Group Project CO559 By:

Enkh-Amgalan (Entwan) Enkhbayar (22135347)

Muneef Ahamed Mohamed Mumthas (22206529)

Crown Caesar (22038807)

Brunaldo Cimo (22209954)

Some Guys From Everywhere Doing AI

Introduction

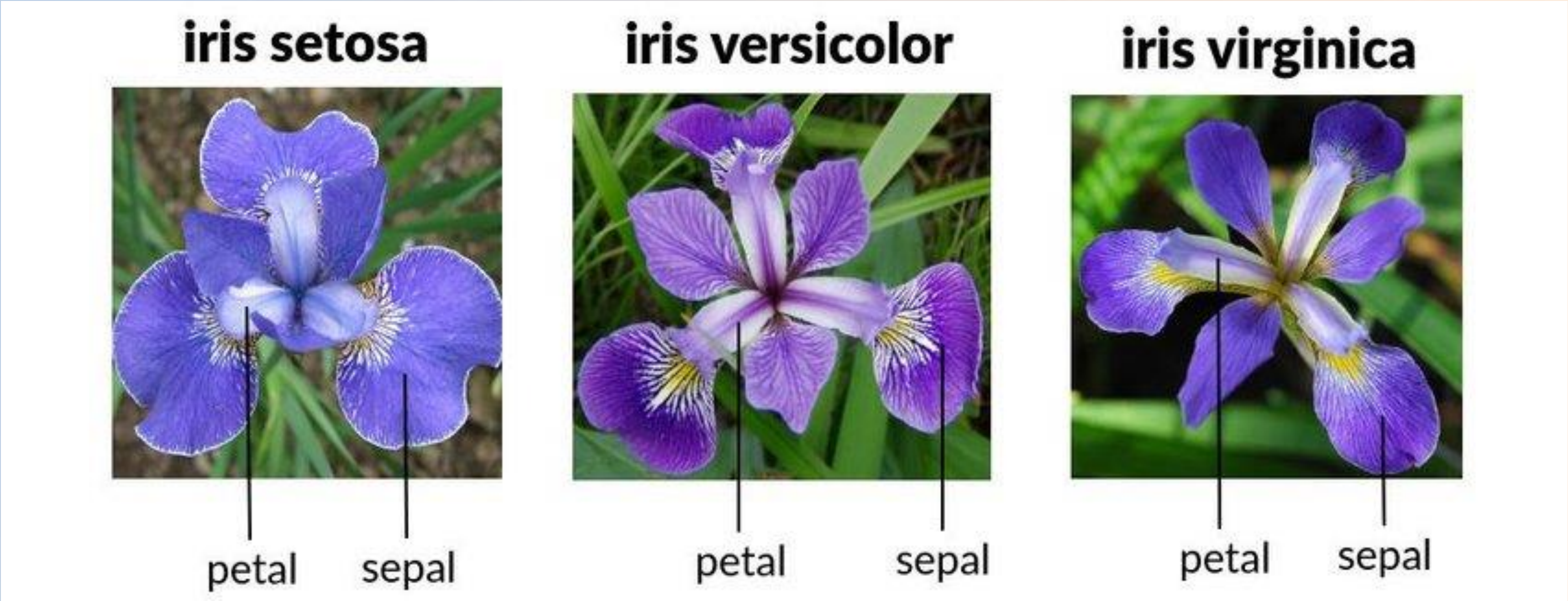
Objective: Apply machine learning algorithms and Deep learning Neural Network to classify species in the Iris dataset.

Datasets Overview: 1. Consists of measurements from three species of Iris (setosa, versicolor, virginica). 2. Consists of images of three different species of iris.

Machine Learning

With Iris CSV Dataset

Classification Analysis and ML prediction on Iris Dataset



Iris CSV dataset

Iris.csv ×

1 to 10 of 150 entries Filter

Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.3	Iris-setosa
8	5.0	3.4	1.5	0.2	Iris-setosa
9	4.4	2.9	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-setosa

Show 10 per page

1 2 10 15

IRIS.csv (4.62 kB)

kaggle

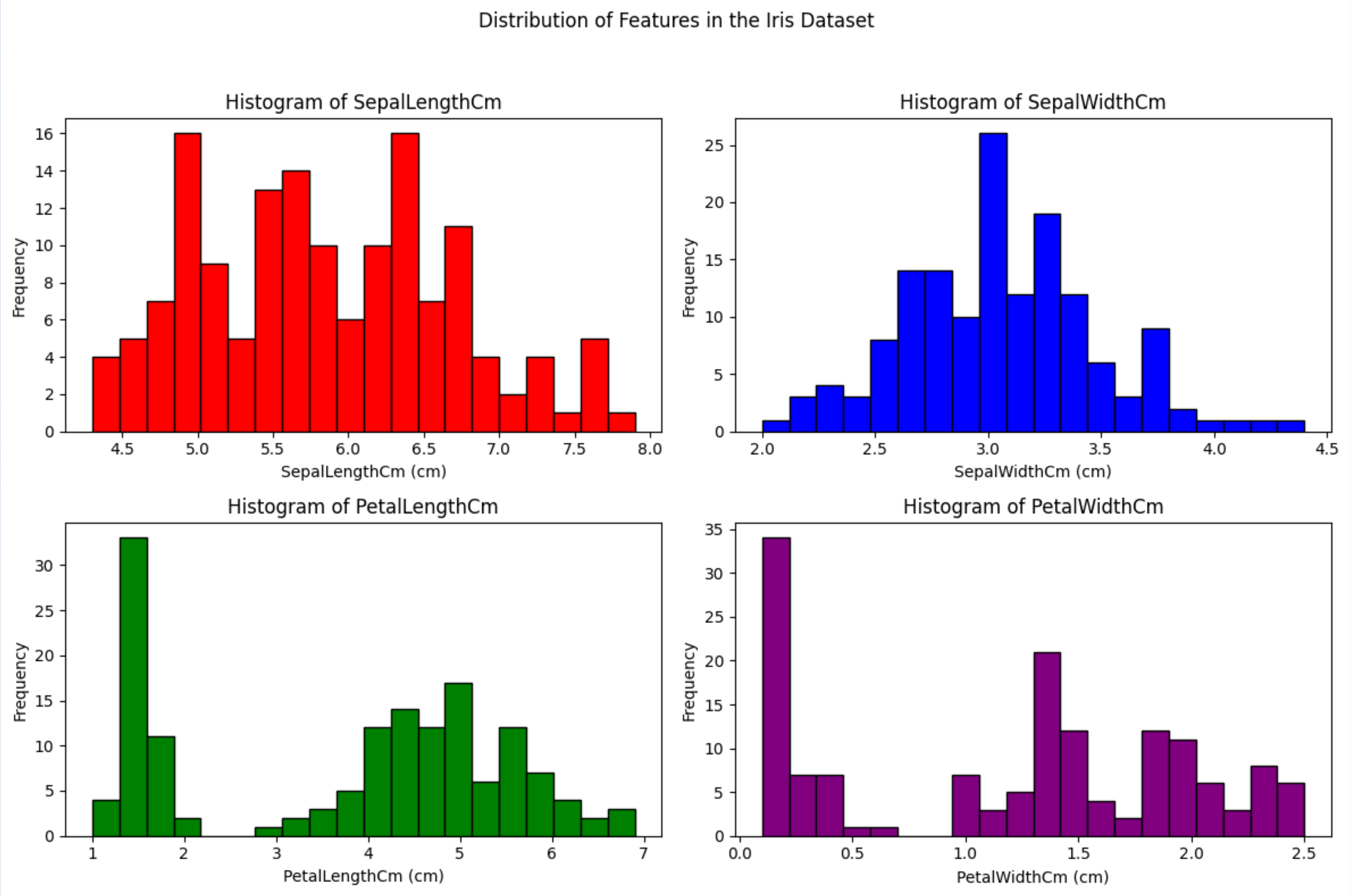
Iris csv dataset info

```
[8] iris.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
```

```
[6] iris.shape
    # 150 records and 5 columns

    (150, 5)
```

Distribution of Features in the Iris Dataset

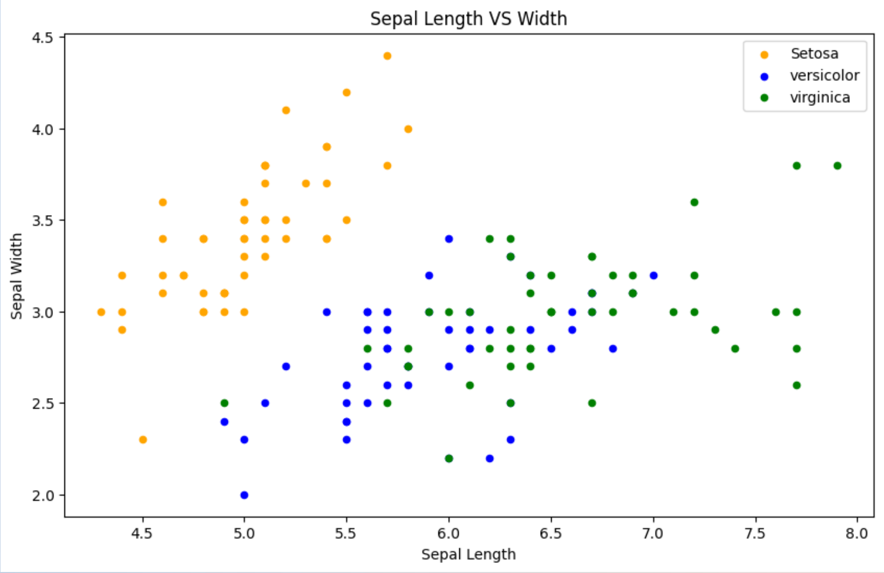
```
import matplotlib.pyplot as plt
import seaborn as sns
```

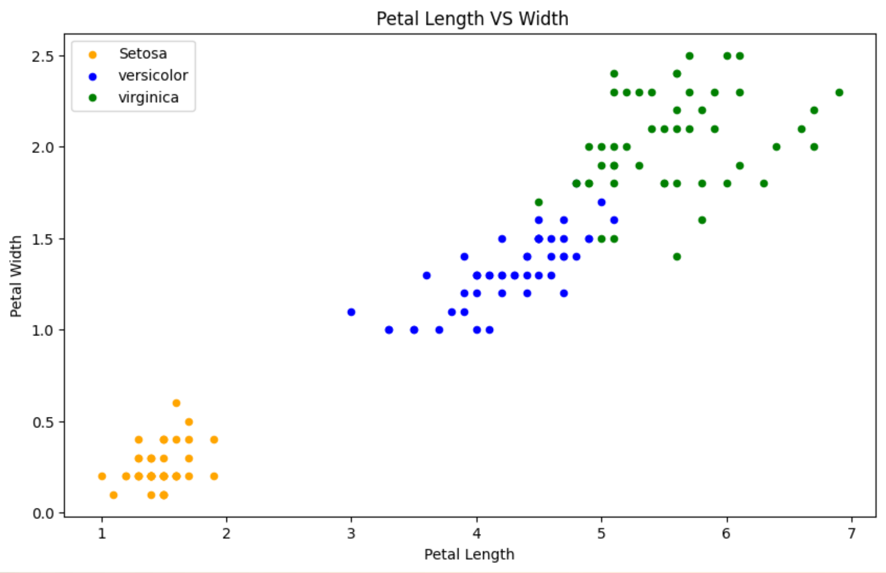


Code block for Histogram

```python
# Define a list of colors for each feature
colors = ['red', 'blue', 'green', 'purple']

# Plot histograms for each feature in the dataset
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(12, 8))
fig.suptitle('Distribution of Features in the Iris Dataset')
features = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']
for i, ax in enumerate(axes.flat):
    ax.hist(iris[features[i]], bins=20, color=colors[i], edgecolor='black')
    ax.set_title(f'Histogram of {features[i]}')
    ax.set_xlabel(f'{features[i]} (cm)')
    ax.set_ylabel('Frequency')

plt.tight_layout(rect=[0, 0, 1, 0.95])
```

Width and Length of Petal and Sepal difference between all three Flowers





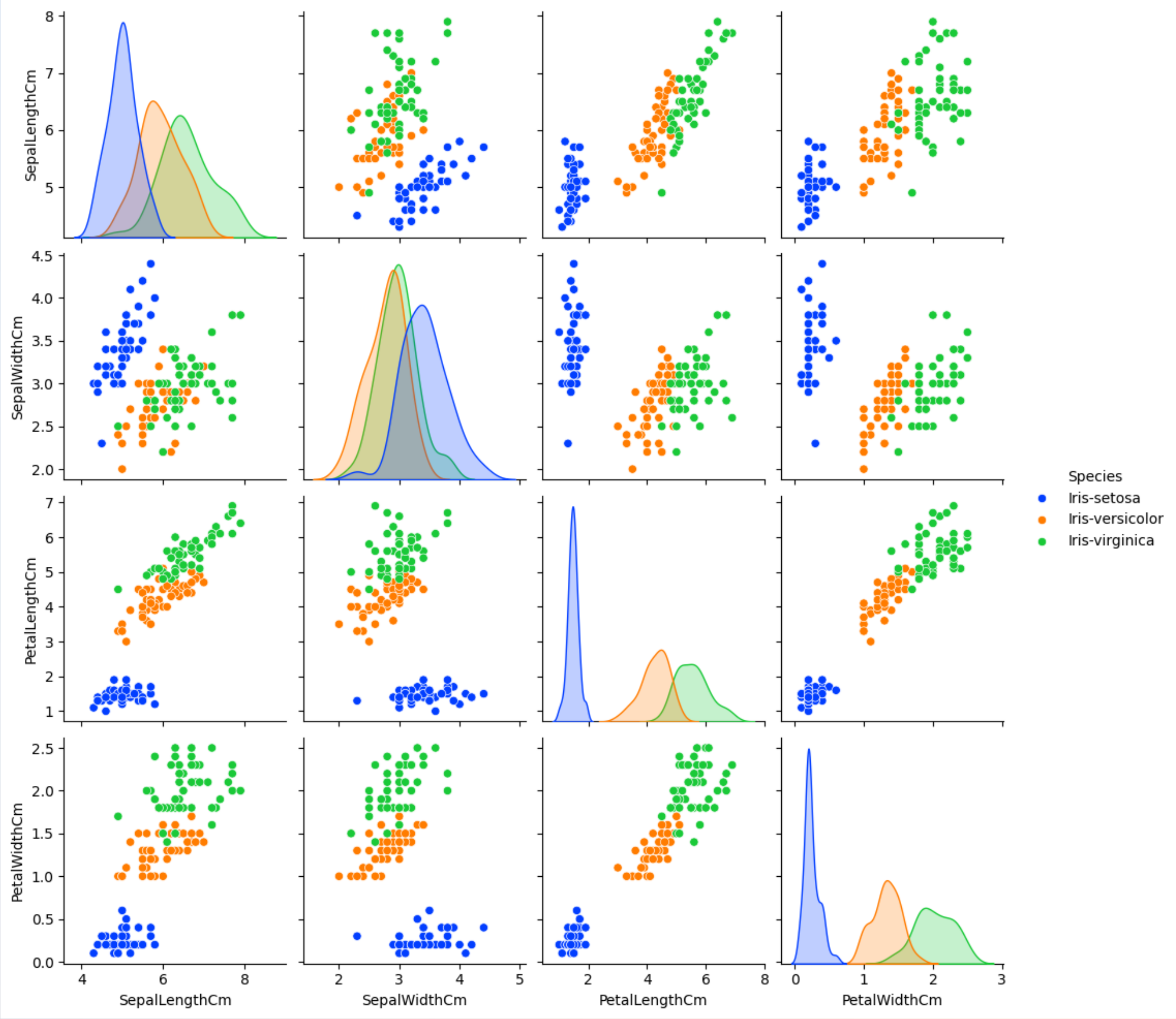
Code blocks for Scatter Plot

```
fig = iris[iris.Species=='Iris-setosa'].plot(kind='scatter',x='SepalLengthCm',y='SepalWidthCm',color='orange', label='Setosa')
iris[iris.Species=='Iris-versicolor'].plot(kind='scatter',x='SepalLengthCm',y='SepalWidthCm',color='blue', label='versicolor',ax=fig)
iris[iris.Species=='Iris-virginica'].plot(kind='scatter',x='SepalLengthCm',y='SepalWidthCm',color='green', label='virginica', ax=fig)
fig.set_xlabel("Sepal Length")
fig.set_ylabel("Sepal Width")
fig.set_title("Sepal Length VS Width")
# gcf - get current figure
fig=plt.gcf()
fig.set_size_inches(10,6)
plt.show()
```

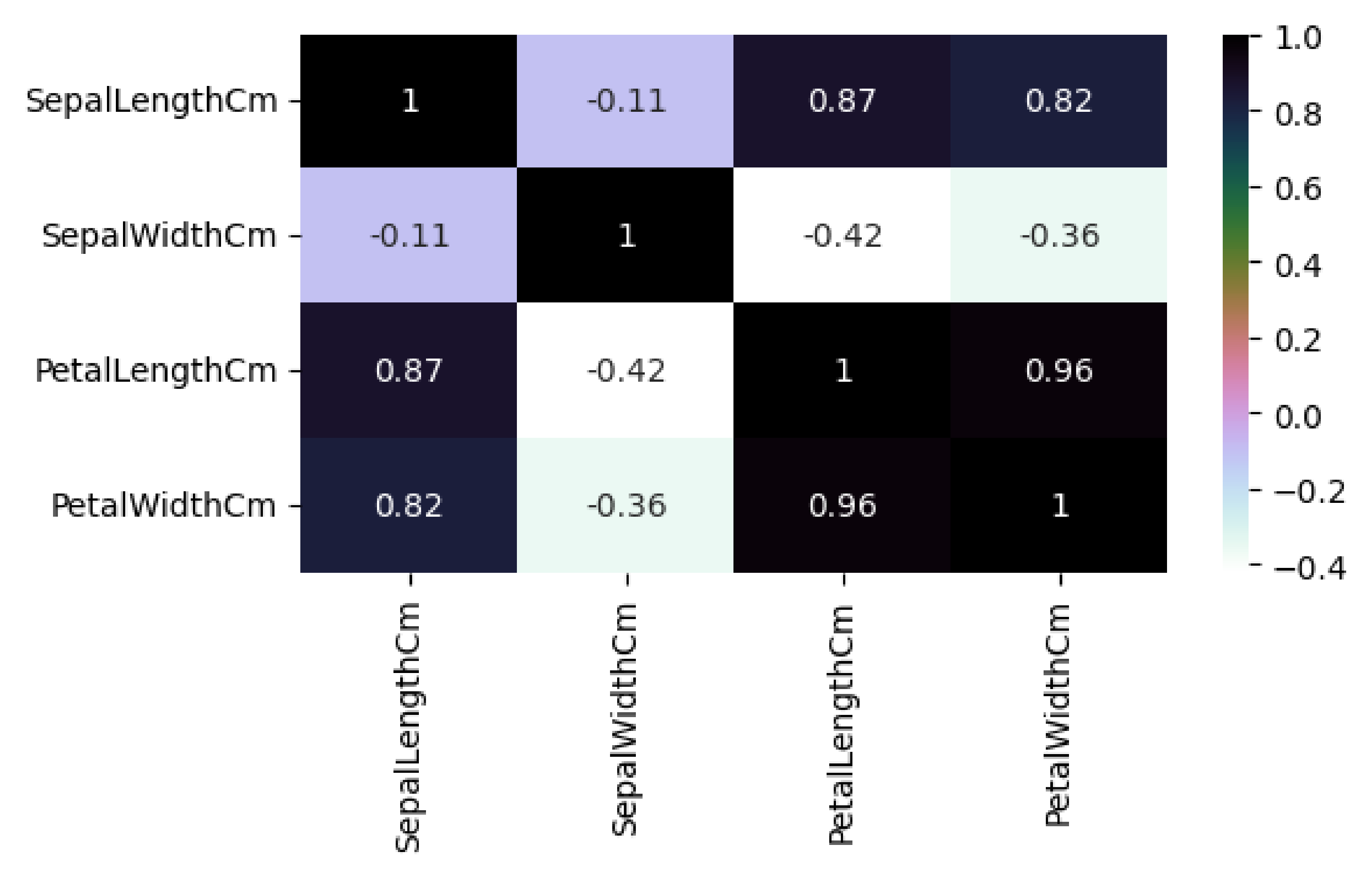
```
fig = iris[iris.Species=='Iris-setosa'].plot.scatter(x='PetalLengthCm',y='PetalWidthCm',color='orange', label='Setosa')
iris[iris.Species=='Iris-versicolor'].plot.scatter(x='PetalLengthCm',y='PetalWidthCm',color='blue', label='versicolor',ax=fig)
iris[iris.Species=='Iris-virginica'].plot.scatter(x='PetalLengthCm',y='PetalWidthCm',color='green', label='virginica', ax=fig)
fig.set_xlabel("Petal Length")
fig.set_ylabel("Petal Width")
fig.set_title(" Petal Length VS Width")
fig=plt.gcf()
fig.set_size_inches(10,6)
plt.show()
```

Iris Dataset Pairplot

```
sns.pairplot(iris, hue='Species', palette='bright')
```



Iris Dataset Heatmap




```
# draws  heatmap with input as the correlation matrix
plt.figure(figsize=(6,3))
sns.heatmap(iris.corr(numeric_only=True), annot=True, cmap='cubehelix_r')
plt.show()
```

Loading the dataset into data frame using pandas

```
[4] # This cell is specific to Google Colab
    from google.colab import drive
    drive.mount('/content/drive')
    # Mount Google Drive to access the Iris dataset CSV file
```

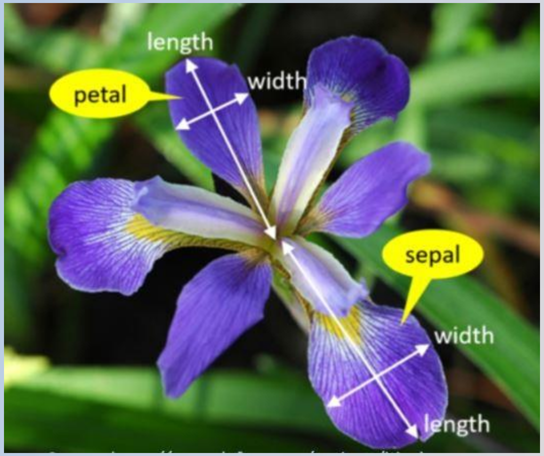
```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Importing the Iris Dataset

```
[5] # Define the path to the Iris dataset CSV file
    path = "/content/drive/MyDrive/Iris.csv"

    iris = pd.read_csv(path)
    # We removed the column ID since the dataframe is indexed.
    #Basically we dont need it
    # Also this data is very clean as well as small we dont have to do pre-process the data.
    iris.drop('Id', axis=1, inplace=True)

    iris.head()
    #Sampling the dataset
```

	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

Methodology



- Algorithms used: Decision Tree, Logistic Regression, SVM, KNN.
- Preprocessing: Data normalization, handling missing values.
- Model Evaluation: Confusion matrix, accuracy.

Importing all libraries

```
# Import essential libraries for data manipulation and visualization
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow import keras
import tensorflow_hub as hub
from tensorflow.keras.utils import to_categorical
from keras.callbacks import ModelCheckpoint, EarlyStopping
#Support Vector Machine (SVM) Algorithm
from sklearn import svm
#metrics is for checking the model accuracy
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

Preparing the data to feed the different algorithm models

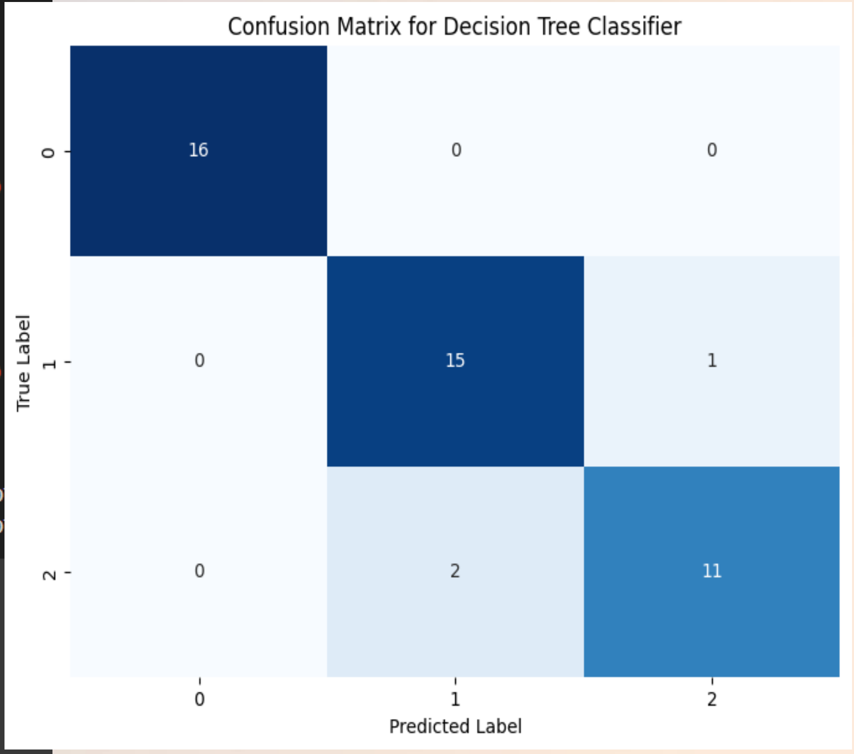
```
[ ] # Split the data into training and testing sets with a test size of 20%
    train_X, test_X, train_Y, test_Y = train_test_split(
        iris[["SepalLengthCm", "SepalWidthCm", "PetalLengthCm", "PetalWidthCm"]],
        iris["Species"],
        test_size=0.3,
        random_state = None
    )
    #output value of test data
    # In this one we are taking all features!
    # 1. X- Taking the training data features, output of our training data
    # 2. Y - Taking test data features, output value of test data
    # The dataset will be divided into two sets which are going to bed used for testing as well as training
    # 70% - training, 30% - testing
```

```
[ ] train_X.shape, train_Y.shape

    ((105, 4), (105,))
```

```
[ ] test_X.shape, test_Y.shape

    ((45, 4), (45,))
```

```
[149] # Decision Tree
      model= DecisionTreeClassifier()
      model.fit(train_x_petal,train_y_petal)
      prediction5 =model.predict(test_x_petal)
      accuracy_DT_p = metrics.accuracy_score(test_y_petal, prediction5)
      print(f'The accuracy of the Decision Tree using Petals is: {accuracy_DT_p:.12f}')

      model= DecisionTreeClassifier()
      model.fit(train_x_sepal,train_y_sepal)
      prediction6 =model.predict(test_x_sepal)
      accuracy_DT_s = metrics.accuracy_score(test_y_sepal, prediction6)
      print(f'The accuracy of the Decision Tree using Sepals is: {accuracy_DT_s:.12f}')

      conf_matrix_petalDT = confusion_matrix(test_y_petal, prediction5)
      conf_matrix_sepalDT = confusion_matrix(test_y_sepal, prediction6)
      print('\nConfusion Matrix for Decision Tree using petals: \n', conf_matrix_petalD
      print('\nConfusion Matrix for Decision Tree using sepals: \n', conf_matrix_sepalD
```

```
The accuracy of the Decision Tree using Petals is: 0.933333333333
The accuracy of the Decision Tree using Sepals is: 0.733333333333

Confusion Matrix for Decision Tree using petals:
 [[17  0  0]
 [ 0 14  0]
 [ 0  3 11]]

Confusion Matrix for Decision Tree using sepals:
 [[19  0  1]
 [ 0  7  8]
 [ 0  3  7]]
```



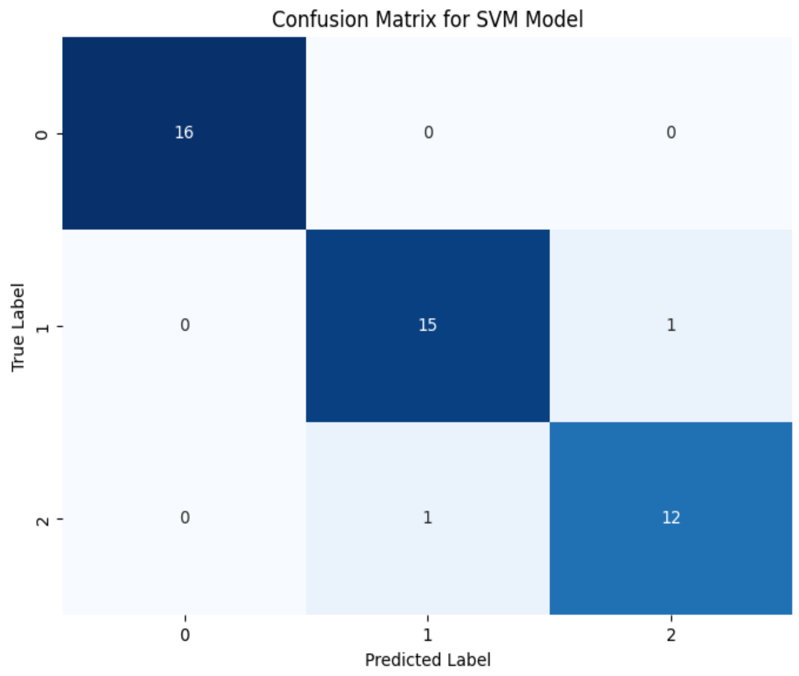
Decision Tree

```
[23] # Decision Tree Classifier
     model3 = DecisionTreeClassifier()
     model3.fit(train_X, train_Y)
     prediction3 = model3.predict(test_X) # Corrected to use model3 for prediction
     accuracy_DT = accuracy_score(test_Y, prediction3)
     print(f'The accuracy of the Decision Tree is: {accuracy_DT:.12f}')

     conf_matrix_dt = confusion_matrix(test_Y, prediction3)

     # Plot the confusion matrix
     plt.figure(figsize=(8, 6))
     sns.heatmap(conf_matrix_dt, annot=True, fmt='d', cmap='Blues', cbar=False)
     plt.title('Confusion Matrix for Decision Tree Classifier')
     plt.xlabel('Predicted Label')
     plt.ylabel('True Label')
     plt.show()
```

```
The accuracy of the Decision Tree is: 0.933333333333
```

```python
#SVC
model=svm.SVC()
model.fit(train_x_petal,train_y_petal)
prediction1=model.predict(test_x_petal)
accuracy_SVM_p = metrics.accuracy_score(test_y_petal, prediction1)
print(f'The accuracy of the SVM using Petals is: {accuracy_SVM_p:.12f}')

model=svm.SVC()
model.fit(train_x_sepal,train_y_sepal)
prediction2=model.predict(test_x_sepal)
accuracy_SVM_s = metrics.accuracy_score(test_y_sepal, prediction2)
print(f'The accuracy of the SVM using Sepals is: {accuracy_SVM_s:.12f}')


conf_matrix_petalSVC = confusion_matrix(test_y_petal, prediction1)
conf_matrix_sepalSVC = confusion_matrix(test_y_sepal, prediction2)
print('\nConfusion Matrix for SVC using petals: \n', conf_matrix_petalSVC)
print('\nConfusion Matrix for SVC using sepals: \n', conf_matrix_sepalSVC)
```

```
The accuracy of the SVM using Petals is: 1.000000000000
The accuracy of the SVM using Sepals is: 0.822222222222

Confusion Matrix for SVC using petals:
 [[15  0  0]
 [ 0 13  0]
 [ 0  0 17]]

Confusion Matrix for SVC using sepals:
 [[16  0  0]
 [ 0 12  4]
 [ 0  4  9]]
```



SVC

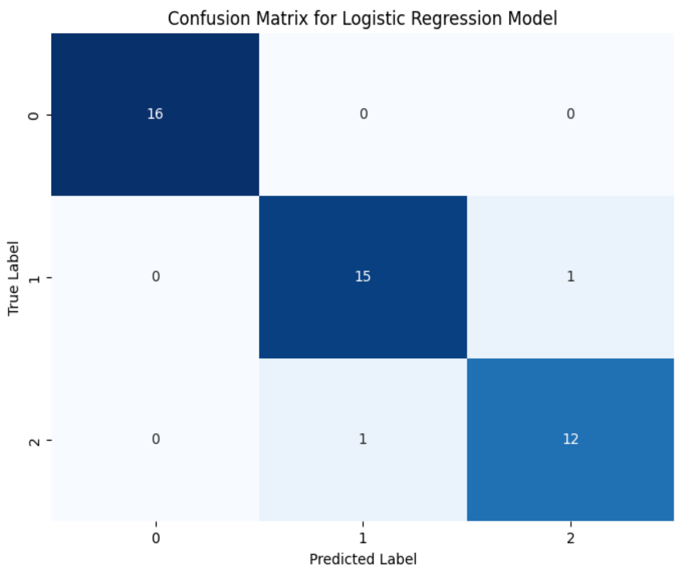
```python
# Initialize and train the Support Vector Machine (SVM) model
model = svm.SVC()
model.fit(train_X,train_Y)
prediction1 = model.predict(test_X)
accuracy_svm = accuracy_score(test_Y, prediction1)
print(f'The accuracy of the SVM is: {accuracy_svm:.12f}')

conf_matrix = confusion_matrix(test_Y, prediction1)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix for SVM Model')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

print(conf_matrix)
```

```
The accuracy of the SVM is: 0.955555555556
```

Logistic Regression

```
#Logistic Regression
model= LogisticRegression()
model.fit(train_x_petal,train_y_petal)
prediction3 = model.predict(test_x_petal)
accuracy_LR_p = metrics.accuracy_score(test_y_petal, prediction3)
print(f'The accuracy of the Logistic Regression using Petals is: {accuracy_LR_p:.12f}')

model= LogisticRegression()
model.fit(train_x_sepal,train_y_sepal)
prediction4 = model.predict(test_x_sepal)
accuracy_LR_s = metrics.accuracy_score(test_y_sepal, prediction4)
print(f'The accuracy of the Logistic Regression using Sepals is: {accuracy_LR_s:.12f}')

conf_matrix_petalLR = confusion_matrix(test_y_petal, prediction3)
conf_matrix_sepalLR = confusion_matrix(test_y_sepal, prediction4)
print('\nConfusion Matrix for Logistic Regression using petals: \n', conf_matrix_petalLR)
print('\nConfusion Matrix for Logistic Regression using sepals: \n', conf_matrix_sepalLR)
```

```
The accuracy of the Logistic Regression using Petals is: 0.977777777778
The accuracy of the Logistic Regression using Sepals is: 0.777777777778

Confusion Matrix for Logistic Regression using petals:
 [[15  0  0]
 [ 0 12  1]
 [ 0  0 17]]

Confusion Matrix for Logistic Regression using sepals:
 [[16  0  0]
 [ 1 10  5]
 [ 0  4  9]]
```




```
# Logistic Regression Model
model2 = LogisticRegression(max_iter=200) # Increased max_iter for convergence
model2.fit(train_X, train_Y)
prediction2 = model2.predict(test_X) # Corrected to use model2 for prediction
accuracy_LR = accuracy_score(test_Y, prediction2)
print(f'The accuracy of the Logistic Regression is: {accuracy_LR:.12f}')

conf_matrix_LR = confusion_matrix(test_Y, prediction2)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_LR, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix for Logistic Regression Model')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

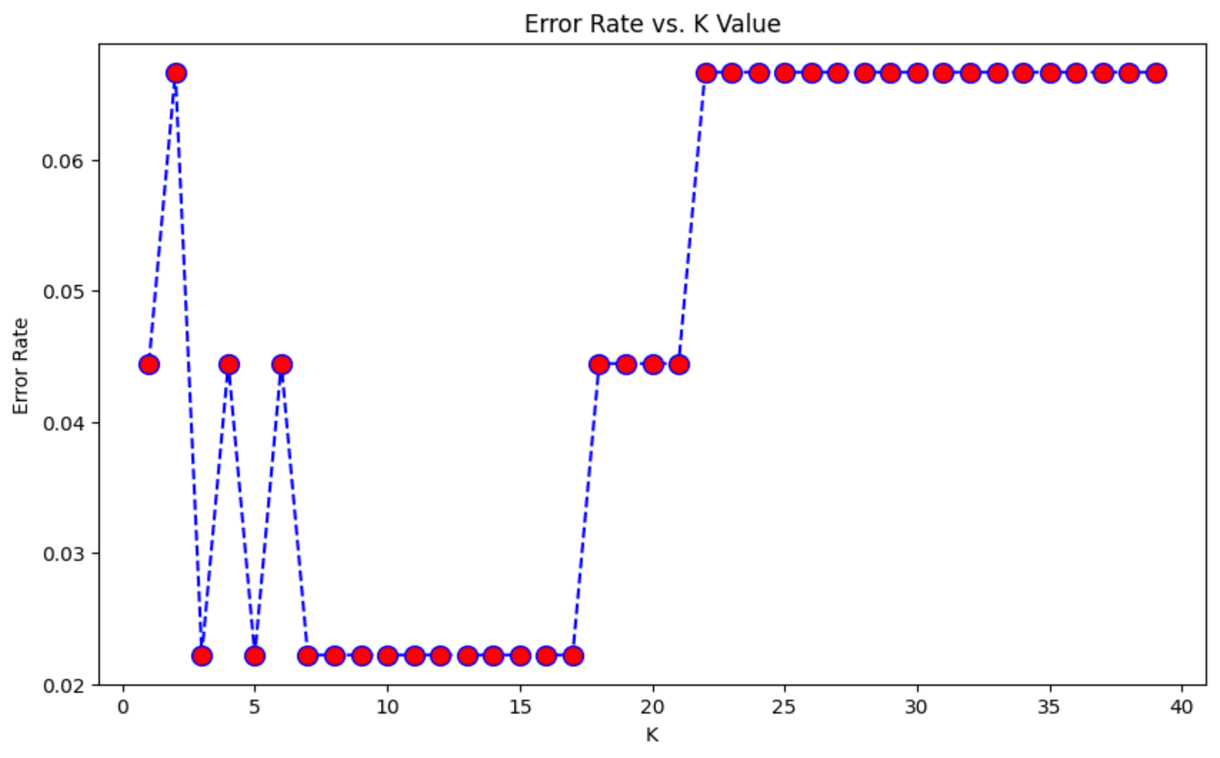
```
The accuracy of the Logistic Regression is: 0.955555555556
```

KNN Finding the perfect K- value

```python
error_rate = []

# Will take some time
for i in range(1,40):

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(train_X,train_Y)
    pred_i = knn.predict(test_X)
    error_rate.append(np.mean(pred_i != test_Y))
```

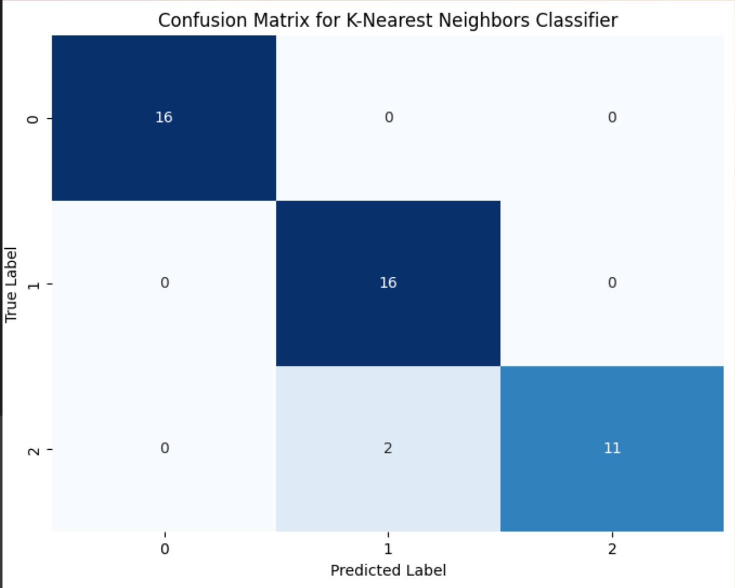
```python
[25] plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

```
Text(0, 0.5, 'Error Rate')
```



```
# k nearest Neighbour
model= KNeighborsClassifier(n_neighbors=8)
model.fit(train_x_petal,train_y_petal)
prediction7=model.predict(test_x_petal)
accuracy_KNN_p = metrics.accuracy_score(test_y_petal, prediction7)
print(f'The accuracy of the k nearest Neighbour using Petals is: {accuracy_KNN_p:.12f}')

model= KNeighborsClassifier(n_neighbors=8)
model.fit(train_x_sepal,train_y_sepal)
prediction8=model.predict(test_x_sepal)
accuracy_KNN_s = metrics.accuracy_score(test_y_sepal, prediction8)
print(f'The accuracy of the k nearest Neighbour using Sepals is: {accuracy_KNN_s:.12f}')

conf_matrix_petalKNN = confusion_matrix(test_y_petal, prediction7)
conf_matrix_sepalKNN = confusion_matrix(test_y_sepal, prediction8)
print('\nConfusion Matrix for k nearest Neighbour using petals: \n', conf_matrix_petalKNN)
print('\nConfusion Matrix for k nearest Neighbour sepals: \n', conf_matrix_sepalKNN)
```

```
The accuracy of the k nearest Neighbour using Petals is: 0.977777777778
The accuracy of the k nearest Neighbour using Sepals is: 0.800000000000

Confusion Matrix for Decision Tree using petals:
 [[15  0  0]
 [ 0 12  1]
 [ 0  0 17]]

Confusion Matrix for Decision Tree using sepals:
 [[16  0  0]
 [ 0 13  3]
 [ 0  6  7]]
```



KNN

```
# K-Nearest Neighbors Classifier
model4 = KNeighborsClassifier(n_neighbors=8)
model4.fit(train_X, train_Y)
prediction4 = model4.predict(test_X) # Corrected to use model4 for prediction
accuracy_KNN = accuracy_score(test_Y, prediction4)
print(f'The accuracy of the KNN is: {accuracy_KNN:.12f}')

# Create the confusion matrix
conf_matrix_knn = confusion_matrix(test_Y, prediction4)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_knn, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix for K-Nearest Neighbors Classifier')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

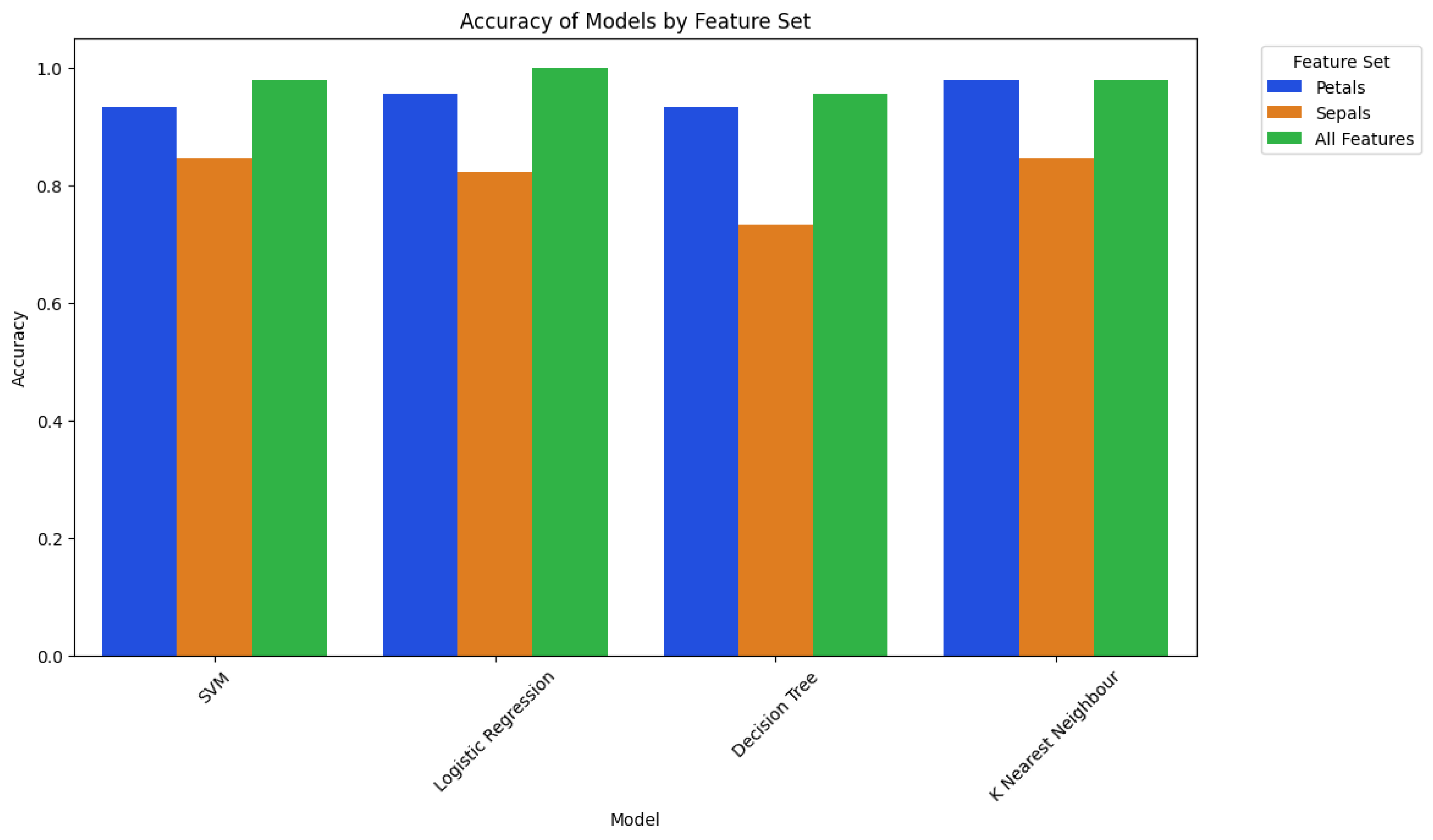
```
The accuracy of the KNN is: 0.955555555556
```

Visualised Result for Machine Learning Algorithms

```python
# Expanded accuracy data
expanded_data = {
    'Feature Set': ['Petals', 'Sepals', 'Petals', 'Sepals', 'Petals', 'Sepals', 'Petals', 'Sepals',
                    'All Features', 'All Features', 'All Features', 'All Features'],
    'Model': ['SVM', 'SVM', 'Logistic Regression', 'Logistic Regression', 'Decision Tree', 'Decision Tree',
              'K Nearest Neighbour', 'K Nearest Neighbour', 'SVM', 'Logistic Regression',
              'Decision Tree', 'K Nearest Neighbour'],
    'Accuracy': [
        accuracy_SVM_p, accuracy_SVM_s, accuracy_LR_p, accuracy_LR_s,
        accuracy_DT_p, accuracy, accuracy_KNN_p, accuracy_KNN_s,
        accuracy_svm, accuracy_LR, accuracy_DT, accuracy_KNN
    ]
}

# Convert the expanded dataset to DataFrame
expanded_accuracy_df = pd.DataFrame(expanded_data)
```

[55]

```python
plt.figure(figsize=(10, 6))
sns.barplot(y='Accuracy', x='Model', hue='Feature Set', data=expanded_accuracy_df, palette='bright')  # Using 'bright' palette
plt.title('Accuracy of Models by Feature Set')
plt.ylabel('Accuracy')
plt.xlabel('Model')
plt.ylim(0, 1.05)  # Adjusted for vertical orientation
plt.tight_layout()
plt.legend(title='Feature Set', bbox_to_anchor=(1.05, 1), loc='upper left')

plt.xticks(rotation=45)
plt.show()
```



Accuracy of Models by Feature Set
Accuracy
1.0
0.8
0.6
0.4
0.2
0.0
SVM
Logistic Regression
Decision Tree
K Nearest Neighbour
Model
Feature Set
Petals
Sepals
All Features

Discussion

Analysis of algorithm performance.

Strengths and limitations of each model.

Insights into data features and classification accuracy.

Logistic Regression

Strengths:

- **Simple and Interpretable: Easy to implement and understand.**
- **Efficient: Fast training, performs well on linear data.**

Limitations:

- **Linear Boundaries: Limited for complex datasets.**
- **Outliers: Sensitive to outliers.**
- **Binary Focus: Naturally suited for binary classification.**

K-Nearest Neighbors (KNN)

Strengths:

- **Simple and Flexible: Easy to understand, works for classification and regression.**
- **Non-parametric: No data distribution assumptions.**

Limitations:

- **Computationally Expensive: Slow for large datasets.**
- **High Dimensionality Issues: Performance degrades with more features.**
- **Tuning Needed: Requires careful selection of k.**

Decision Tree (DT)

Strengths:

- Intuitive: Easy to interpret and visualize.
- Non-parametric: No distribution assumptions, handles complex boundaries.

Limitations:

- Overfitting: Prone to overfitting with deep trees.
- Instability: Sensitive to data changes.
- Needs Pruning: Requires pruning or ensembles to reduce overfitting.

Support Vector Machine (SVM)

Strengths:

- High-dimensional Effectiveness: Works well with many features.
- Kernel Trick: Handles non-linear relationships.

Limitations:

- Computationally Intensive: Slow training for large datasets.
- Parameter Tuning: Performance depends on kernel and parameter choices.
- Less Interpretable: Difficult to interpret compared to simpler models.

Summary

1. **Logistic Regression:** Simple, interpretable, but struggles with non-linear data and outliers.

2. **K-Nearest Neighbors:** Flexible, simple, but computationally expensive, sensitive to dimensions.

3. **Decision Tree:** Intuitive, handles complexity, but prone to overfitting and instability.

4. **Support Vector Machine:** Excels with high-dimensional, non-linear data, but needs careful tuning and is computationally intensive.

Deep Learning

With Iris Image Dataset

Importing the Image Dataset
&
shorting the data set to be 50 images per class:

```
[ ] import os
    from PIL import Image
    import matplotlib.image as img
```

```
for dirname, _, filenames in os.walk('/content/drive/MyDrive/Iris_picture'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
[ ] import pathlib
    pic_path = '/content/drive/MyDrive/Iris_picture' # Datasets path
    pic_path = pathlib.Path(pic_path)
    pic_path
```

```
PosixPath('/content/drive/MyDrive/Iris_picture')
```

```
[ ] setosa = list(pic_path.glob('iris-setosa/*'))
    setosa = setosa[: 50] # shorted dataset

    versicolour = list(pic_path.glob('iris-versicolour/*'))
    versicolour = versicolour[: 50] # shorted dataset

    virginica = list(pic_path.glob('iris-virginica/*'))
    virginica = virginica[: 50] # shorted dataset


    print("Length of setosa: ", len(setosa))
    print("Length of versicolour: ", len(versicolour))
    print("Length of virginica: ", len(virginica))
```

```
Length of setosa:  50
Length of versicolour:  50
Length of virginica:  50
```

EDA: Checking out a random image of iris for each class/species:

In [39]:

```
from matplotlib.image import imread

fig, ax = plt.subplots(ncols=3, nrows=1, figsize=(15, 3))
fig.suptitle('Category')

def display_random_image(category, axis, title):
    if category:
        random_index = np.random.randint(0, len(category))
        image_path = category[random_index]
        try:
            image = imread(image_path)  # Use imread directly
            axis.imshow(image)
        except FileNotFoundError:
            axis.imshow(np.zeros((10, 10, 3)))  # Display an empty image if the file is not found
            title += " (Missing)"
    else:
        axis.imshow(np.zeros((10, 10, 3)))  # Display an empty image if the category is empty
        title += " (Empty)"

    axis.set_title(title)
    axis.axis('off')  # Hide axes ticks

# Display images
display_random_image(setosa, ax[0], 'Setosa')
display_random_image(versicolour, ax[1], 'Versicolour')
display_random_image(virginica, ax[2], 'Virginica')
plt.show()
```

Defining the Dataframe, reading and resizing images:

```
[ ] #Defining Dataframe here

    #images path
    df_images = {
        'setosa' : setosa,
        'versicolour' : versicolour,
        'virginica': virginica
    }

    #numerical labels for the categories
    df_labels = {
        'setosa' : 0,
        'versicolour' : 1,
        'virginica': 2
    }
```

```
[ ] import cv2
    rand_virg = np.random.randint(-1, len(virginica))
    img = cv2.imread(str(df_images['virginica'][rand_virg])) # Converting it into numerical arrays
    img.shape

    (256, 256, 3)
```

```
[ ] X, y = [], [] # X = images, y = labels
    for label, images in df_images.items():
        for image in images:
            img = cv2.imread(str(image))
            resized_img = cv2.resize(img, (224, 224)) # Resizing the images to be able to pass on MobileNetv2 model
            X.append(resized_img)
            y.append(df_labels[label])
    print(len(X), len(y))

    #Runtime - 1m39s

    150 150
```

Resize all the images to fit into Input Layer 224,224,3

```python
X_train, X_test_val, y_train, y_test_val = train_test_split(X, y, train_size = 0.8, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_test_val, y_test_val, random_state=42)
X_train.shape, X_val.shape, X_test.shape, y_train.shape, y_val.shape, y_test.shape
```

```
((336, 224, 224, 3),
 (22, 224, 224, 3),
 (63, 224, 224, 3),
 (336,),
 (22,),
 (63,))
```

The Model 1 Architecture (4 layers)

Rescaling Layer:

This Rescaling layer normalizes the pixel values of images to the range [0, 1]. This is a common practice as it helps the neural network train faster and more effectively.

```
[ ]  normalizer = tf.keras.layers.Rescaling(scale=1/255)
```

MobileNetV2 Layer:

The model uses a URL to load a pre-trained MobileNetV2 model from TensorFlow Hub, which is designed to work with TensorFlow 2.0. hub.KerasLayer wraps the MobileNetV2 model as a Keras layer and sets trainable=False to freeze the weights of the model during training, meaning the backpropagation won't alter them. This is useful when using the model as a feature extractor.

```
[ ] mobile_net = 'https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4'
    mobile_net = hub.KerasLayer(
        mobile_net, input_shape=(224,224, 3), trainable=False)
```

The model is built using the Sequential API, which allows you to stack layers in a linear fashion.

Starts with an input layer specifying the input shape as 224x224 pixels with 3 channels (RGB).

Followed by the rescaling layer.

MobileNetV2 is then added as the feature extractor.

Dropout Layer
&
Dense Layer with
Softmax Activation and L2 regularization:

A Dropout layer is included to reduce overfitting by randomly setting a fraction (20% here) of input units to 0 at each update during training.

The final layer is a Dense layer with a softmax activation function, used to classify the images into num_label classes.

```
[ ] from tensorflow.keras.regularizers import l2
    model = keras.Sequential([
        keras.Input(shape=(224,224,3)),
        normalizer,
        mobile_net,
        Dropout(0.25),  # Adding dropout; adjust the rate as needed
        Dense(num_label, activation='softmax', kernel_regularizer=l2(0.01))  # Adding L2 regularization
    ])
```

Model 1 Compilation

Compiles the model with the Adam optimizer and categorical crossentropy as the loss function, suitable for multi-class classification problems. Metrics are set to 'accuracy' to monitor the classification accuracy.

```
[ ]  model.compile(optimizer='adam',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])
```

Callbacks Used in Model 1:

1 - Learning Rate Scheduler:

```
import tensorflow as tf

learning_rate = tf.Variable(0.001, trainable=False)
learning_rate.assign(0.0001)
```

```
<tf.Variable 'UnreadVariable' shape=() dtype=float32, numpy=1e-04>
```

```
from tensorflow.keras.callbacks import LearningRateScheduler

def scheduler(epoch, lr):
    if epoch < 5:
        return lr  # Keep the initial learning rate for the first 5 epochs
    else:
        return lr * tf.math.exp(-0.1)  # Decrease the learning rate exponentially after the 5th epoch

lr_scheduler = LearningRateScheduler(scheduler)
```

2 - Model Checkpoint:

Saves the best model based on the validation loss.

```
mc = ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min', save_best_only=True)
```

Model 1 Prediction:

```
history = model.fit(X_train, y_train_cat, batch_size=12, epochs=30, validation_data=(X_val, y_val_cat), callbacks=[ mc, lr_scheduler])
```

```
[ ] prediction = model.predict(X_test, batch_size=64, verbose=1)
    prediction = np.argmax(model.predict(X_test), axis=-1) #for multiclass
    prediction = prediction.flatten()
    prediction
```

```
1/1 [==============================] - 1s 583ms/step
1/1 [==============================] - 0s 130ms/step
array([2, 1, 1, 2, 1, 0, 0, 0, 0, 2, 2, 0, 2, 2, 1, 0, 0, 2, 0, 0, 1, 0])
```

```
[ ] accuracy = accuracy_score(y_test, prediction)
    print(accuracy)
```

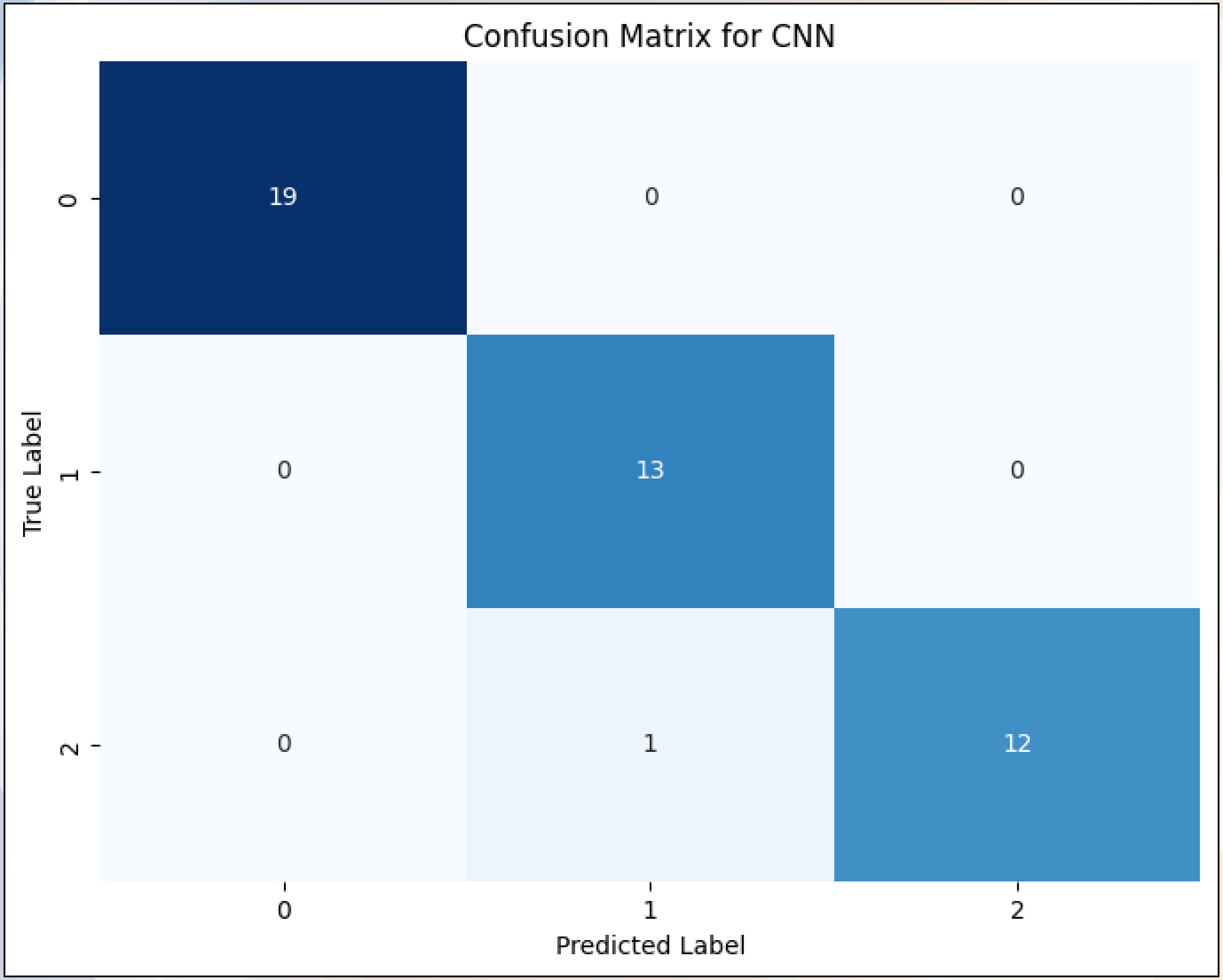
```
0.7272727272727273
```

```
[ ] model.evaluate(X_test,y_test_cat)
```

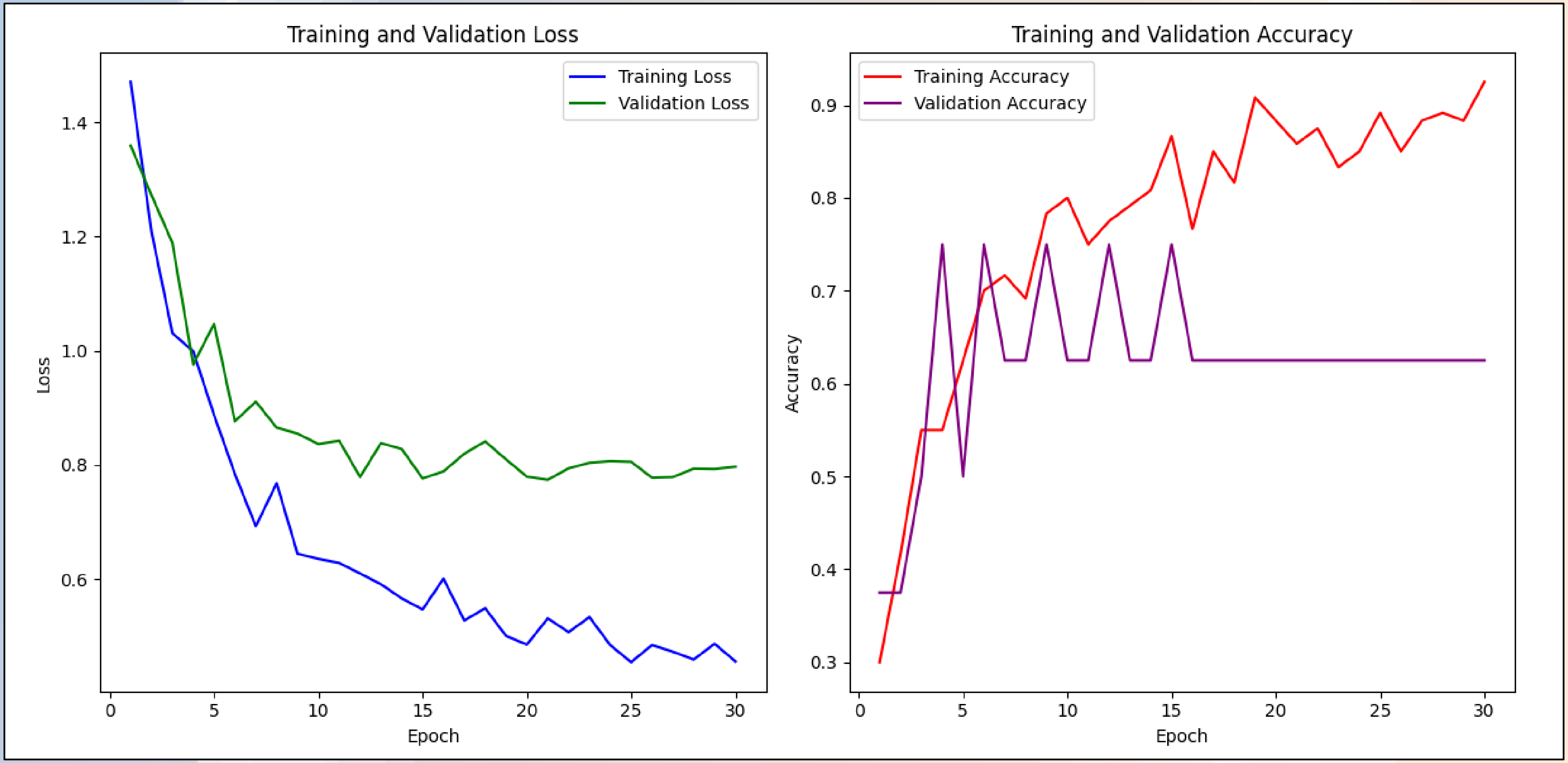
```
1/1 [==============================] - 0s 52ms/step - loss: 0.7861 - accuracy: 0.7273
[0.7861070036888123, 0.7272727489471436]
```

Accuracy = 72.7%

Model 1 - Visualized Result 1



Model 1 - Visualized Result 2



```
[ ] model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

Monitors the validation loss and will stop the training if it does not improve for 25 e

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 224, 224, 3)       0

 keras_layer (KerasLayer)    (None, 1280)              2257984

 dropout_1 (Dropout)         (None, 1280)              0

 dense_1 (Dense)             (None, 3)                 3843

=================================================================
Total params: 2261827 (8.63 MB)
Trainable params: 3843 (15.01 KB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

```
from tensorflow.keras.regularizers import l2
model = keras.Sequential([
    keras.Input(shape=(224,224,3)),
    normalizer,
    mobile_net,
    Dropout(0.5),  # Adding dropout; adjust the rate as needed
    Dense(num_label, activation='softmax', kernel_regularizer=l2(0.001))  # Adding L2 regularization
])
```

Model 2

Layers:

1. **Rescaling Normalizer Layer**
2. **MobileNet Feature Extractor Layer**
3. **Dropout Layer with 50% rate**
4. **Dense layer with Softmax activation func and L2 regulizer with 0.001 learning rate.**

Balanced the dataset to 60 each.

Model 2

```
[ ] early_stopping = EarlyStopping(monitor='val_loss', mode='min', patience=25, verbose=1)
```

Saves the best model based on the validation loss.

```
[ ] mc = ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min', save_best_only=True)
```

Trains the model for a maximum of 100 epochs on the training data.

Uses the validation data to evaluate the performance after each epoch.

Employs callbacks for early stopping and model checkpointing to enhance training effectiveness and efficiency.

```
[ ] history = model.fit(X_train, y_train_cat, epochs=100, validation_data=(X_val, y_val_cat), callbacks=[early_stopping, mc])
```

Model 2 result

```
[58] prediction = model.predict(X_test, batch_size=64, verbose=1)
     prediction = np.argmax(model.predict(X_test), axis=-1) #for multiclass
     prediction = prediction.flatten()
     prediction
```

```
1/1 [==============================] - 3s 3s/step
2/2 [==============================] - 1s 1s/step
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1])
```

```
accuracy = accuracy_score(y_test, prediction)
print(accuracy)
```

```
0.6190476190476191
```

```
[60] model.evaluate(X_test,y_test_cat)
```

```
2/2 [==============================] - 0s 47ms/step - loss: 1.0455 - accuracy: 0.6190
[1.0454580783843994, 0.6190476417541504]
```

Accuracy = 62%

Summary

Data Distribution:

Both models now use a balanced dataset, though Model 2 has slightly more images per class (60 vs. 50).

Architecture:

Both models have identical architectures, except for the dropout rate. Model 1 uses a dropout rate of 0.25, whereas Model 2 uses a dropout rate of 0.5.

Training:

Model 1 uses a more sophisticated learning rate scheduler and trains for fewer epochs (30) with a batch size of 12.
Model 2 trains for a longer period (100 epochs) with early stopping and default learning rate settings.

Summary

Even with balanced data, Model 1 outperforms Model 2 in terms of accuracy and loss on the test set. The lower dropout rate and more sophisticated learning rate scheduling in Model 1 likely contributed to its better performance. The higher dropout rate in Model 2 might have led to underfitting, especially given the longer training duration and early stopping that might have been triggered before the model could converge to a better solution. The differences in training approaches highlight the importance of carefully tuning hyperparameters and monitoring the training process to achieve optimal performance.

Conclusion

Project Exploratory Data Analysis (EDA) was valuable experience.

Enhanced understanding of ML and DL algorithms.

Fun and easy project :P

Questions & Answers

- Please feel free to ask any questions or clarify doubts regarding the project.