Evaluating Performance of Different Machine Learning Models

Subject: ECEData

Date: January 19, 2024

Members:

Arellano, Aldrich

Coronel, Ericka May

Fadol, Noelle Joshua

Mendoza, Joshua Jericho

TO CHECK THE VERSION OF LIBRARIES

In [ ]:
```
from platform import python_version
print (python_version())
```

```
3.11.4
```

TO IMPORT LIBRARIES

In [ ]:
```
# Import all libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings('ignore')
```

TO LOAD THE DATASET

In [ ]:
```
df = pd.read_csv("/Users/Shuahua/Downloads/Iris_Data.csv")
```

TO DETERMINE THE DIMENSIONS OF THE DATASET

In [ ]:
```
df_dim = df.shape

print("Dimensions of the Dataset: ", df_dim)
```

```
Dimensions of the Dataset:  (150, 5)
```

TO PEEK AT THE DATA

In [ ]:
```
#Displaying first 5 rows of the dataset
df.head()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	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

In [ ]:

```
#Displaying last 5 rows of the dataset
df.tail()
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width	species
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

TO SEE THE STATISTICAL SUMMARY

In [ ]:

```
#Display the statistcal summary of the data set
print("Statistical Sumamry of the Dataset")

df.describe()
```

```
Statistical Sumamry of the Dataset
```

Out[ ]:

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

TO SEE THE CLASS DISTRIBUTION

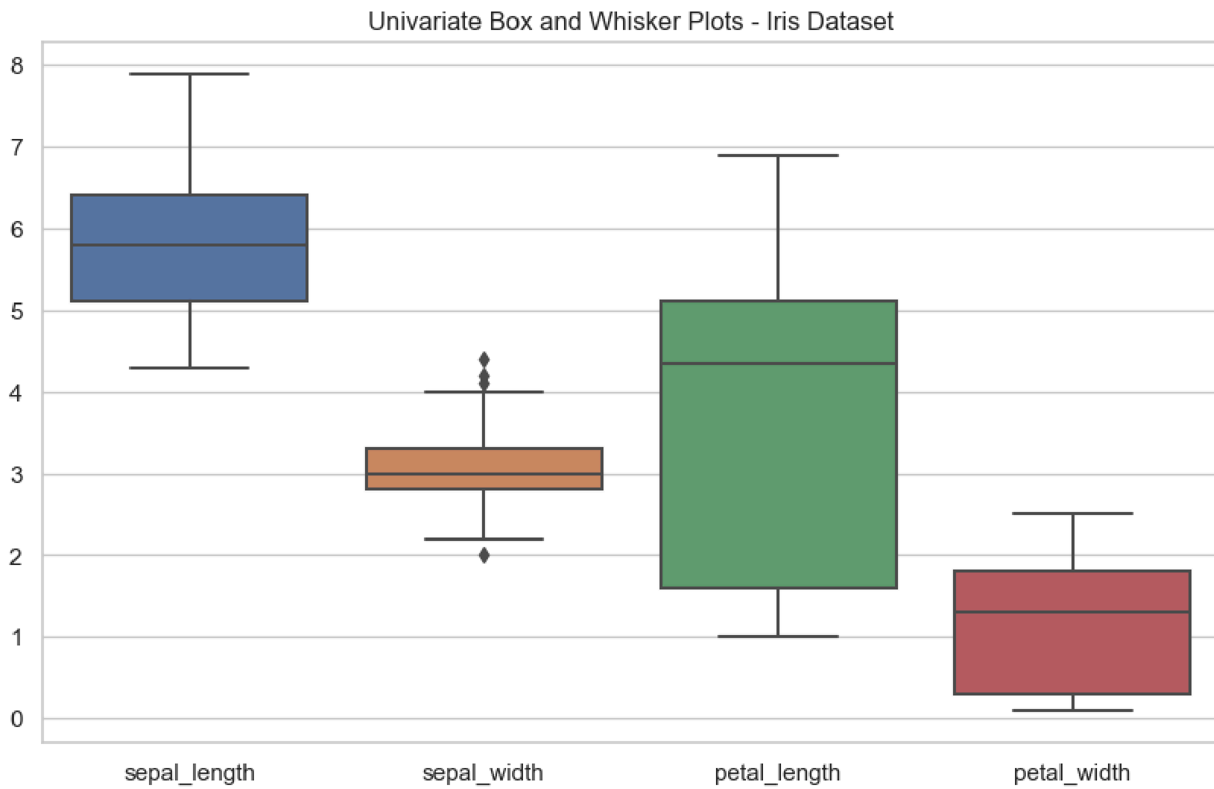
In [ ]:

```
# To see the class distribution
class_dist = df['species'].value_counts()

# Print class distribution
print("Class Distribution:")
print(class_dist)
```

```
Class Distribution:
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: species, dtype: int64
```

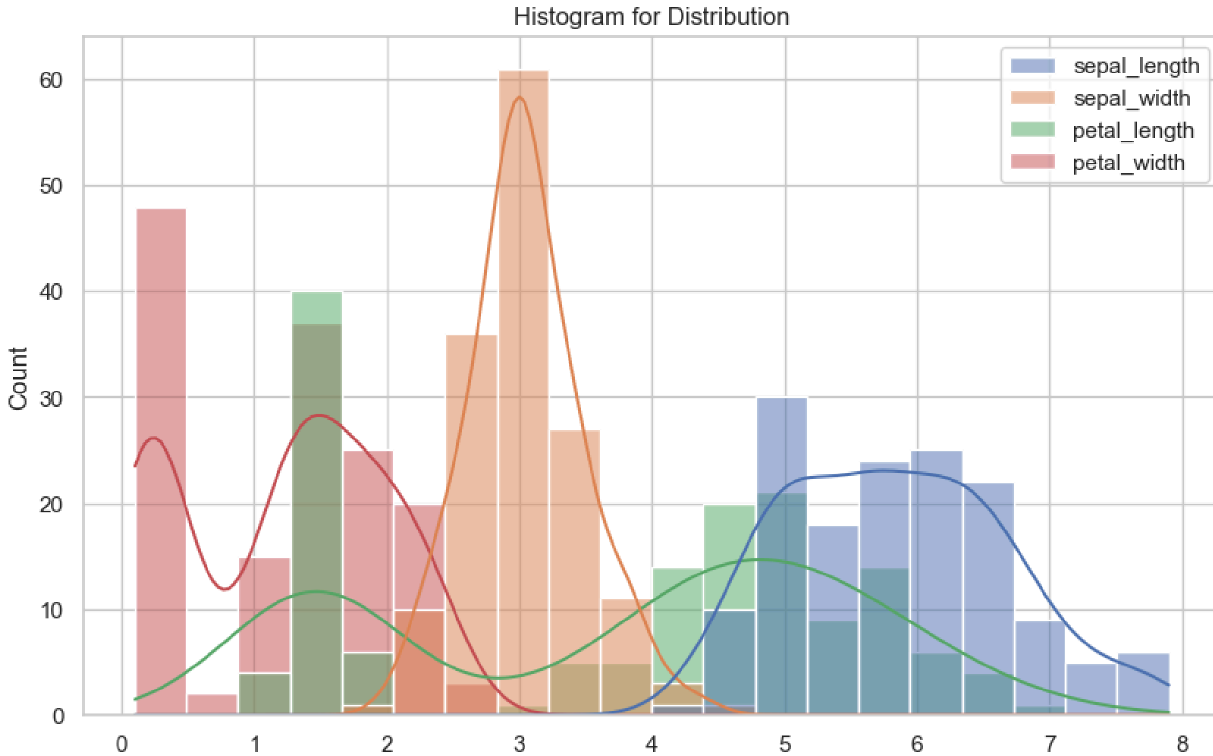
TO SHOW THE UNIVARIATE PLOT (BOX and WHISKER PLOTS)

In [ ]:
```
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))

# Display univariate plot
df_Frame = pd.DataFrame(df)

# Create box and whisker plots
sns.boxplot(data=df)
plt.title("Univariate Box and Whisker Plots - Iris Dataset")
plt.show()
```



TO SHOW THE HISTOGRAM FOR THE DISTRIBUTION

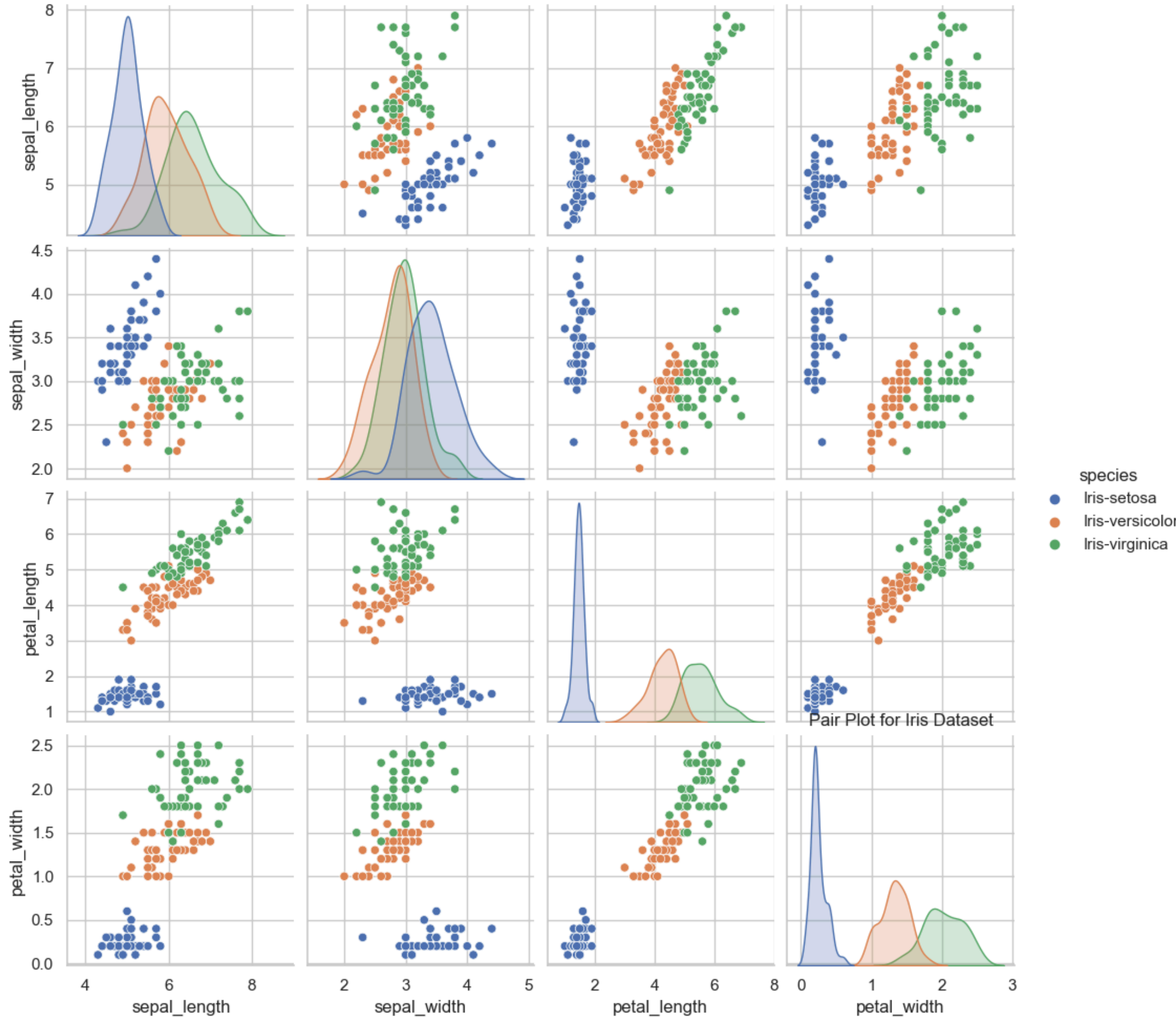
In [ ]:
```
# Histogram plot for the distribution
sns.set(style="whitegrid")
plt.figure(figsize=(10, 6))

# Create the histogram
sns.histplot(data=df, kde=True, bins=20)
plt.title('Histogram for Distribution')
plt.show()
```



FOR THE MULTIVARIATE PLOT

In [ ]:
```
# Create a pair plot
sns.set(style="whitegrid")

plt.figure(figsize=(15, 10))
sns.pairplot(df, hue="species")
plt.title("Pair Plot for Iris Dataset")
plt.show()
```

```
<Figure size 1500x1000 with 0 Axes>
```



TO CREATE THE MATRIX OF INDEPENDENT VARIABLE, X

In [ ]:
```
# Identify and select the columns that represent independent variables
selected_features = ['sepal_length', 'sepal_width', 'petal_length', 'petal_wid

# Create the matrix of independent variables, in terms of X
X = df[selected_features]

# Display the matrix of X
print("Marix of Independent variables, X: \n", X)
```

```
Marix of Independent variables, X:
      sepal_length  sepal_width  petal_length  petal_width
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

[150 rows x 4 columns]
```

TO CREATE THE MATRIX OF DEPENDENT VARIABLE, Y

In [ ]:
```
# Display the matrix of dependent variable, in terms of  Y
Y = df[['species']]

print("Matrix of Dependent Variable, Y: \n", Y)
```

```
Matrix of Dependent Variable, Y:
              species
0        Iris-setosa
1        Iris-setosa
2        Iris-setosa
3        Iris-setosa
4        Iris-setosa
..               ...
145   Iris-virginica
146   Iris-virginica
147   Iris-virginica
148   Iris-virginica
149   Iris-virginica

[150 rows x 1 columns]
```

TO ENCODE THE CATEGORICAL DATA IN THE DEPENDENT VARIABLE, Y

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
# Dependent column specification, Y = species
Y = 'species'

# Use label encoding for categorical data
label_encoder = LabelEncoder()
df['Encoded_Category'] = label_encoder.fit_transform(df[Y])

print("Categorical Data Encoded in the Dependent Variable, Y: \n \n ", df)
```

```
Categorical Data Encoded in the Dependent Variable, Y:

     sepal_length  sepal_width  petal_length  petal_width          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   
..            ...          ...           ...          ...              ...   
145           6.7          3.0           5.2          2.3   Iris-virginica   
146           6.3          2.5           5.0          1.9   Iris-virginica   
147           6.5          3.0           5.2          2.0   Iris-virginica   
148           6.2          3.4           5.4          2.3   Iris-virginica   
149           5.9          3.0           5.1          1.8   Iris-virginica   

     Encoded_Category  
0                   0  
1                   0  
2                   0  
3                   0  
4                   0  
..                ...  
145                 2  
146                 2  
147                 2  
148                 2  
149                 2  

[150 rows x 6 columns]
```

A. Evaluation Procedure Number 1: Train and Test on the entire dataset.

A.1. USING SUPPORT VECTOR MACHINE

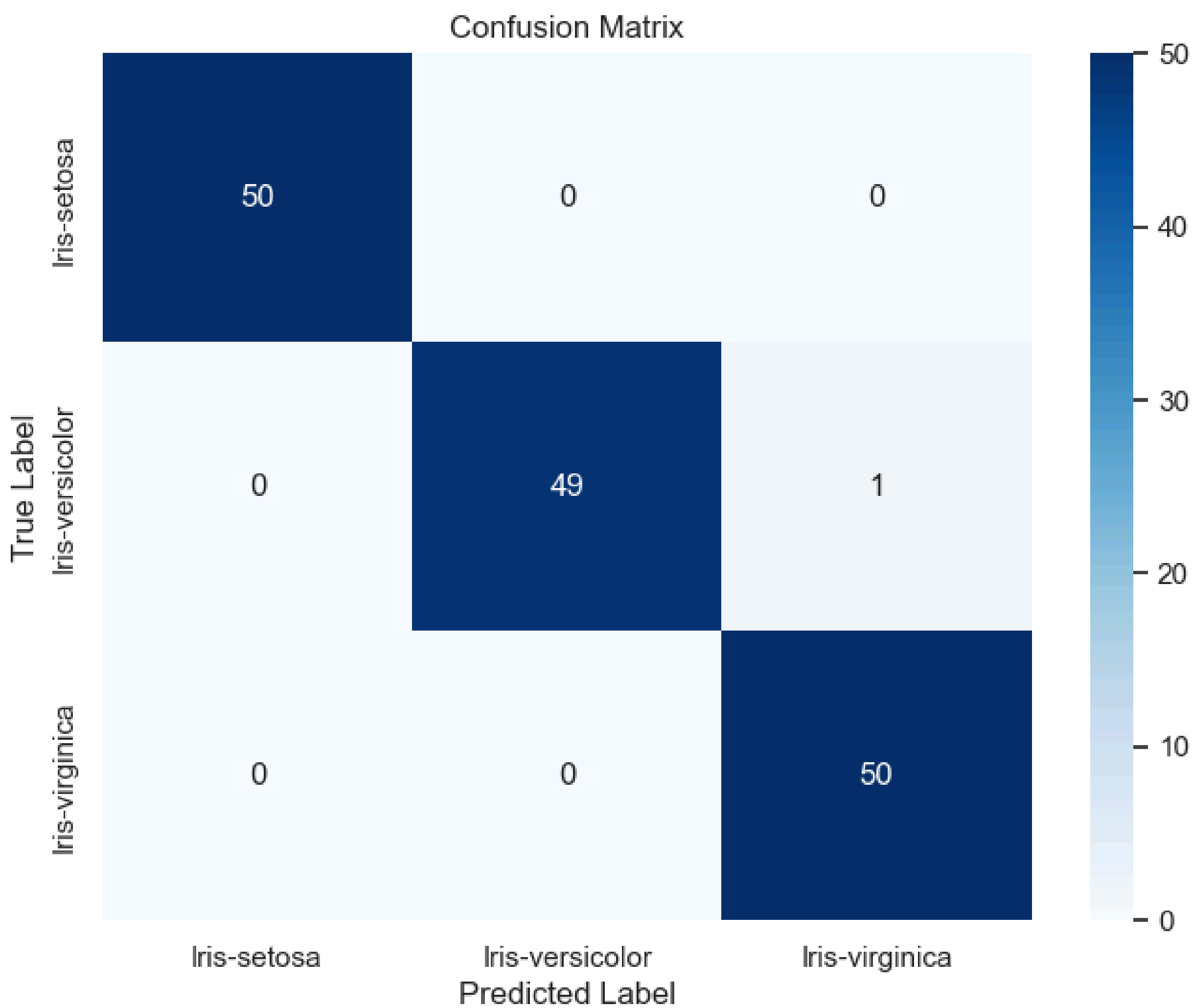
A.1.a To Create the Support Vector Machine Model

In [ ]:
```python
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = df['species']

# Split the dataset into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand

# Initialize the SVM model
svm_model = SVC(kernel='linear')

# Fit the model
svm_model.fit(X, y)

# Make predictions on the test set
y_pred = svm_model.predict(X)
```

A.1.b To Evaluate the Performance of the Support Vector Machine Model

In [ ]:
```
#To evaluate the Performance of SVM Model
from sklearn.metrics import confusion_matrix, accuracy_score, classification_re

# To Show the Confusion Matrix
c_matrix = confusion_matrix(y, y_pred)

# Print the results
print("Confusion Matrix:\n", c_matrix)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(c_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=svm_model
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Confusion Matrix:
 [[50  0  0]
 [ 0 49  1]
 [ 0  0 50]]
```



In [ ]:
```
# For the Classification Accuracy

# Evaluate the model performance
accuracy_svm = accuracy_score(y, y_pred)
```

```
# Print the results
print("Classification Accuracy: ", accuracy_svm)
```

```
Classification Accuracy:  0.9933333333333333
```

In [ ]:
```
# For the Classification Report

# Evaluate the model performance
classification_rep_svm = classification_report(y, y_pred)

# Print the results
print("Classification Report:\n", classification_rep_svm)
```

```
Classification Report:
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        50
Iris-versicolor       1.00      0.98      0.99        50
 Iris-virginica       0.98      1.00      0.99        50

       accuracy                           0.99       150
      macro avg       0.99      0.99      0.99       150
   weighted avg       0.99      0.99      0.99       150
```

A.2. USING LOGISTIC REGRESSION

A.2.a Create Logistic Regression Model

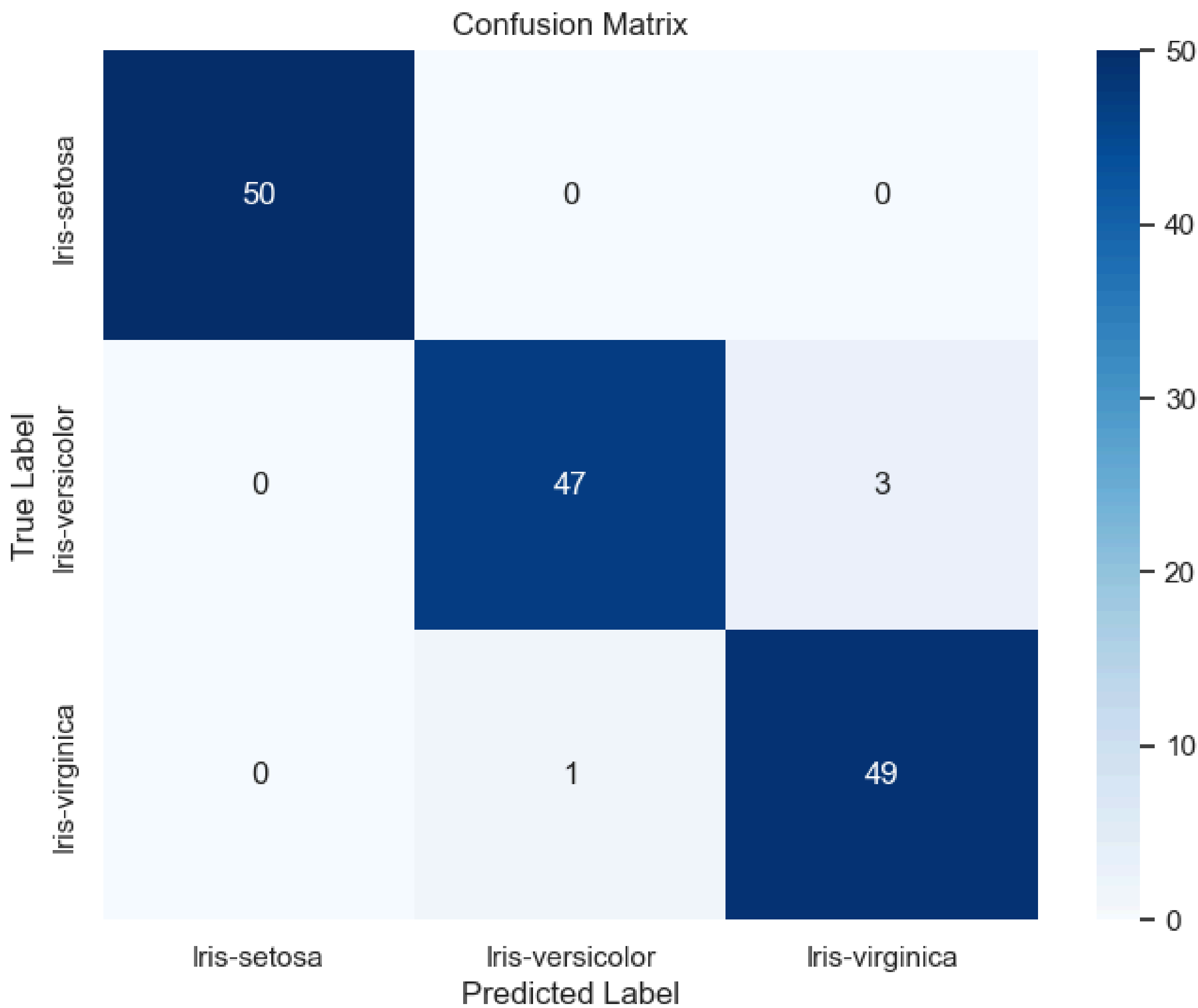
In [ ]:
```
from sklearn.linear_model import LogisticRegression

# Defining independent and dependent variables
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = df['species']

# Create a Logistic Regression model
logreg_model = LogisticRegression(max_iter=1000, random_state=42)

# Fit the model
logreg_model.fit(X, y)

# Make predictions on the entire dataset
y_pred = logreg_model.predict(X)
```

A.2.b To Evaluate the Performance of the Logistic Regression Model

In [ ]:
```
#To evaluate the Performance of Logistic Regression Model
from sklearn.metrics import confusion_matrix, accuracy_score, classification_re

# To Show the Confusion Matrix
conf_matrix = confusion_matrix(y, y_pred)

# Print the results
print("Confusion Matrix:\n", conf_matrix)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=logreg_
plt.title("Confusion Matrix")
```

```
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Confusion Matrix:
 [[50  0  0]
 [ 0 47  3]
 [ 0  1 49]]
```



In [ ]:
```
# For the Classification Accuracy

# Evaluate the model performance
accuracy_logreg = accuracy_score(y, y_pred)

# Print the results
print("Classification Accuracy: ", accuracy_logreg)
```

```
Classification Accuracy:  0.9733333333333334
```

In [ ]:
```
# For the Classification Report

# Evaluate the model performance
classification_rep_logreg = classification_report(y, y_pred)

# Print the results
print("Classification Report:\n", classification_rep_logreg)
```

```
Classification Report:
                  precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        50
Iris-versicolor       0.98      0.94      0.96        50
 Iris-virginica       0.94      0.98      0.96        50

       accuracy                           0.97       150
      macro avg       0.97      0.97      0.97       150
   weighted avg       0.97      0.97      0.97       150
```

A.3. USING K NEAREST NEIGHBOR WITH K = 5

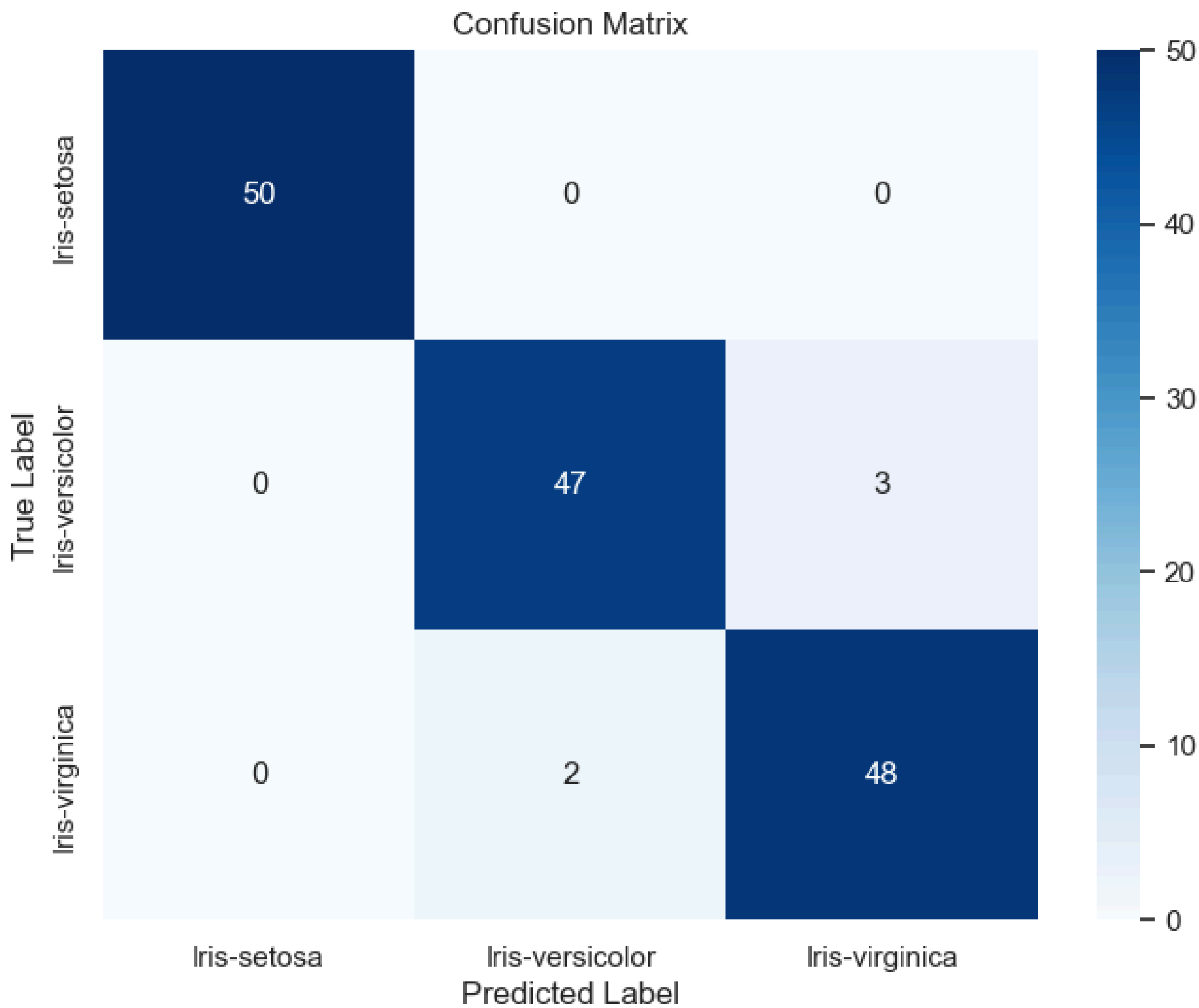
A.3.a Create K Nearest Neighbors model

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier

# Defining independent and dependent variables
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = df['species']

# Create a K Nearest Neighbors (KNN) model with k=5
knn_model = KNeighborsClassifier(n_neighbors=5)

# Fit the model
knn_model.fit(X, y)

# Make predictions on the entire dataset
y_pred = knn_model.predict(X)
```

A.2.b To Evaluate the Performance of the K Nearest Neighbors model

In [ ]:
```
#To evaluate the Performance of Logistic Regression Model
from sklearn.metrics import confusion_matrix, accuracy_score, classification_re

# To Show the Confusion Matrix
conf_matrix = confusion_matrix(y, y_pred)

# Print the results
print("Confusion Matrix:\n", conf_matrix)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=knn_mod
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Confusion Matrix:
 [[50  0  0]
 [ 0 47  3]
 [ 0  2 48]]
```



In [ ]:
```
# For the Classification Accuracy

# Evaluate the model performance
accuracy_knn = accuracy_score(y, y_pred)

# Print the results
print("Classification Accuracy: ", accuracy_knn)
```

```
Classification Accuracy:  0.9666666666666667
```

In [ ]:
```
# For the Classification Report

# Evaluate the model performance
classification_rep_knn = classification_report(y, y_pred)

# Print the results
print("Classification Report:\n", classification_rep_knn)
```

```
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        50
Iris-versicolor       0.96      0.94      0.95        50
 Iris-virginica       0.94      0.96      0.95        50

       accuracy                           0.97       150
      macro avg       0.97      0.97      0.97       150
   weighted avg       0.97      0.97      0.97       150
```

B. Evaluation Procedure Number Two: Training and Testing Dataset Splitting

TO SPLIT THE DATASET INTO TRAINING DATASET AND TESTING DATASET

In [ ]:
```
from sklearn.model_selection import train_test_split

# Defining independent and dependent variables
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
Y = df['species']

# Split the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, rando
```

In [ ]:
```
# To Show the Shapes of X and Y Data
print("X shape: ", X.shape)
print("Y shape: ", Y.shape)
```

```
X shape:  (150, 4)
Y shape:  (150,)
```

In [ ]:
```
# To Show the Shapes of X_train and Y_train Data
print("X_train shape:", X_train.shape)
print("Y_train shape:", Y_train.shape)
```

```
X_train shape: (120, 4)
Y_train shape: (120,)
```

In [ ]:
```
# To Show the Shapes of X_test and Y_test Data
print("X_test shape:", X_test.shape)
print("Y_test shape:", Y_test.shape)
```

```
X_test shape: (30, 4)
Y_test shape: (30,)
```

B.1 USING SUPPORT VECTOR MACHINE

B.1.a To Create the Support Vector Machine Model

In [ ]:
```
# Create an SVM classifier
svm_classifier = SVC(kernel='linear', C=1.0)

# Train the classifier
svm_classifier.fit(X_train, Y_train)
```

```
# Make predictions on the test set
y_pred = svm_classifier.predict(X_test)
```

B.1.b To Evaluate the Performance of the Suport Vector Machine Model

In [ ]:
```
# Show the Confusion Matrix and Evaluate
accuracy_svm = accuracy_score(Y_test, y_pred)
conf_matrix_svm = confusion_matrix(Y_test, y_pred)

print(f"Accuracy: {accuracy_svm}")
print(f"Confusion Matrix:\n{conf_matrix_svm}")
```

```
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

In [ ]:
```
# For the Classification Accuracy
print(f"Classification Accuracy: {accuracy_score(Y_test, y_pred)}")
```

```
Classification Accuracy: 1.0
```

In [ ]:
```
# For the Classification Report
print(f"Classification Report:\n{classification_report(Y_test, y_pred)}")
```

```
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       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
```

B.2 USING LOGISTIC REGRESSION

In [ ]:
```
# Create a Logistic Regression classifier
logreg_classifier = LogisticRegression()

# Train the classifier
logreg_classifier.fit(X_train, Y_train)

# Make predictions on the test set
y_pred = logreg_classifier.predict(X_test)
```

In [ ]:
```
# Show the Confusion Matrix and Evaluate
accuracy_svm = accuracy_score(Y_test, y_pred)
conf_matrix_svm = confusion_matrix(Y_test, y_pred)

print(f"Accuracy: {accuracy_svm}")
print(f"Confusion Matrix:\n{conf_matrix_svm}")
```

```
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

In [ ]:
```
# For the Classification Accuracy
print(f"Classification Accuracy: {accuracy_score(Y_test, y_pred)}")
```

```
Classification Accuracy: 1.0
```

In [ ]:
```
# For the Classification Report
print(f"Classification Report:\n{classification_report(Y_test, y_pred)}")
```

```
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       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
```

B.3 USING K NEAREST NEIGHBOR WITH K = 5

In [ ]:
```
# Create KNN Classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)

# Train the classifier
knn_classifier.fit(X_train, Y_train)

# Make predictions on the test set
y_pred = knn_classifier.predict(X_test)
```

In [ ]:
```
# Show the Confusion Matrix and Evaluate
accuracy_svm = accuracy_score(Y_test, y_pred)
conf_matrix_svm = confusion_matrix(Y_test, y_pred)

print(f"Accuracy: {accuracy_svm}")
print(f"Confusion Matrix:\n{conf_matrix_svm}")
```

```
Accuracy: 1.0
Confusion Matrix:
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

In [ ]:
```
# For the Classification Accuracy
print(f"Classification Accuracy: {accuracy_score(Y_test, y_pred)}")
```

```
Classification Accuracy: 1.0
```

In [ ]:
```
# For the Classification Report
print(f"Classification Report:\n{classification_report(Y_test, y_pred)}")
```

```
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        10
Iris-versicolor       1.00      1.00      1.00         9
 Iris-virginica       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
```

C. Evaluation Procedure Number 3: K-fold Cross Validation

C.1 USING SUPPORT VECTOR MACHINE

In [ ]:
```
# To Apply K-fold Cross Validation for the Support Vector Machine Model Perforn
from sklearn.model_selection import cross_val_score

# Create a SVM classifier
svm_classifier = SVC(kernel='linear', C=1.0)

# Train the classifier
svm_classifier.fit(X_train, Y_train)

# Perform 5-fold cross validation
scores = cross_val_score(estimator=svm_classifier, X=X_train, y=Y_train, cv=5)

# Print the accuracy for each fold:
print(scores)
```

```
[1.         0.95833333 0.875      1.         0.95833333]
```

C.2 USING LOGISTIC REGRESSION

In [ ]:
```
# To apply K-fold Cross Validation for the Logistic Regression Model Performanc

# Create a Logistic Regression classifier
logreg_classifier = LogisticRegression()

# Train the classifier
logreg_classifier.fit(X_train, Y_train)

# Perform 5-fold cross validation
scores = cross_val_score(estimator=logreg_classifier, X=X_train, y=Y_train, cv=

# Print the accuracy for each fold:
print(scores)
```

```
[1.         1.         0.875      1.         0.95833333]
```

```
/Users/Shuahua/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_lo
gistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
/Users/Shuahua/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_lo
gistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
```

C.3 USING K NEAREST NEIGHBOR WITH K = 5

In [ ]:
```python
# To apply K-fold Cross Validation for the KNN Model Performance

# Create a KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5)

# Train the classifier
knn_classifier.fit(X_train, Y_train)

# Perform 5-fold cross validation
scores = cross_val_score(estimator=knn_classifier, X=X_train, y=Y_train, cv=5)

# Print the accuracy for each fold:
print(scores)
```

```
[0.95833333 0.95833333 0.83333333 1.         0.95833333]
```