_test)
```

Confusion Matrix:



Train Classification Report:

	precision	recall	f1-score	support
0	1	1	1	34
1	0.972222	0.972222	0.972222	36
2	0.971429	0.971429	0.971429	35
accuracy	0.980952	0.980952	0.980952	0.980952
macro avg	0.981217	0.981217	0.981217	105
weighted avg	0.980952	0.980952	0.980952	105

Test Classification Report:

	precision	recall	f1-score	support

```
| 0            |          1 |          1 |          1 |         16 |
| 1            |          1 |          1 |          1 |         14 |
| 2            |          1 |          1 |          1 |         15 |
| accuracy     |          1 |          1 |          1 |          1 |
| macro avg    |          1 |          1 |          1 |         45 |
| weighted avg |          1 |          1 |          1 |         45 |
```

[36]:
```python
score['Logistic regression tuned'] = lr_score2
```

[37]:
```python
# Updated Evaluation metric Score Chart
score
```

[37]:

	Logistic regression	Logistic regression tuned
Precision Train	0.961905	0.980952
Precision Test	0.945098	1.000000
Recall Train	0.961905	0.980952
Recall Test	0.933333	1.000000
Accuracy Train	0.961905	0.980952
Accuracy Test	0.933333	1.000000
F1 macro Train	0.961905	0.980952
F1 macro Test	0.932855	1.000000

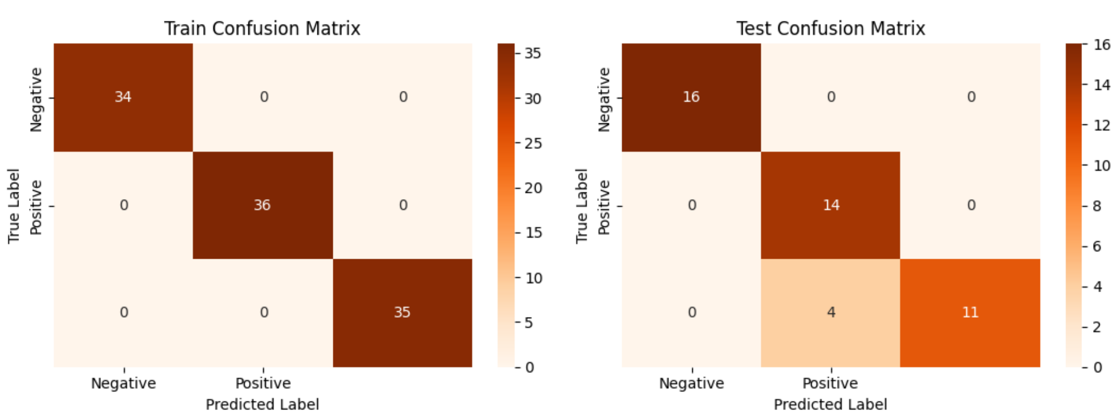
6.2 ML Model - 2 : Decision Tree

[38]:
```python
# ML Model - 2 Implementation
dt_model = DecisionTreeClassifier(random_state=20)
```

6.2.1 1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

[39]:
```python
# Visualizing evaluation Metric Score chart
dt_score = evaluate_model(dt_model, x_train, x_test, y_train, y_test)
```

Confusion Matrix:



```
Train Classification Report:
|              |   precision |    recall |   f1-score |   support |
|:-------------|------------:|----------:|-----------:|----------:|
| 0            |           1 |         1 |          1 |        34 |
| 1            |           1 |         1 |          1 |        36 |
| 2            |           1 |         1 |          1 |        35 |
| accuracy     |           1 |         1 |          1 |         1 |
| macro avg    |           1 |         1 |          1 |       105 |
| weighted avg |           1 |         1 |          1 |       105 |

Test Classification Report:
|              |   precision |    recall |   f1-score |   support |
|:-------------|------------:|----------:|-----------:|----------:|
| 0            |    1        | 1         |   1        | 16        |
| 1            |    0.777778 | 1         |   0.875    | 14        |
| 2            |    1        | 0.733333  |   0.846154 | 15        |
| accuracy     |    0.911111 | 0.911111  |   0.911111 |  0.911111 |
| macro avg    |    0.925926 | 0.911111  |   0.907051 | 45        |
| weighted avg |    0.930864 | 0.911111  |   0.909829 | 45        |
```

[40]:
```
# Updated Evaluation metric Score Chart
score['Decision Tree'] = dt_score
score
```

[40]:

	Logistic regression	Logistic regression tuned	Decision Tree
Precision Train	0.961905	0.980952	1.000000
Precision Test	0.945098	1.000000	0.930864
Recall Train	0.961905	0.980952	1.000000
Recall Test	0.933333	1.000000	0.911111
Accuracy Train	0.961905	0.980952	1.000000
Accuracy Test	0.933333	1.000000	0.911111
F1 macro Train	0.961905	0.980952	1.000000
F1 macro Test	0.932855	1.000000	0.909829

6.2.2 2. Cross- Validation & Hyperparameter Tuning

[41]:
```
# ML Model - 2 Implementation with hyperparameter optimization techniques (i.e.
↪, GridSearch CV, RandomSearch CV, Bayesian Optimization etc.)
# Define the hyperparameter grid
grid = {'max_depth' : [3,4,5,6,7,8],
        'min_samples_split' : np.arange(2,8),
        'min_samples_leaf' : np.arange(10,20)}

# Initialize the model
```

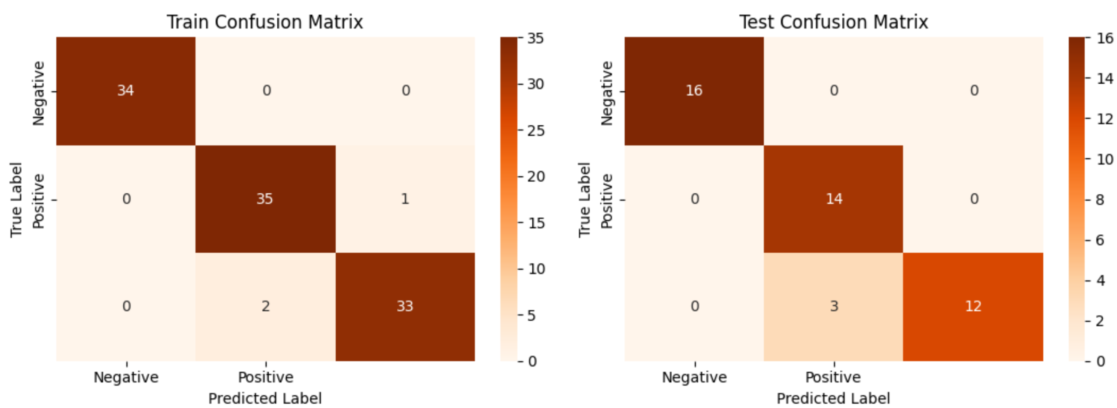
```python
model = DecisionTreeClassifier()

# repeated stratified kfold
rskf = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=0)

# Initialize GridSearchCV
grid_search = GridSearchCV(model, grid, cv=rskf)

# Fit the GridSearchCV to the training data
grid_search.fit(x_train, y_train)

# Select the best hyperparameters
best_params = grid_search.best_params_
print("Best hyperparameters: ", best_params)
```

```
Best hyperparameters:  {'max_depth': 3, 'min_samples_leaf': 10,
'min_samples_split': 2}
```

[42]:
```python
# Train a new model with the best hyperparameters
dt_model2 = DecisionTreeClassifier(max_depth=best_params['max_depth'],
                                                                    ␣
 ↪min_samples_leaf=best_params['min_samples_leaf'],
                                                                    ␣
 ↪min_samples_split=best_params['min_samples_split'],
                                   random_state=20)
```

[43]:
```python
# Visualizing evaluation Metric Score chart
dt2_score = evaluate_model(dt_model2, x_train, x_test, y_train, y_test)
```

```
Confusion Matrix:
```




```
Train Classification Report:
```

```
|              |   precision |   recall |   f1-score |    support |
|:-------------|------------:|---------:|-----------:|-----------:|
| 0            |    1        | 1        |   1        |  34        |
| 1            |    0.945946 | 0.972222 |   0.958904 |  36        |
| 2            |    0.970588 | 0.942857 |   0.956522 |  35        |
| accuracy     |    0.971429 | 0.971429 |   0.971429 |   0.971429 |
| macro avg    |    0.972178 | 0.971693 |   0.971809 | 105        |
| weighted avg |    0.971663 | 0.971429 |   0.971417 | 105        |

Test Classification Report:
|              |   precision |   recall |   f1-score |   support |
|:-------------|------------:|---------:|-----------:|----------:|
| 0            |    1        | 1        |   1        | 16        |
| 1            |    0.823529 | 1        |   0.903226 | 14        |
| 2            |    1        | 0.8      |   0.888889 | 15        |
| accuracy     |    0.933333 | 0.933333 |   0.933333 |  0.933333 |
| macro avg    |    0.941176 | 0.933333 |   0.930705 | 45        |
| weighted avg |    0.945098 | 0.933333 |   0.932855 | 45        |
```

[44]:
```python
# Updated Evaluation metric Score Chart
score
```

[44]:

	Logistic regression	Logistic regression tuned	Decision Tree
Precision Train	0.961905	0.980952	1.000000
Precision Test	0.945098	1.000000	0.930864
Recall Train	0.961905	0.980952	1.000000
Recall Test	0.933333	1.000000	0.911111
Accuracy Train	0.961905	0.980952	1.000000
Accuracy Test	0.933333	1.000000	0.911111
F1 macro Train	0.961905	0.980952	1.000000
F1 macro Test	0.932855	1.000000	0.909829

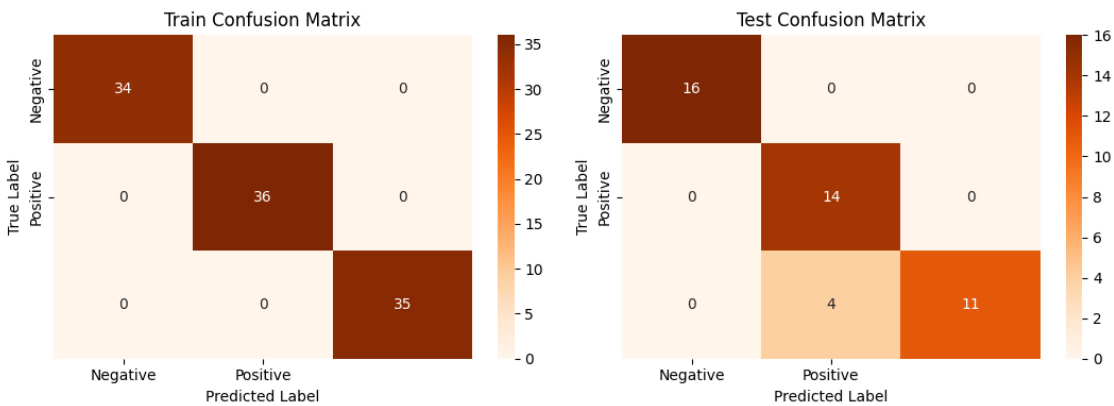
6.3 ML Model - 3 : Random Forest

[45]:
```python
# ML Model - 3 Implementation
rf_model = RandomForestClassifier(random_state=0)
```

6.3.1 1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

[46]:
```python
# Visualizing evaluation Metric Score chart
rf_score = evaluate_model(rf_model, x_train, x_test, y_train, y_test)
```

```
Confusion Matrix:
```



Train Classification Report:

	precision	recall	f1-score	support
0	1	1	1	34
1	1	1	1	36
2	1	1	1	35
accuracy	1	1	1	1
macro avg	1	1	1	105
weighted avg	1	1	1	105

Test Classification Report:

	precision	recall	f1-score	support
0	1	1	1	16
1	0.777778	1	0.875	14
2	1	0.733333	0.846154	15
accuracy	0.911111	0.911111	0.911111	0.911111
macro avg	0.925926	0.911111	0.907051	45
weighted avg	0.930864	0.911111	0.909829	45

[47]:

```python
# Updated Evaluation metric Score Chart
score['Random Forest'] = rf_score
score
```

[47]:

	Logistic regression	Logistic regression tuned \
Precision Train	0.961905	0.980952
Precision Test	0.945098	1.000000
Recall Train	0.961905	0.980952
Recall Test	0.933333	1.000000
Accuracy Train	0.961905	0.980952
Accuracy Test	0.933333	1.000000
F1 macro Train	0.961905	0.980952

```
F1 macro Test              0.932855                  1.000000

                 Decision Tree  Random Forest
Precision Train       1.000000       1.000000
Precision Test        0.930864       0.930864
Recall Train          1.000000       1.000000
Recall Test           0.911111       0.911111
Accuracy Train        1.000000       1.000000
Accuracy Test         0.911111       0.911111
F1 macro Train        1.000000       1.000000
F1 macro Test         0.909829       0.909829
```

6.3.2 2. Cross- Validation & Hyperparameter Tuning

[48]:

```
# ML Model - 3 Implementation with hyperparameter optimization techniques (i.e.
 ↪, GridSearch CV, RandomSearch CV, Bayesian Optimization etc.)
# Define the hyperparameter grid
grid = {'n_estimators': [10, 50, 100, 200],
              'max_depth': [8, 9, 10, 11, 12,13, 14, 15],
              'min_samples_split': [2, 3, 4, 5]}

# Initialize the model
rf = RandomForestClassifier(random_state=0)

# Repeated stratified kfold
rskf = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=0)

# Initialize RandomSearchCV
random_search = RandomizedSearchCV(rf, grid,cv=rskf, n_iter=10, n_jobs=-1)

# Fit the RandomSearchCV to the training data
random_search.fit(x_train, y_train)

# Select the best hyperparameters
best_params = random_search.best_params_
print("Best hyperparameters: ", best_params)
```

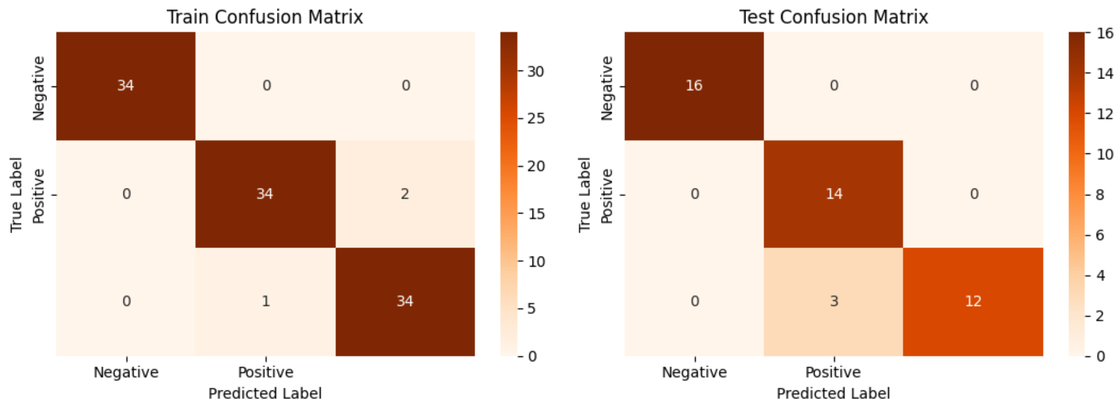
```
Best hyperparameters:  {'n_estimators': 10, 'min_samples_split': 5, 'max_depth':
14}
```

[49]:

```
# Initialize model with best parameters
rf_model2 = RandomForestClassifier(n_estimators = best_params['n_estimators'],
                                  min_samples_leaf=␣
 ↪best_params['min_samples_split'],
                                  max_depth = best_params['max_depth'],
                                  random_state=0)
```

[50]:
```
# Visualizing evaluation Metric Score chart
rf2_score = evaluate_model(rf_model2, x_train, x_test, y_train, y_test)
```

Confusion Matrix:



Train Classification Report:

	precision	recall	f1-score	support
0	1	1	1	34
1	0.971429	0.944444	0.957746	36
2	0.944444	0.971429	0.957746	35
accuracy	0.971429	0.971429	0.971429	0.971429
macro avg	0.971958	0.971958	0.971831	105
weighted avg	0.971686	0.971429	0.971429	105

Test Classification Report:

	precision	recall	f1-score	support
0	1	1	1	16
1	0.823529	1	0.903226	14
2	1	0.8	0.888889	15
accuracy	0.933333	0.933333	0.933333	0.933333
macro avg	0.941176	0.933333	0.930705	45
weighted avg	0.945098	0.933333	0.932855	45

[51]:
```
# Updated Evaluation metric Score Chart
score
```

[51]:
```
                 Logistic regression  Logistic regression tuned  \
Precision Train             0.961905                   0.980952
Precision Test              0.945098                   1.000000
```

Recall Train	0.961905	0.980952
Recall Test	0.933333	1.000000
Accuracy Train	0.961905	0.980952
Accuracy Test	0.933333	1.000000
F1 macro Train	0.961905	0.980952
F1 macro Test	0.932855	1.000000

	Decision Tree	Random Forest
Precision Train	1.000000	1.000000
Precision Test	0.930864	0.930864
Recall Train	1.000000	1.000000
Recall Test	0.911111	0.911111
Accuracy Train	1.000000	1.000000
Accuracy Test	0.911111	0.911111
F1 macro Train	1.000000	1.000000
F1 macro Test	0.909829	0.909829

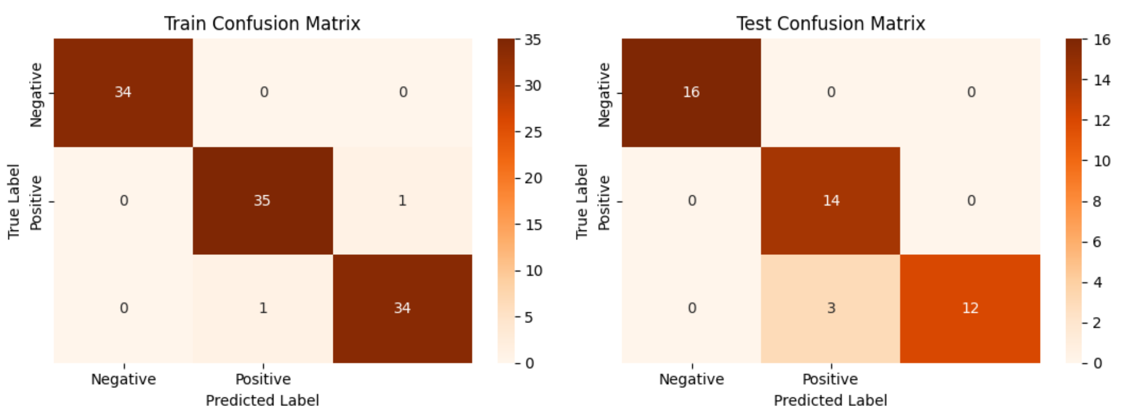
6.4 ML Model - 4 : SVM (Support Vector Machine)

[52]:
```
# ML Model - 4 Implementation
svm_model = SVC(kernel='linear', random_state=0, probability=True)
```

6.4.1 1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

[53]:
```
# Visualizing evaluation Metric Score chart
svm_score = evaluate_model(svm_model, x_train, x_test, y_train, y_test)
```

Confusion Matrix:



Train Classification Report:

```
|              |   precision |   recall |   f1-score |    support |
|:-------------|------------:|---------:|-----------:|-----------:|
| 0            |    1        | 1        |   1        |  34        |
| 1            |    0.972222 | 0.972222 |   0.972222 |  36        |
| 2            |    0.971429 | 0.971429 |   0.971429 |  35        |
| accuracy     |    0.980952 | 0.980952 |   0.980952 |   0.980952 |
| macro avg    |    0.981217 | 0.981217 |   0.981217 | 105        |
| weighted avg |    0.980952 | 0.980952 |   0.980952 | 105        |

Test Classification Report:
|              |   precision |   recall |   f1-score |   support |
|:-------------|------------:|---------:|-----------:|----------:|
| 0            |    1        | 1        |   1        | 16        |
| 1            |    0.823529 | 1        |   0.903226 | 14        |
| 2            |    1        | 0.8      |   0.888889 | 15        |
| accuracy     |    0.933333 | 0.933333 |   0.933333 |  0.933333 |
| macro avg    |    0.941176 | 0.933333 |   0.930705 | 45        |
| weighted avg |    0.945098 | 0.933333 |   0.932855 | 45        |
```

[54]:

```python
# Updated Evaluation metric Score Chart
score['SVM'] = svm_score
score
```

[54]:

```
                 Logistic regression  Logistic regression tuned  \
Precision Train             0.961905                   0.980952
Precision Test              0.945098                   1.000000
Recall Train                0.961905                   0.980952
Recall Test                 0.933333                   1.000000
Accuracy Train              0.961905                   0.980952
Accuracy Test               0.933333                   1.000000
F1 macro Train              0.961905                   0.980952
F1 macro Test               0.932855                   1.000000

                 Decision Tree  Random Forest       SVM
Precision Train       1.000000       1.000000  0.980952
Precision Test        0.930864       0.930864  0.945098
Recall Train          1.000000       1.000000  0.980952
Recall Test           0.911111       0.911111  0.933333
Accuracy Train        1.000000       1.000000  0.980952
Accuracy Test         0.911111       0.911111  0.933333
F1 macro Train        1.000000       1.000000  0.980952
F1 macro Test         0.909829       0.909829  0.932855
```

6.4.2 2. Cross- Validation & Hyperparameter Tuning

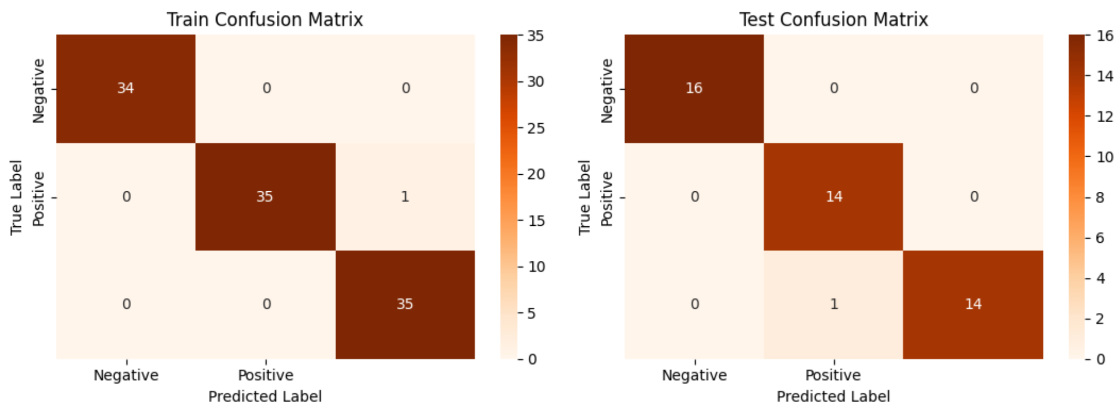
[55]:
```
# ML Model - 4 Implementation with hyperparameter optimization techniques (i.e.
↪, GridSearch CV, RandomSearch CV, Bayesian Optimization etc.)
# Define the hyperparameter grid
param_grid = {'C': np.arange(0.1, 10, 0.1),
              'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
              'degree': np.arange(2, 6, 1)}

# Initialize the model
svm = SVC(random_state=0, probability=True)

# Repeated stratified kfold
rskf = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=0)

# Initialize RandomizedSearchCV with kfold cross-validation
random_search = RandomizedSearchCV(svm, param_grid, n_iter=10, cv=rskf,␣
↪n_jobs=-1)

# Fit the RandomizedSearchCV to the training data
random_search.fit(x_train, y_train)

# Select the best hyperparameters
best_params = random_search.best_params_
print("Best hyperparameters: ", best_params)
```

```
Best hyperparameters:  {'kernel': 'rbf', 'degree': 4, 'C': 4.3999999999999995}
```

[56]:
```
# Initialize model with best parameters
svm_model2 = SVC(C = best_params['C'],
           kernel = best_params['kernel'],
           degree = best_params['degree'],
           random_state=0, probability=True)
```

[57]:
```
# Visualizing evaluation Metric Score chart
svm2_score = evaluate_model(svm_model2, x_train, x_test, y_train, y_test)
```

```
Confusion Matrix:
```




```
Train Classification Report:
|              |   precision |   recall |   f1-score |    support |
|:-------------|------------:|---------:|-----------:|-----------:|
| 0            |    1        | 1        |   1        |  34        |
| 1            |    1        | 0.972222 |   0.985915 |  36        |
| 2            |    0.972222 | 1        |   0.985915 |  35        |
| accuracy     |    0.990476 | 0.990476 |   0.990476 |   0.990476 |
| macro avg    |    0.990741 | 0.990741 |   0.99061  | 105        |
| weighted avg |    0.990741 | 0.990476 |   0.990476 | 105        |

Test Classification Report:
|              |   precision |   recall |   f1-score |   support |
|:-------------|------------:|---------:|-----------:|----------:|
| 0            |    1        | 1        |   1        | 16        |
| 1            |    0.933333 | 1        |   0.965517 | 14        |
| 2            |    1        | 0.933333 |   0.965517 | 15        |
| accuracy     |    0.977778 | 0.977778 |   0.977778 |  0.977778 |
| macro avg    |    0.977778 | 0.977778 |   0.977011 | 45        |
| weighted avg |    0.979259 | 0.977778 |   0.977778 | 45        |
```

[58]:

```python
score['SVM tuned'] = svm2_score
```

[59]:

```python
# Updated Evaluation metric Score Chart
score
```

[59]:

```
                 Logistic regression  Logistic regression tuned  \
Precision Train             0.961905                   0.980952
Precision Test              0.945098                   1.000000
Recall Train                0.961905                   0.980952
Recall Test                 0.933333                   1.000000
Accuracy Train              0.961905                   0.980952
Accuracy Test               0.933333                   1.000000
```

F1 macro Train	0.961905	0.980952
F1 macro Test	0.932855	1.000000

	Decision Tree	Random Forest	SVM	SVM tuned
Precision Train	1.000000	1.000000	0.980952	0.990741
Precision Test	0.930864	0.930864	0.945098	0.979259
Recall Train	1.000000	1.000000	0.980952	0.990476
Recall Test	0.911111	0.911111	0.933333	0.977778
Accuracy Train	1.000000	1.000000	0.980952	0.990476
Accuracy Test	0.911111	0.911111	0.933333	0.977778
F1 macro Train	1.000000	1.000000	0.980952	0.990476
F1 macro Test	0.909829	0.909829	0.932855	0.977778

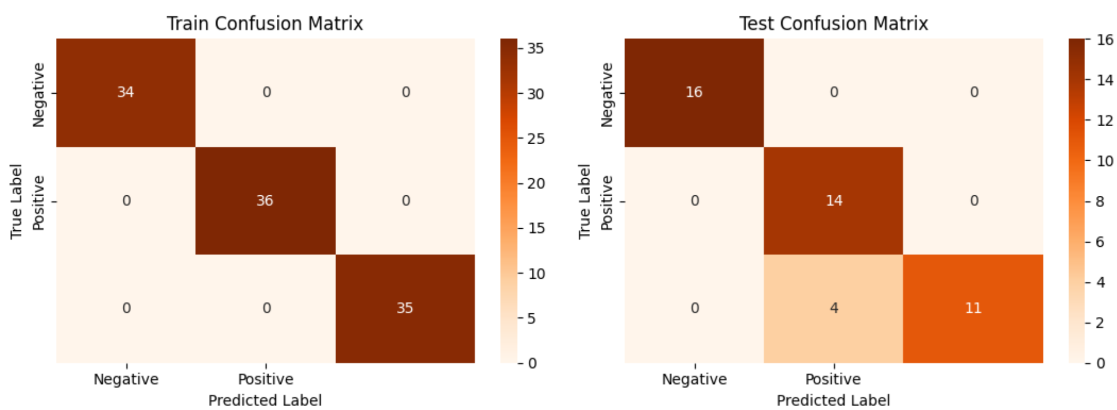
6.5 ML Model - 5 : Xtreme Gradient Boosting

[60]:
```
# ML Model - 5 Implementation
xgb_model = xgb.XGBClassifier()
```

6.5.1 1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

[61]:
```
# Visualizing evaluation Metric Score chart
xgb_score = evaluate_model(xgb_model, x_train, x_test, y_train, y_test)
```

```
Confusion Matrix:
```




```
Train Classification Report:
|              |   precision |   recall |   f1-score |   support |
|:-------------|------------:|---------:|-----------:|----------:|
| 0            |           1 |        1 |          1 |        34 |
| 1            |           1 |        1 |          1 |        36 |
```

```
| 2            |           1 |        1 |          1 |        35 |
| accuracy     |           1 |        1 |          1 |         1 |
| macro avg    |           1 |        1 |          1 |       105 |
| weighted avg |           1 |        1 |          1 |       105 |

Test Classification Report:
|              |   precision |   recall |   f1-score |   support |
|:-------------|------------:|---------:|-----------:|----------:|
| 0            |    1        | 1        |   1        | 16        |
| 1            |    0.777778 | 1        |   0.875    | 14        |
| 2            |    1        | 0.733333 |   0.846154 | 15        |
| accuracy     |    0.911111 | 0.911111 |   0.911111 |  0.911111 |
| macro avg    |    0.925926 | 0.911111 |   0.907051 | 45        |
| weighted avg |    0.930864 | 0.911111 |   0.909829 | 45        |
```

[62]:
```python
# Updated Evaluation metric Score Chart
score['XGB'] = xgb_score
score
```

[62]:
```
                 Logistic regression  Logistic regression tuned  \
Precision Train             0.961905                   0.980952
Precision Test              0.945098                   1.000000
Recall Train                0.961905                   0.980952
Recall Test                 0.933333                   1.000000
Accuracy Train              0.961905                   0.980952
Accuracy Test               0.933333                   1.000000
F1 macro Train              0.961905                   0.980952
F1 macro Test               0.932855                   1.000000

                 Decision Tree  Random Forest       SVM  SVM tuned       XGB
Precision Train       1.000000       1.000000  0.980952   0.990741  1.000000
Precision Test        0.930864       0.930864  0.945098   0.979259  0.930864
Recall Train          1.000000       1.000000  0.980952   0.990476  1.000000
Recall Test           0.911111       0.911111  0.933333   0.977778  0.911111
Accuracy Train        1.000000       1.000000  0.980952   0.990476  1.000000
Accuracy Test         0.911111       0.911111  0.933333   0.977778  0.911111
F1 macro Train        1.000000       1.000000  0.980952   0.990476  1.000000
F1 macro Test         0.909829       0.909829  0.932855   0.977778  0.909829
```

6.5.2 2. Cross- Validation & Hyperparameter Tuning

[63]:
```python
# ML Model - 5 Implementation with hyperparameter optimization techniques (i.e.
↪, GridSearch CV, RandomSearch CV, Bayesian Optimization etc.)
# Define the hyperparameter grid
param_grid = {'learning_rate': np.arange(0.01, 0.3, 0.01),
              'max_depth': np.arange(3, 15, 1),
              'n_estimators': np.arange(100, 200, 10)}
```

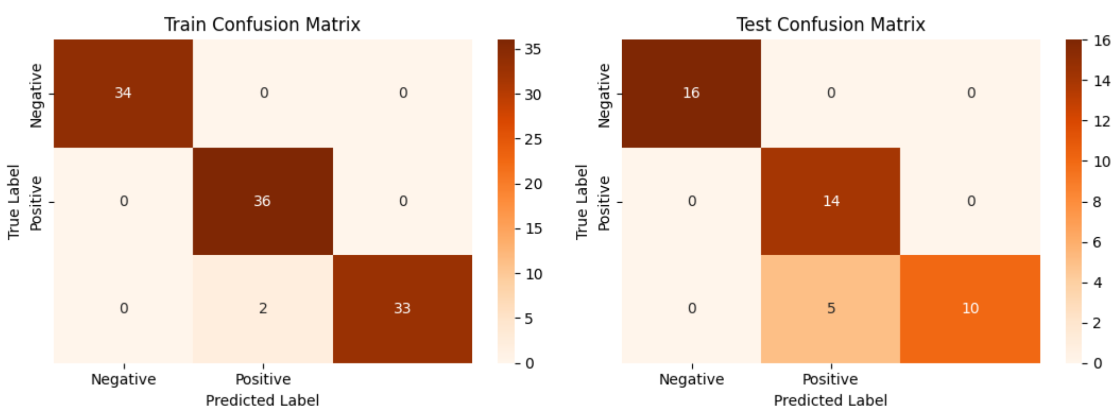
```python
# Initialize the model
xgb2 = xgb.XGBClassifier(random_state=0)

# Repeated stratified kfold
rskf = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=0)

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(xgb2, param_grid, n_iter=10, cv=rskf)

# Fit the RandomizedSearchCV to the training data
random_search.fit(x_train, y_train)

# Select the best hyperparameters
best_params = random_search.best_params_
print("Best hyperparameters: ", best_params)
```

```
Best hyperparameters:  {'n_estimators': 150, 'max_depth': 14, 'learning_rate':
0.01}
```

[64]:
```python
# Initialize model with best parameters
xgb_model2 = xgb.XGBClassifier(learning_rate = best_params['learning_rate'],
                                 max_depth = best_params['max_depth'],
                               n_estimators = best_params['n_estimators'],
                                 random_state=0)
```

[65]:
```python
# Visualizing evaluation Metric Score chart
xgb2_score = evaluate_model(xgb_model2, x_train, x_test, y_train, y_test)
```

```
Confusion Matrix:
```




```
Train Classification Report:
```

```
|              |   precision |   recall |   f1-score |    support |
|:-------------|------------:|---------:|-----------:|-----------:|
| 0            |    1        | 1        |   1        |  34        |
| 1            |    0.947368 | 1        |   0.972973 |  36        |
| 2            |    1        | 0.942857 |   0.970588 |  35        |
| accuracy     |    0.980952 | 0.980952 |   0.980952 |   0.980952 |
| macro avg    |    0.982456 | 0.980952 |   0.981187 | 105        |
| weighted avg |    0.981955 | 0.980952 |   0.98093  | 105        |

Test Classification Report:
|              |   precision |   recall |   f1-score |   support |
|:-------------|------------:|---------:|-----------:|----------:|
| 0            |    1        | 1        |   1        | 16        |
| 1            |    0.736842 | 1        |   0.848485 | 14        |
| 2            |    1        | 0.666667 |   0.8      | 15        |
| accuracy     |    0.888889 | 0.888889 |   0.888889 |  0.888889 |
| macro avg    |    0.912281 | 0.888889 |   0.882828 | 45        |
| weighted avg |    0.918129 | 0.888889 |   0.886195 | 45        |
```

[66]:
```python
score['XGB tuned'] = xgb2_score
```

[67]:
```python
# Updated Evaluation metric Score Chart
score
```

[67]:
```
                 Logistic regression  Logistic regression tuned  \
Precision Train             0.961905                   0.980952
Precision Test              0.945098                   1.000000
Recall Train                0.961905                   0.980952
Recall Test                 0.933333                   1.000000
Accuracy Train              0.961905                   0.980952
Accuracy Test               0.933333                   1.000000
F1 macro Train              0.961905                   0.980952
F1 macro Test               0.932855                   1.000000

                 Decision Tree  Random Forest       SVM  SVM tuned       XGB  \
Precision Train       1.000000       1.000000  0.980952   0.990741  1.000000
Precision Test        0.930864       0.930864  0.945098   0.979259  0.930864
Recall Train          1.000000       1.000000  0.980952   0.990476  1.000000
Recall Test           0.911111       0.911111  0.933333   0.977778  0.911111
Accuracy Train        1.000000       1.000000  0.980952   0.990476  1.000000
Accuracy Test         0.911111       0.911111  0.933333   0.977778  0.911111
F1 macro Train        1.000000       1.000000  0.980952   0.990476  1.000000
F1 macro Test         0.909829       0.909829  0.932855   0.977778  0.909829

                 XGB tuned
Precision Train   0.981955
Precision Test    0.918129
```

```
Recall Train       0.980952
Recall Test        0.888889
Accuracy Train     0.980952
Accuracy Test      0.888889
F1 macro Train     0.980930
F1 macro Test      0.886195
```

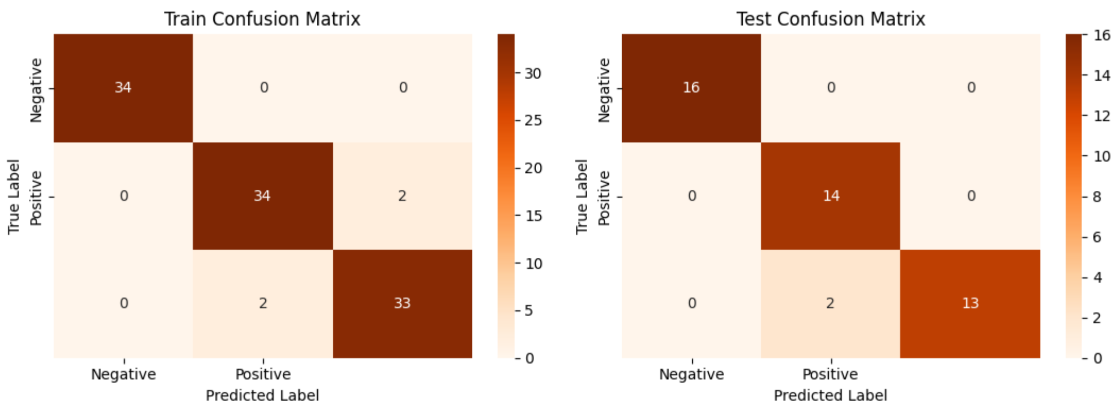
6.6 ML Model - 6 : Naive Bayes

[68]:
```python
# ML Model - 6 Implementation
nb_model = GaussianNB()
```

6.6.1 1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

[69]:
```python
# Visualizing evaluation Metric Score chart
nb_score = evaluate_model(nb_model, x_train, x_test, y_train, y_test)
```

Confusion Matrix:



Train Classification Report:

	precision	recall	f1-score	support
0	1	1	1	34
1	0.944444	0.944444	0.944444	36
2	0.942857	0.942857	0.942857	35
accuracy	0.961905	0.961905	0.961905	0.961905
macro avg	0.962434	0.962434	0.962434	105
weighted avg	0.961905	0.961905	0.961905	105

Test Classification Report:

	precision	recall	f1-score	support
0	1	1	1	16
1	0.875	1	0.933333	14
2	1	0.866667	0.928571	15
accuracy	0.955556	0.955556	0.955556	0.955556
macro avg	0.958333	0.955556	0.953968	45
weighted avg	0.961111	0.955556	0.95545	45

[70]:

```
# Updated Evaluation metric Score Chart
score['Naive Bayes'] = nb_score
score
```

[70]:

```
                 Logistic regression  Logistic regression tuned  \
Precision Train             0.961905                   0.980952
Precision Test              0.945098                   1.000000
Recall Train                0.961905                   0.980952
Recall Test                 0.933333                   1.000000
Accuracy Train              0.961905                   0.980952
Accuracy Test               0.933333                   1.000000
F1 macro Train              0.961905                   0.980952
F1 macro Test               0.932855                   1.000000

                 Decision Tree  Random Forest       SVM  SVM tuned       XGB  \
Precision Train       1.000000       1.000000  0.980952   0.990741  1.000000
Precision Test        0.930864       0.930864  0.945098   0.979259  0.930864
Recall Train          1.000000       1.000000  0.980952   0.990476  1.000000
Recall Test           0.911111       0.911111  0.933333   0.977778  0.911111
Accuracy Train        1.000000       1.000000  0.980952   0.990476  1.000000
Accuracy Test         0.911111       0.911111  0.933333   0.977778  0.911111
F1 macro Train        1.000000       1.000000  0.980952   0.990476  1.000000
F1 macro Test         0.909829       0.909829  0.932855   0.977778  0.909829

                 XGB tuned  Naive Bayes
Precision Train   0.981955     0.961905
Precision Test    0.918129     0.961111
Recall Train      0.980952     0.961905
Recall Test       0.888889     0.955556
Accuracy Train    0.980952     0.961905
Accuracy Test     0.888889     0.955556
F1 macro Train    0.980930     0.961905
F1 macro Test     0.886195     0.955450
```

6.6.2 2. Cross- Validation & Hyperparameter Tuning

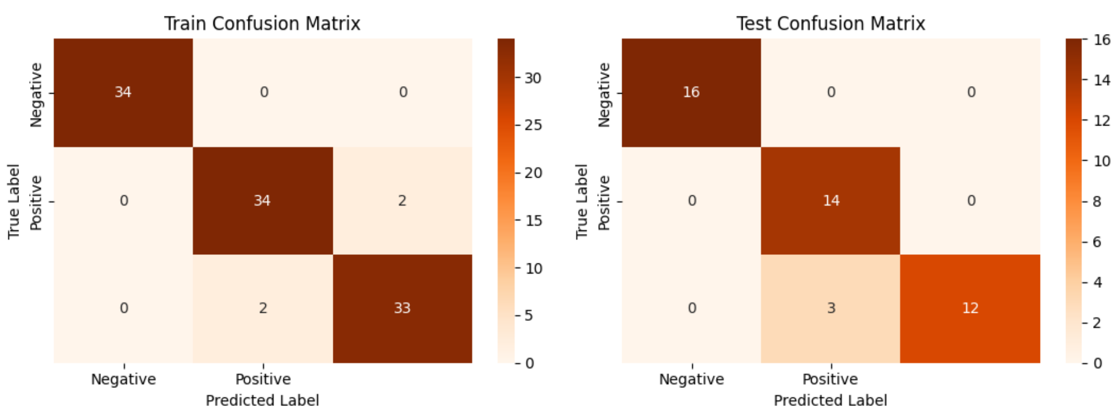
[71]:
```
# ML Model - 6 Implementation with hyperparameter optimization techniques (i.e.
↪, GridSearch CV, RandomSearch CV, Bayesian Optimization etc.)
# Define the hyperparameter grid
param_grid = {'var_smoothing': np.logspace(0,-9, num=100)}

# Initialize the model
naive = GaussianNB()

# repeated stratified kfold
rskf = RepeatedStratifiedKFold(n_splits=4, n_repeats=4, random_state=0)

# Initialize GridSearchCV
GridSearch = GridSearchCV(naive, param_grid, cv=rskf, n_jobs=-1)

# Fit the GridSearchCV to the training data
GridSearch.fit(x_train, y_train)

# Select the best hyperparameters
best_params = GridSearch.best_params_
print("Best hyperparameters: ", best_params)
```

```
Best hyperparameters:  {'var_smoothing': 0.02848035868435802}
```

[72]:
```
# Initiate model with best parameters
nb_model2 = GaussianNB(var_smoothing = best_params['var_smoothing'])
```

[73]:
```
# Visualizing evaluation Metric Score chart
nb2_score = evaluate_model(nb_model2, x_train, x_test, y_train, y_test)
```

```
Confusion Matrix:
```



```
Train Classification Report:
|              |   precision |   recall |   f1-score |    support |
|:-------------|------------:|---------:|-----------:|-----------:|
| 0            |    1        | 1        |   1        |  34        |
| 1            |    0.944444 | 0.944444 |   0.944444 |  36        |
| 2            |    0.942857 | 0.942857 |   0.942857 |  35        |
| accuracy     |    0.961905 | 0.961905 |   0.961905 |   0.961905 |
| macro avg    |    0.962434 | 0.962434 |   0.962434 | 105        |
| weighted avg |    0.961905 | 0.961905 |   0.961905 | 105        |

Test Classification Report:
|              |   precision |   recall |   f1-score |   support |
|:-------------|------------:|---------:|-----------:|----------:|
| 0            |    1        | 1        |   1        | 16        |
| 1            |    0.823529 | 1        |   0.903226 | 14        |
| 2            |    1        | 0.8      |   0.888889 | 15        |
| accuracy     |    0.933333 | 0.933333 |   0.933333 |  0.933333 |
| macro avg    |    0.941176 | 0.933333 |   0.930705 | 45        |
| weighted avg |    0.945098 | 0.933333 |   0.932855 | 45        |
```

[74]:
```python
score['Naive Bayes tuned']= nb2_score
```

[75]:
```python
# Updated Evaluation metric Score Chart
score
```

[75]:
```
                 Logistic regression  Logistic regression tuned  \
Precision Train             0.961905                   0.980952
Precision Test              0.945098                   1.000000
Recall Train                0.961905                   0.980952
Recall Test                 0.933333                   1.000000
Accuracy Train              0.961905                   0.980952
Accuracy Test               0.933333                   1.000000
F1 macro Train              0.961905                   0.980952
F1 macro Test               0.932855                   1.000000

                 Decision Tree  Random Forest       SVM  SVM tuned       XGB  \
Precision Train       1.000000       1.000000  0.980952   0.990741  1.000000
Precision Test        0.930864       0.930864  0.945098   0.979259  0.930864
Recall Train          1.000000       1.000000  0.980952   0.990476  1.000000
Recall Test           0.911111       0.911111  0.933333   0.977778  0.911111
Accuracy Train        1.000000       1.000000  0.980952   0.990476  1.000000
Accuracy Test         0.911111       0.911111  0.933333   0.977778  0.911111
F1 macro Train        1.000000       1.000000  0.980952   0.990476  1.000000
F1 macro Test         0.909829       0.909829  0.932855   0.977778  0.909829

                 XGB tuned  Naive Bayes  Naive Bayes tuned
```

```
Precision Train   0.981955     0.961905           0.961905
Precision Test    0.918129     0.961111           0.945098
Recall Train      0.980952     0.961905           0.961905
Recall Test       0.888889     0.955556           0.933333
Accuracy Train    0.980952     0.961905           0.961905
Accuracy Test     0.888889     0.955556           0.933333
F1 macro Train    0.980930     0.961905           0.961905
F1 macro Test     0.886195     0.955450           0.932855
```

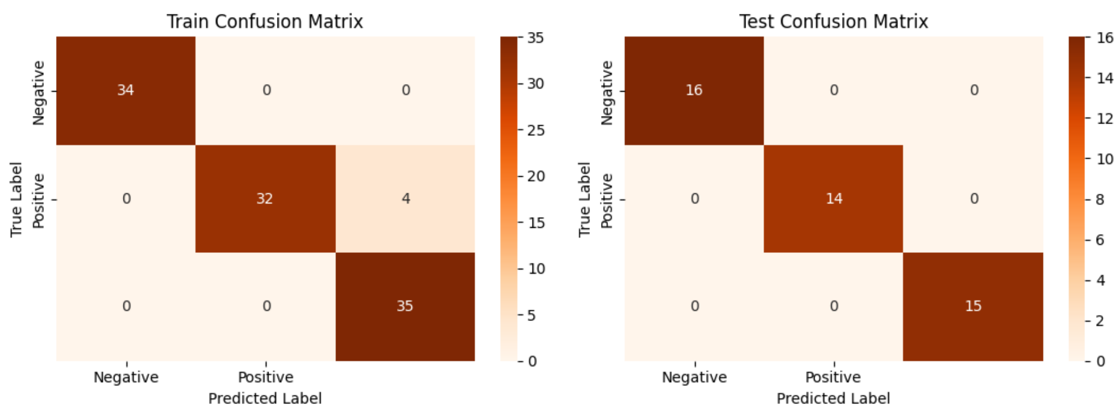
6.7 ML Model - 7 : Neural Network

[76]:
```
# ML Model - 7 Implementation
nn_model = MLPClassifier(random_state=0)
```

6.7.1 1. Explain the ML Model used and it's performance using Evaluation metric Score Chart.

[77]:
```
# Visualizing evaluation Metric Score chart
neural_score = evaluate_model(nn_model, x_train, x_test, y_train, y_test)
```

```
Confusion Matrix:
```




```
Train Classification Report:
|              |   precision |   recall |   f1-score |    support |
|:-------------|------------:|---------:|-----------:|-----------:|
| 0            |    1        | 1        |   1        |  34        |
| 1            |    1        | 0.888889 |   0.941176 |  36        |
| 2            |    0.897436 | 1        |   0.945946 |  35        |
| accuracy     |    0.961905 | 0.961905 |   0.961905 |   0.961905 |
| macro avg    |    0.965812 | 0.962963 |   0.962374 | 105        |
| weighted avg |    0.965812 | 0.961905 |   0.961814 | 105        |
```

Test Classification Report:

	precision	recall	f1-score	support
0	1	1	1	16
1	1	1	1	14
2	1	1	1	15
accuracy	1	1	1	1
macro avg	1	1	1	45
weighted avg	1	1	1	45

[78]:

```python
# Updated Evaluation metric Score Chart
score['Neural Network'] = neural_score
score
```

[78]:

	Logistic regression	Logistic regression tuned
Precision Train	0.961905	0.980952
Precision Test	0.945098	1.000000
Recall Train	0.961905	0.980952
Recall Test	0.933333	1.000000
Accuracy Train	0.961905	0.980952
Accuracy Test	0.933333	1.000000
F1 macro Train	0.961905	0.980952
F1 macro Test	0.932855	1.000000

	Decision Tree	Random Forest	SVM	SVM tuned	XGB
Precision Train	1.000000	1.000000	0.980952	0.990741	1.000000
Precision Test	0.930864	0.930864	0.945098	0.979259	0.930864
Recall Train	1.000000	1.000000	0.980952	0.990476	1.000000
Recall Test	0.911111	0.911111	0.933333	0.977778	0.911111
Accuracy Train	1.000000	1.000000	0.980952	0.990476	1.000000
Accuracy Test	0.911111	0.911111	0.933333	0.977778	0.911111
F1 macro Train	1.000000	1.000000	0.980952	0.990476	1.000000
F1 macro Test	0.909829	0.909829	0.932855	0.977778	0.909829

	XGB tuned	Naive Bayes	Naive Bayes tuned	Neural Network
Precision Train	0.981955	0.961905	0.961905	0.965812
Precision Test	0.918129	0.961111	0.945098	1.000000
Recall Train	0.980952	0.961905	0.961905	0.961905
Recall Test	0.888889	0.955556	0.933333	1.000000
Accuracy Train	0.980952	0.961905	0.961905	0.961905
Accuracy Test	0.888889	0.955556	0.933333	1.000000
F1 macro Train	0.980930	0.961905	0.961905	0.961814
F1 macro Test	0.886195	0.955450	0.932855	1.000000

6.7.2 2. Cross- Validation & Hyperparameter Tuning

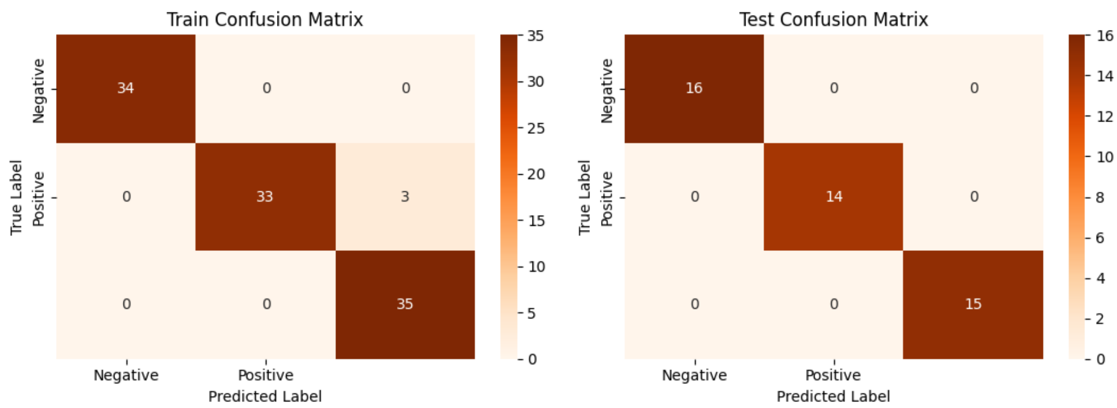
[79]:
```
# ML Model - 7 Implementation with hyperparameter optimization techniques (i.e.
↪, GridSearch CV, RandomSearch CV, Bayesian Optimization etc.)
# Define the hyperparameter grid
param_grid = {'hidden_layer_sizes': np.arange(10, 100, 10),
              'alpha': np.arange(0.0001, 0.01, 0.0001)}

# Initialize the model
neural = MLPClassifier(random_state=0)

# Repeated stratified kfold
rskf = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=0)

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(neural, param_grid, n_iter=10, cv=rskf,␣
↪n_jobs=-1)

# Fit the RandomizedSearchCV to the training data
random_search.fit(x_train, y_train)

# Select the best hyperparameters
best_params = random_search.best_params_
print("Best hyperparameters: ", best_params)
```

```
Best hyperparameters:  {'hidden_layer_sizes': 90, 'alpha': 0.0077}
```

[80]:
```
# Initiate model with best parameters
nn_model2 = MLPClassifier(hidden_layer_sizes =␣
↪best_params['hidden_layer_sizes'],
                        alpha = best_params['alpha'],
                        random_state = 0)
```

[81]:
```
# Visualizing evaluation Metric Score chart
neural2_score = evaluate_model(nn_model2, x_train, x_test, y_train, y_test)
```

```
Confusion Matrix:
```



Train Classification Report:

	precision	recall	f1-score	support
0	1	1	1	34
1	1	0.916667	0.956522	36
2	0.921053	1	0.958904	35
accuracy	0.971429	0.971429	0.971429	0.971429
macro avg	0.973684	0.972222	0.971809	105
weighted avg	0.973684	0.971429	0.971395	105

Test Classification Report:

	precision	recall	f1-score	support
0	1	1	1	16
1	1	1	1	14
2	1	1	1	15
accuracy	1	1	1	1
macro avg	1	1	1	45
weighted avg	1	1	1	45

[82]:
```python
# Updated Evaluation metric Score Chart
score
```

[82]:

	Logistic regression	Logistic regression tuned \
Precision Train	0.961905	0.980952
Precision Test	0.945098	1.000000
Recall Train	0.961905	0.980952
Recall Test	0.933333	1.000000
Accuracy Train	0.961905	0.980952
Accuracy Test	0.933333	1.000000
F1 macro Train	0.961905	0.980952
F1 macro Test	0.932855	1.000000

```
                 Decision Tree  Random Forest       SVM  SVM tuned       XGB  \
Precision Train       1.000000       1.000000  0.980952   0.990741  1.000000
Precision Test        0.930864       0.930864  0.945098   0.979259  0.930864
Recall Train          1.000000       1.000000  0.980952   0.990476  1.000000
Recall Test           0.911111       0.911111  0.933333   0.977778  0.911111
Accuracy Train        1.000000       1.000000  0.980952   0.990476  1.000000
Accuracy Test         0.911111       0.911111  0.933333   0.977778  0.911111
F1 macro Train        1.000000       1.000000  0.980952   0.990476  1.000000
F1 macro Test         0.909829       0.909829  0.932855   0.977778  0.909829

                 XGB tuned  Naive Bayes  Naive Bayes tuned  Neural Network
Precision Train   0.981955     0.961905           0.961905        0.965812
Precision Test    0.918129     0.961111           0.945098        1.000000
Recall Train      0.980952     0.961905           0.961905        0.961905
Recall Test       0.888889     0.955556           0.933333        1.000000
Accuracy Train    0.980952     0.961905           0.961905        0.961905
Accuracy Test     0.888889     0.955556           0.933333        1.000000
F1 macro Train    0.980930     0.961905           0.961905        0.961814
F1 macro Test     0.886195     0.955450           0.932855        1.000000
```

[83]:
```
print(score.to_markdown())
```

```
|                 |   Logistic regression |   Logistic regression tuned |
Decision Tree |   Random Forest |       SVM |   SVM tuned |       XGB |   XGB
tuned |    Naive Bayes |    Naive Bayes tuned |    Neural Network |
|:----------------|----------------------:|----------------------------:|-------
---------:|----------------:|---------:|------------:|---------:|------------:|-
-------------:|--------------------:|-----------------:|
| Precision Train |              0.961905 |                    0.980952 |
1         |       1         | 0.980952 |    0.990741 | 1        |    0.981955 |
0.961905 |          0.961905 |         0.965812 |
| Precision Test  |              0.945098 |                    1        |
0.930864 |        0.930864 | 0.945098 |    0.979259 | 0.930864 |    0.918129 |
0.961111 |          0.945098 |         1        |
| Recall Train    |              0.961905 |                    0.980952 |
1         |       1         | 0.980952 |    0.990476 | 1        |    0.980952 |
0.961905 |          0.961905 |         0.961905 |
| Recall Test     |              0.933333 |                    1        |
0.911111 |        0.911111 | 0.933333 |    0.977778 | 0.911111 |    0.888889 |
0.955556 |          0.933333 |         1        |
| Accuracy Train  |              0.961905 |                    0.980952 |
1         |       1         | 0.980952 |    0.990476 | 1        |    0.980952 |
0.961905 |          0.961905 |         0.961905 |
| Accuracy Test   |              0.933333 |                    1        |
0.911111 |        0.911111 | 0.933333 |    0.977778 | 0.911111 |    0.888889 |
0.955556 |          0.933333 |         1        |
| F1 macro Train  |              0.961905 |                    0.980952 |
```

```
1          |          1          | 0.980952 |     0.990476 | 1          |     0.98093   |
0.961905 |             0.961905 |            0.961814 |
| F1 macro Test    |                 0.932855 |                      1          |
0.909829 |          0.909829 | 0.932855 |     0.977778 | 0.909829 |     0.886195 |
0.95545   |             0.932855 |            1          |
```

6.8 Selection of best model

[84]:
```python
# Removing the overfitted models which have precision, recall, f1 scores for train as 1
score_t = score.transpose()            # taking transpose of the score dataframe to create new difference column
remove_models = score_t[score_t['Recall Train']>=0.98].index  # creating a list of models which have 1 for train and score_t['Accuracy Train']==1.0 and score_t['Precision Train']==1.0 and score_t['F1 macro Train']==1.0
remove_models
```

[84]:
```
Index(['Logistic regression tuned', 'Decision Tree', 'Random Forest', 'SVM',
       'SVM tuned', 'XGB', 'XGB tuned'],
      dtype='object')
```

[85]:
```python
adj = score_t.drop(remove_models)                          # creating a new dataframe with required models
adj
```

[85]:
```
                     Precision Train  Precision Test  Recall Train  \
Logistic regression         0.961905        0.945098      0.961905
Naive Bayes                 0.961905        0.961111      0.961905
Naive Bayes tuned           0.961905        0.945098      0.961905
Neural Network              0.965812        1.000000      0.961905

                     Recall Test  Accuracy Train  Accuracy Test  \
Logistic regression     0.933333        0.961905       0.933333
Naive Bayes             0.955556        0.961905       0.955556
Naive Bayes tuned       0.933333        0.961905       0.933333
Neural Network          1.000000        0.961905       1.000000

                     F1 macro Train  F1 macro Test
Logistic regression        0.961905       0.932855
Naive Bayes                0.961905       0.955450
Naive Bayes tuned          0.961905       0.932855
Neural Network             0.961814       1.000000
```

[86]:
```python
def select_best_model(df, metrics):

    best_models = {}
    for metric in metrics:
```

```
        max_test = df[metric + ' Test'].max()
        best_model_test = df[df[metric + ' Test'] == max_test].index[0]
        best_model = best_model_test
        best_models[metric] = best_model
    return best_models
```

[87]:
```
metrics = ['Precision', 'Recall', 'Accuracy', 'F1 macro']

best_models = select_best_model(adj, metrics)
print("The best models are:")
for metric, best_model in best_models.items():
    print(f"{metric}: {best_model} - {adj[metric+' Test'][best_model].
 ↪round(4)}")
```

```
The best models are:
Precision: Neural Network - 1.0
Recall: Neural Network - 1.0
Accuracy: Neural Network - 1.0
F1 macro: Neural Network - 1.0
```

[88]:
```
# Take recall as the primary evaluation metric
score_smpl = score.transpose()
remove_overfitting_models = score_smpl[score_smpl['Recall Train']>=0.98].index
remove_overfitting_models
new_score = score_smpl.drop(remove_overfitting_models)
new_score = new_score.drop(['Precision Train','Precision Test','Accuracy␣
 ↪Train','Accuracy Test','F1 macro Train','F1 macro Test'], axis=1)
new_score.index.name = 'Classification Model'
print(new_score.to_markdown())
```

Classification Model	Recall Train	Recall Test
Logistic regression	0.961905	0.933333
Naive Bayes	0.961905	0.955556
Naive Bayes tuned	0.961905	0.933333
Neural Network	0.961905	1

3. Explain the model which i have used for the prediction

[89]:
```
# Define a list of category labels for reference.
Category_RF = ['Iris-Setosa', 'Iris-Versicolor', 'Iris-Virginica']


# In this example, it's a data point with Sepal Length, Sepal Width, Petal␣
 ↪Length, and Petal Width.
x_rf = np.array([[5.1, 3.5, 1.4, 0.2]])

# Use the tuned random forest model (rf_model2) to make a prediction.
```

```
x_rf_prediction = rf_model2.predict(x_rf)
x_rf_prediction[0]

# Display the predicted category label.
print(Category_RF[int(x_rf_prediction[0])])
```

```
Iris-Setosa
```