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

C

Introduction

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

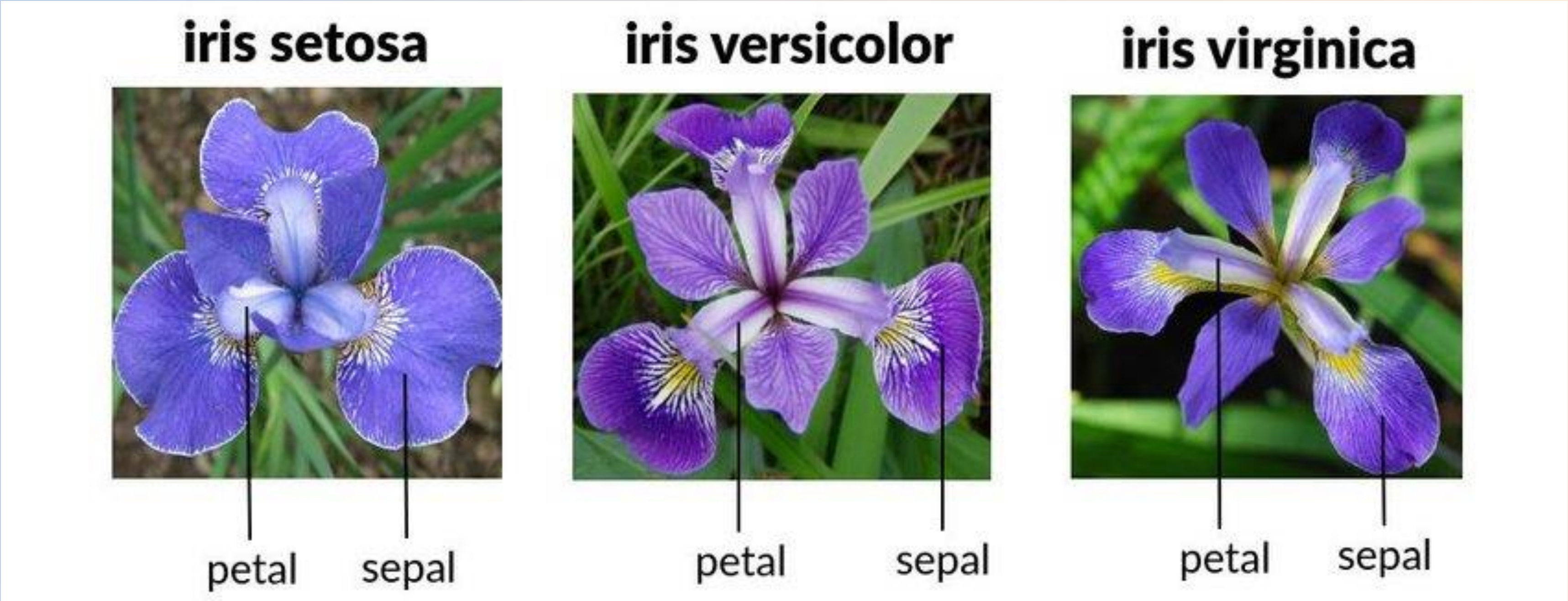
Datasets Overview: 1. Consists of measurements from three species of Iris (setosa, versicolor, virginica). 2. Consists of images of three different species of iris.

Machine Learning

With Iris CSV Dataset

B

Classification Analysis and ML prediction on Iris Dataset



B

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
```

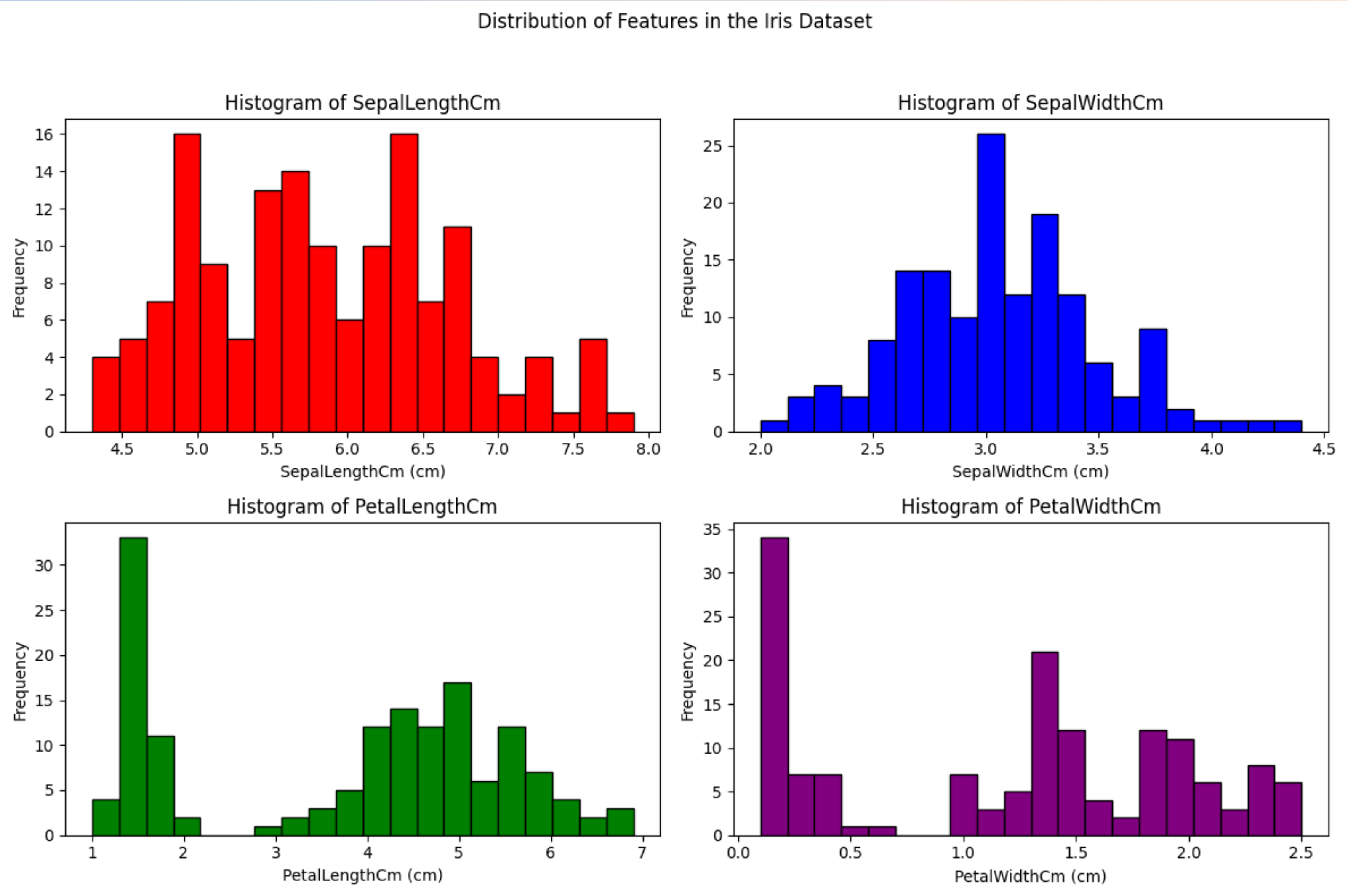
```
[6] iris.shape
    # 150 records and 5 columns

(150, 5)
```

EDA

With Iris CSV Dataset

Distribution of Features in the Iris Dataset

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

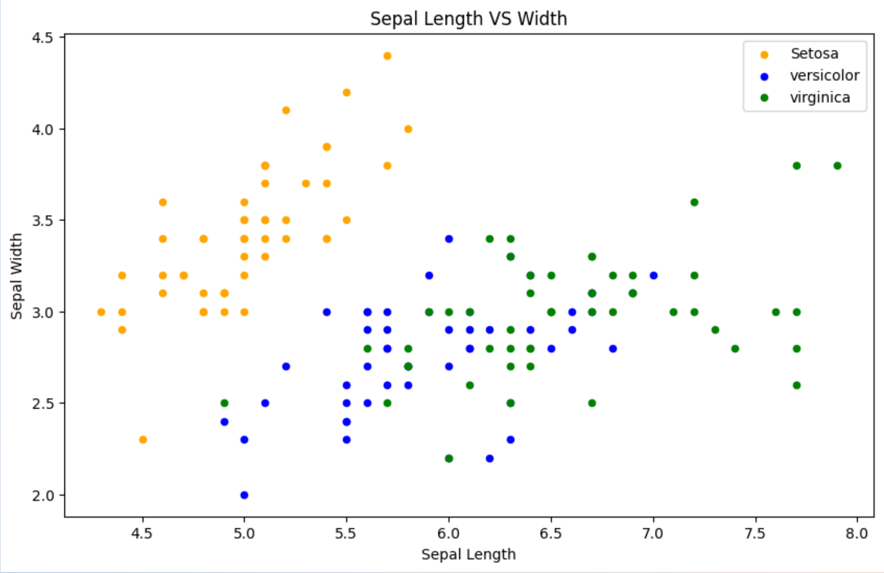


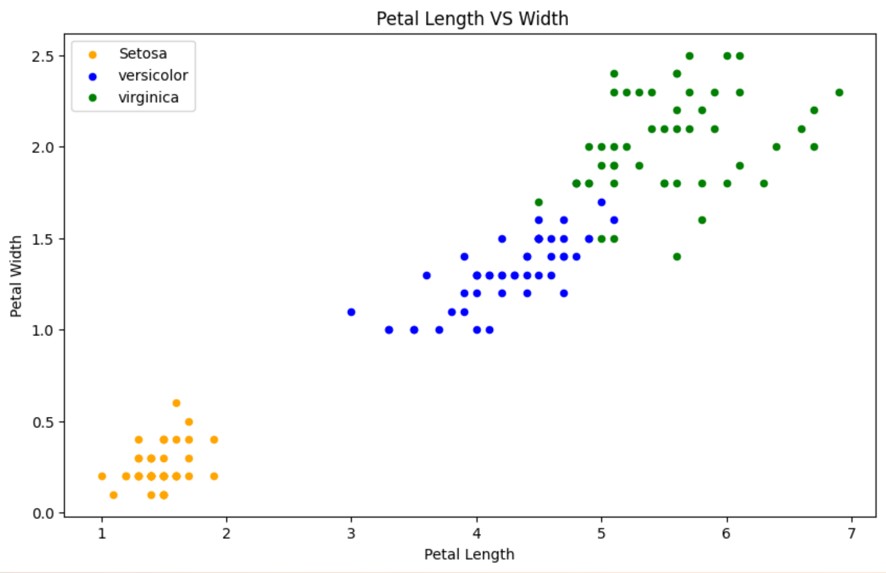
Code block for Histogram

```
# 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])
```

M

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





M

Code blocks for Scatter Plot

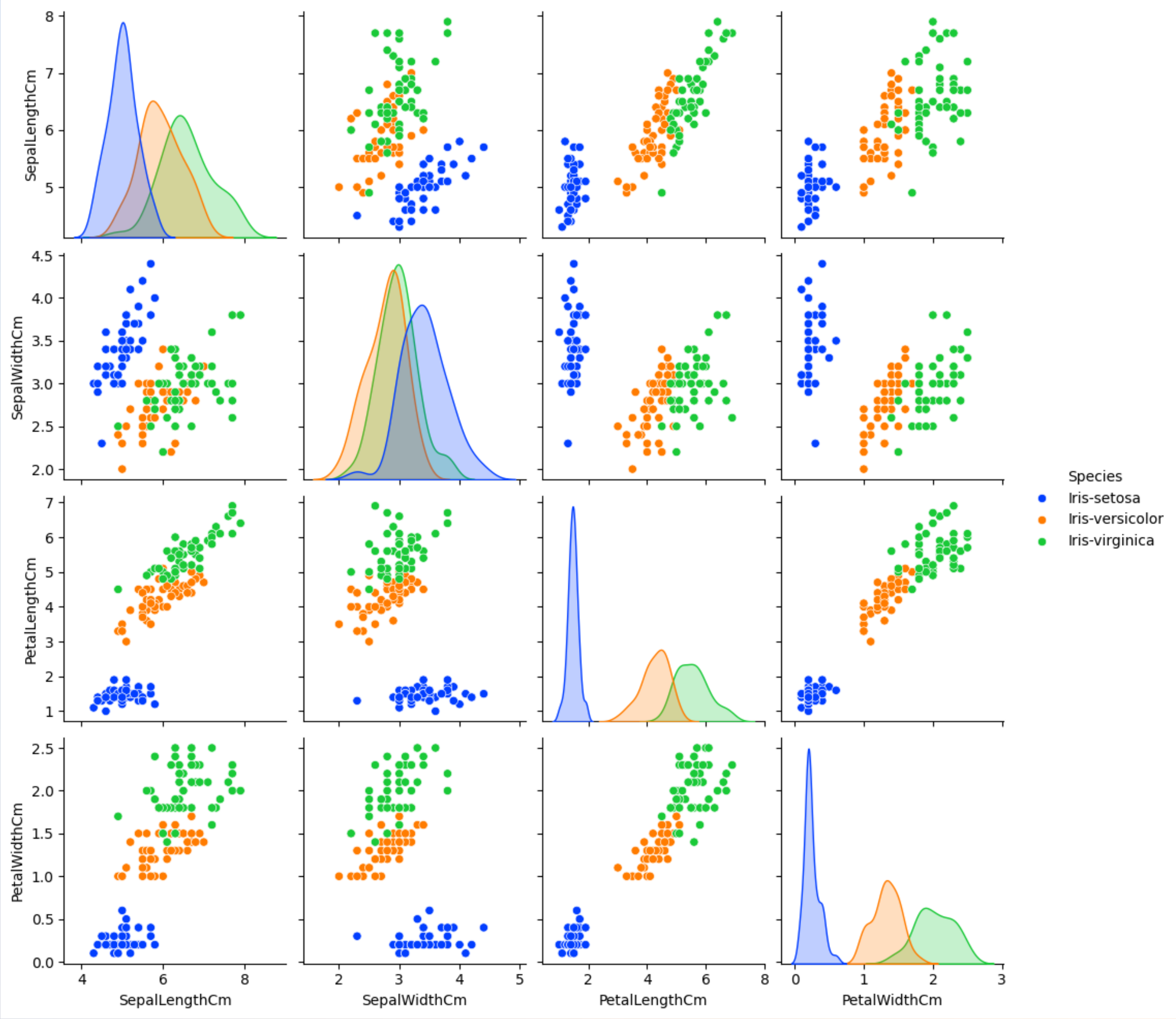
```
fig = iris[iris.Species=='Iris-setosa'].plot(kind='scatter',x='SepalLengthCm',y='SepalWidthCm',color='orange', label='Setosa')
iris[iris.Species=='Iris-versicolor'].plot(kind='scatter',x='SepalLengthCm',y='SepalWidthCm',color='blue', label='versicolor',ax=fig)
iris[iris.Species=='Iris-virginica'].plot(kind='scatter',x='SepalLengthCm',y='SepalWidthCm',color='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()
```

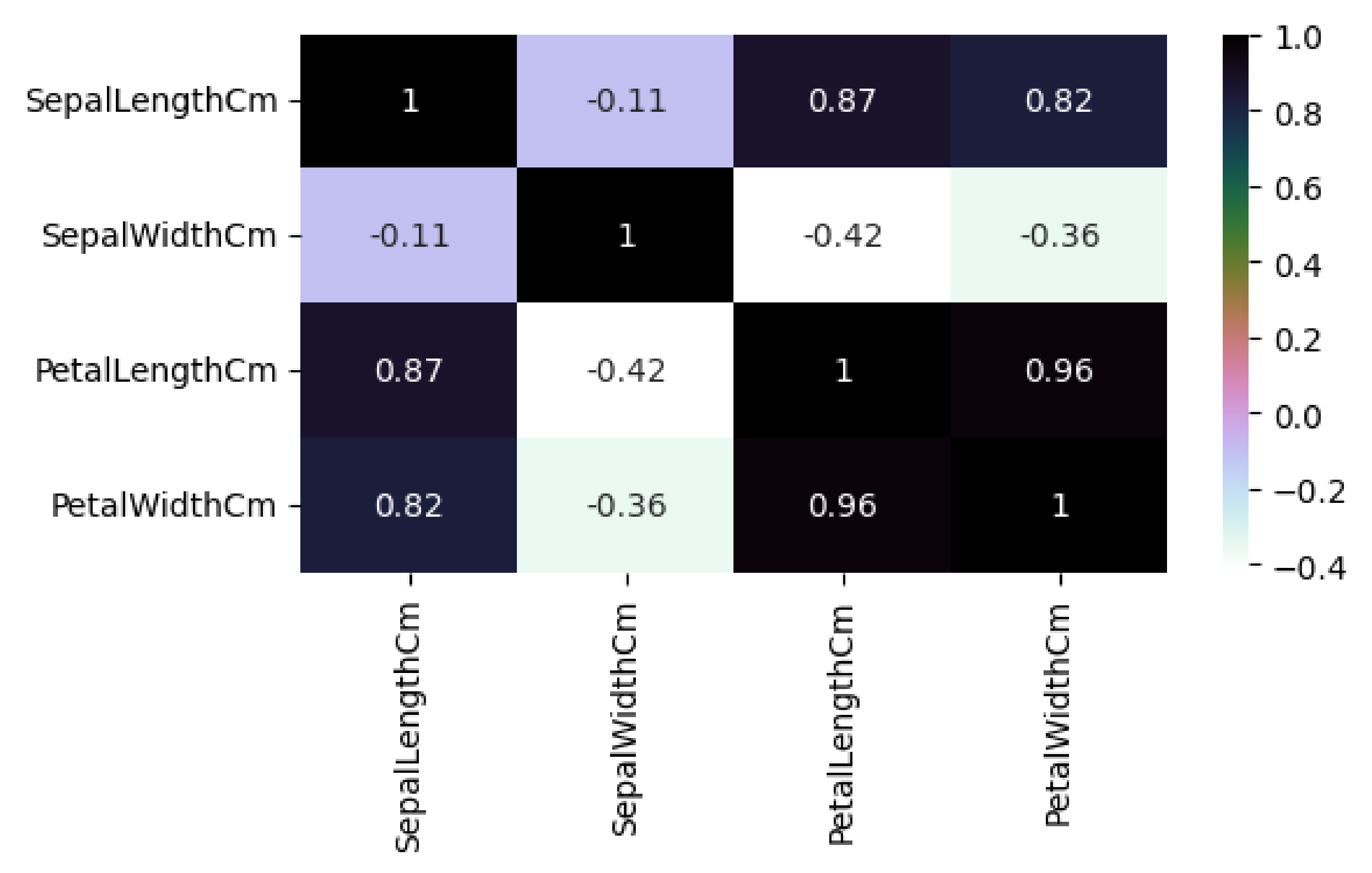
M

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

Iris Dataset Pairplot



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()
```

M

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).
```

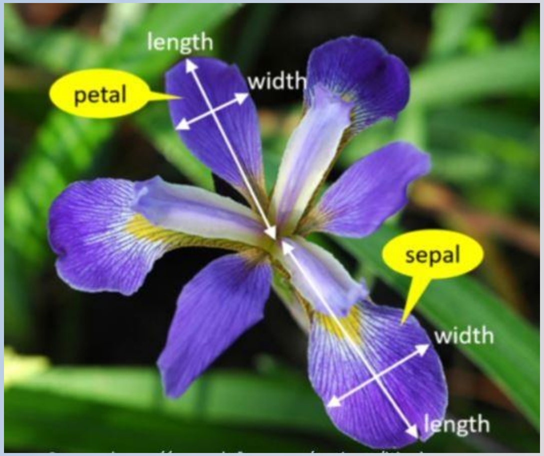
Importing the Iris Dataset

```
[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

E

Methodology



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

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
```

E

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,))
```

B

Dividing dataset for comparison

```
[ ] # Dividing the dataframe into two
    petal = iris[['PetalLengthCm','PetalWidthCm','Species']]
    sepal = iris[['SepalLengthCm','SepalWidthCm','Species']]
```

```
[ ]
    #Petal
    train_petal,test_petal = train_test_split(petal,test_size=0.3,random_state=None)
    train_x_petal = train_petal[['PetalWidthCm','PetalLengthCm']]
    train_y_petal = train_petal.Species
    test_x_petal = test_petal[['PetalWidthCm','PetalLengthCm']]
    test_y_petal = test_petal.Species

    #Sepal
    train_sepal,test_sepal = train_test_split(sepal,test_size=0.3,random_state=None)
    train_x_sepal = train_sepal[['SepalWidthCm','SepalLengthCm']]
    train_y_sepal = train_sepal.Species
    test_x_sepal = test_sepal[['SepalWidthCm','SepalLengthCm']]
    test_y_sepal = test_sepal.Species
```

B

Support Vector Classifier (SVC)

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

time_SVC_petal = end_time - start_time
print(f"Elapsed time: {time_SVC_petal} seconds")

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

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)
```

Sepals and Petals

```python
# Initialize and train the SVC model
model = svm.SVC()
model.fit(train_X,train_Y)
start_time = time.time()
prediction1 = model.predict(test_X)
end_time = time.time()
accuracy_svm = accuracy_score(test_Y, prediction1)
print(f'The accuracy of the SVC 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 SVC Model')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

time_SVC = end_time - start_time
print(f"Elapsed time: {time_SVC} seconds")
```

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

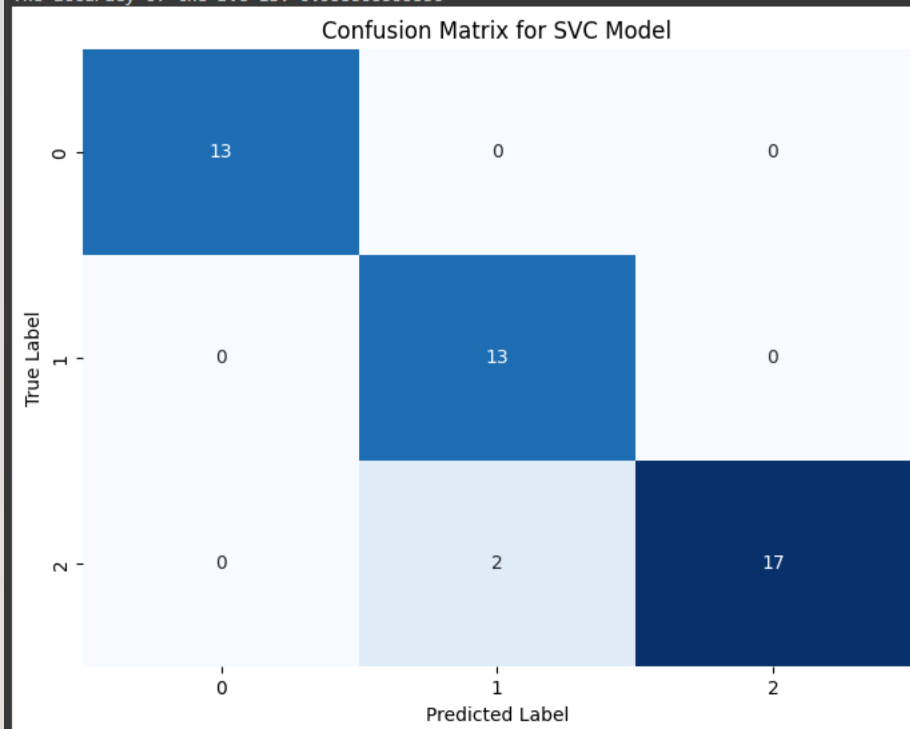
All Features

B

Support Vector Classifier (SVC)

```
The accuracy of the SVM using Petals is: 0.911111111111
Elapsed time: 0.0026574134826660156 seconds
The accuracy of the SVM using Sepals is: 0.844444444444
Elapsed time: 0.0017066001892089844 seconds

Confusion Matrix for SVC using petals:
 [[13  0  0]
 [ 0 16  3]
 [ 0  1 12]]

Confusion Matrix for SVC using sepals:
 [[13  0  0]
 [ 0 13  5]
 [ 0  2 12]]
```

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




```
Elapsed time: 0.0015594959259033203 seconds
```

Sepals and Petals

All features

B

Decision Tree

```python
# Decision Tree
model= DecisionTreeClassifier()
model.fit(train_x_petal,train_y_petal)
start_time = time.time()
prediction5 =model.predict(test_x_petal)
end_time = time.time()
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}')
time_DT_p = end_time - start_time
print(f"Elapsed time: {time_DT_p} seconds")

model= DecisionTreeClassifier()
model.fit(train_x_sepal,train_y_sepal)
start_time = time.time()
prediction6 =model.predict(test_x_sepal)
end_time = time.time()
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}')
time_DT_s = end_time - start_time
print(f"Elapsed time: {time_DT_s} seconds")

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_petalDT)
print('\nConfusion Matrix for Decision Tree using sepals: \n', conf_matrix_sepalDT)
```

```python
# Decision Tree Classifier
model3 = DecisionTreeClassifier()
model3.fit(train_X, train_Y)
start_time = time.time()
prediction3 = model3.predict(test_X) # Corrected to use model3 for prediction
end_time = time.time()
accuracy_DT = accuracy_score(test_Y, prediction3)
print(f'The accuracy of the Decision Tree is: {accuracy_DT:.12f}')
time_DT = end_time - start_time
print(f"Elapsed time: {time_DT} seconds")

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()
```

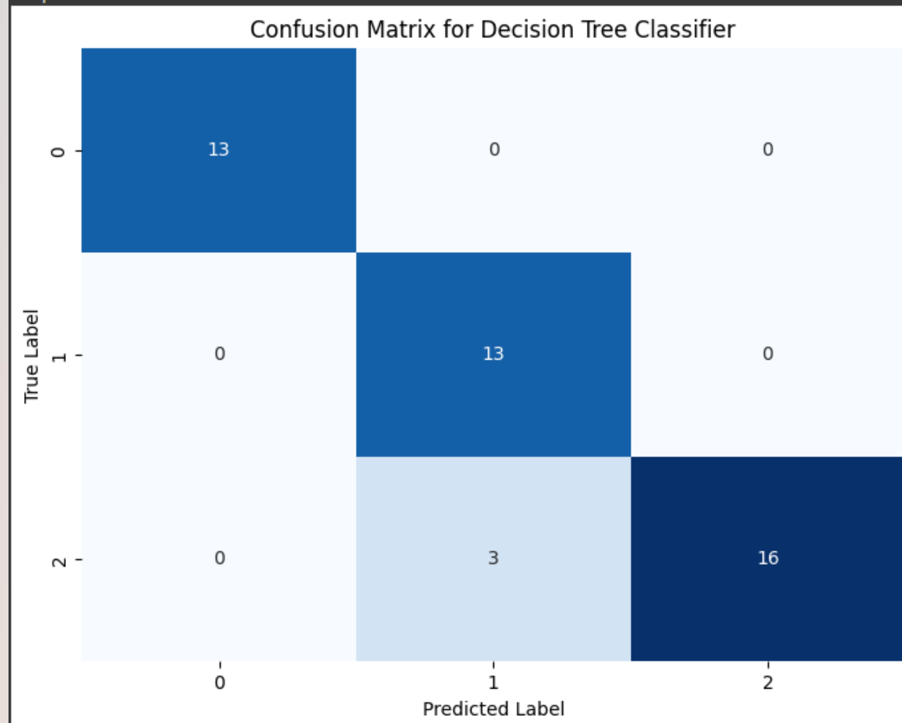
Sepals and Petals

All Features

B

Decision Tree

```
The accuracy of the Decision Tree is: 0.933333333333
Elapsed time: 0.002412080764770508 seconds
```




```
The accuracy of the Decision Tree using Petals is: 0.911111111111
Elapsed time: 0.002856016159057617 seconds
The accuracy of the Decision Tree using Sepals is: 0.666666666667
Elapsed time: 0.0014157295227050781 seconds

Confusion Matrix for Decision Tree using petals:
 [[13  0  0]
 [ 0 16  3]
 [ 0  1 12]]

Confusion Matrix for Decision Tree using sepals:
 [[12  1  0]
 [ 0  9  9]
 [ 0  5  9]]
```

Sepals and Petals

All features

B

Logistic Regression

```
#Logistic Regression
model= LogisticRegression()
model.fit(train_x_petal,train_y_petal)
start_time = time.time()
prediction3 = model.predict(test_x_petal)
end_time = time.time()
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}')
time_LR_p = end_time - start_time
print(f"Elapsed time: {time_LR_p} seconds")

model= LogisticRegression()
model.fit(train_x_sepal,train_y_sepal)
start_time = time.time()
prediction4 = model.predict(test_x_sepal)
end_time = time.time()
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}')
time_LR_s = end_time - start_time
print(f"Elapsed time: {time_LR_s} seconds")

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)
```

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

Sepals and Petals | **All Features**

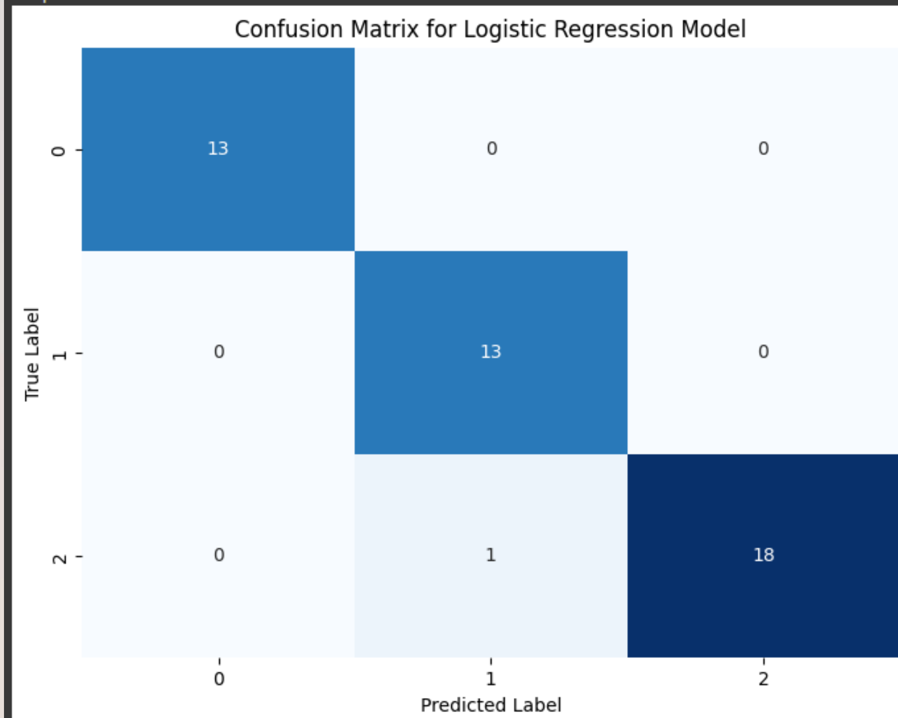
C

Logistic Regression

```
The accuracy of the Logistic Regression using Petals is: 0.911111111111
Elapsed time: 0.0018253326416015625 seconds
The accuracy of the Logistic Regression using Sepals is: 0.866666666667
Elapsed time: 0.0015850067138671875 seconds

Confusion Matrix for Logistic Regression using petals:
 [[13  0  0]
 [ 0 16  3]
 [ 0  1 12]]

Confusion Matrix for Logistic Regression using sepals:
 [[13  0  0]
 [ 0 13  5]
 [ 0  1 13]]
```

Sepals and Petals

```
The accuracy of the Logistic Regression is: 0.977777777778
Elapsed time: 0.0018439292907714844 seconds
```



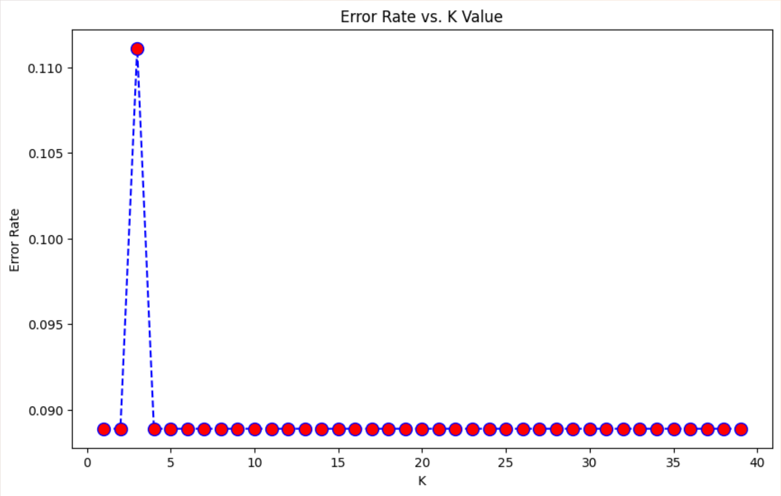
All features

C

KNN

```
[ ]  error_rate = []

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

         knn = KNeighborsClassifier(n_neighbors=i)
         knn.fit(train_x_petal,train_y_petal)
         pred_i = knn.predict(test_x_petal)
         error_rate.append(np.mean(pred_i != test_y_petal))
```

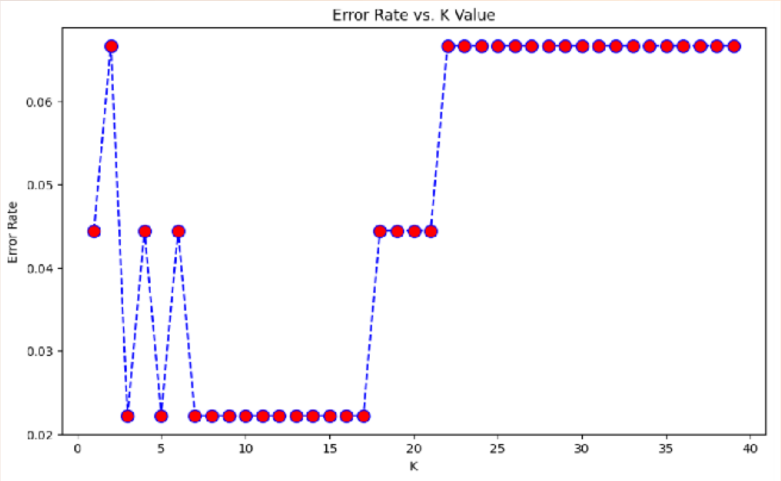
```
[ ]  plt.figure(figsize=(10,6))
     plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
              markerfacecolor='red', markersize=10)
     plt.title('Error Rate vs. K Value')
     plt.xlabel('K')
     plt.ylabel('Error Rate')
```



Finding Right K value

```
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))
```

```
[ ]  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')
```



C

KNN

```
# k nearest Neighbour
model= KNeighborsClassifier(n_neighbors=8)
model.fit(train_x_petal,train_y_petal)
start_time = time.time()
prediction7=model.predict(test_x_petal)
end_time = time.time()
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}')
time_KNN_p = end_time - start_time
print(f"Elapsed time: {time_KNN_p} seconds")


model= KNeighborsClassifier(n_neighbors=8)
model.fit(train_x_sepal,train_y_sepal)
start_time = time.time()
prediction8=model.predict(test_x_sepal)
end_time = time.time()
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}')
time_KNN_s = end_time - start_time
print(f"Elapsed time: {time_KNN_s} seconds")

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)
```

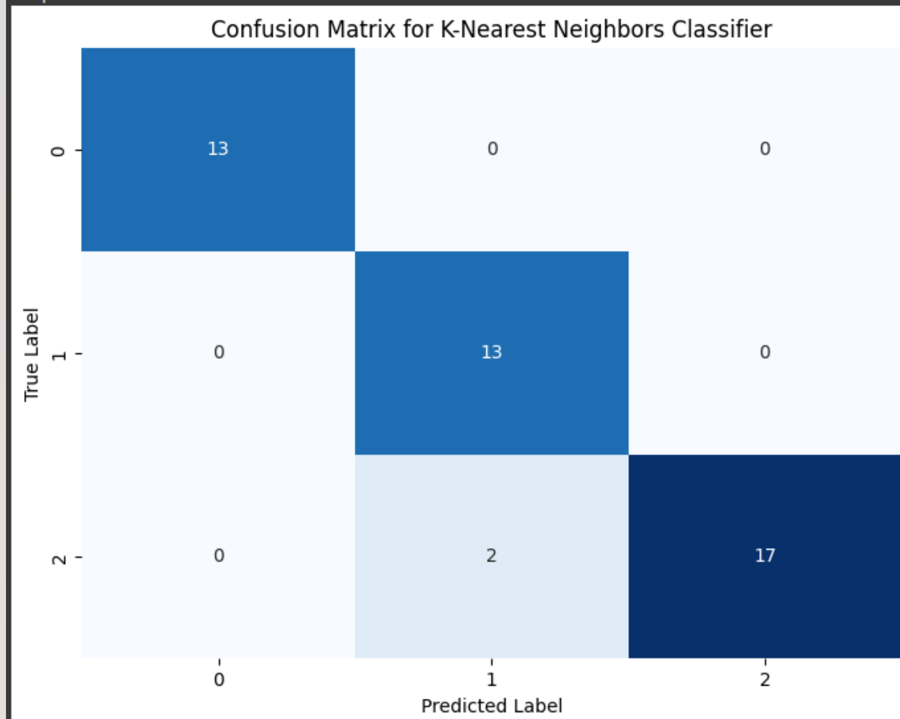
Sepals and Petals

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

# 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()
```

All Features

KNN

```
The accuracy of the KNN is: 0.955555555556
Elapsed time: 0.005745887756347656 seconds
```



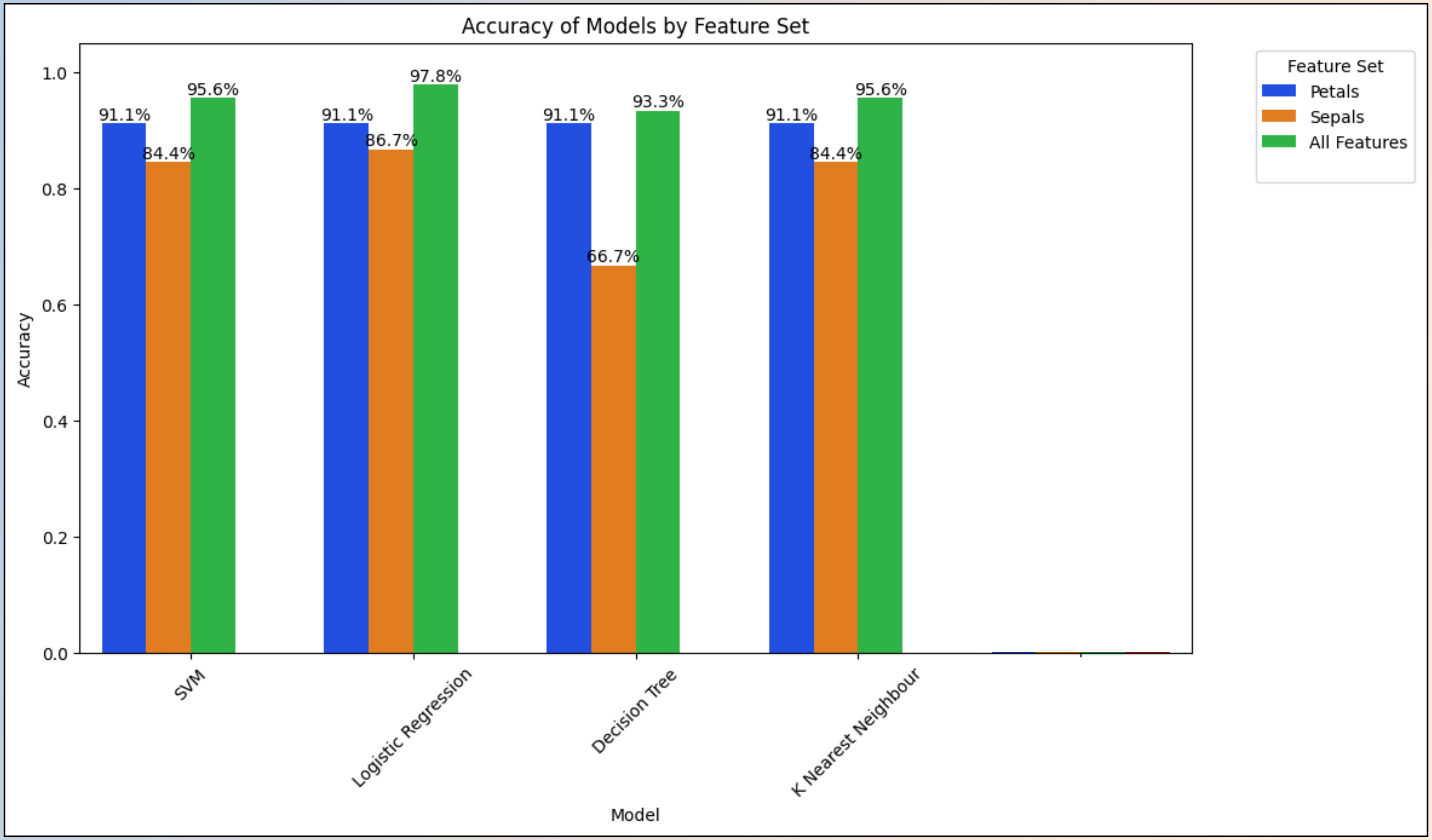

```
The accuracy of the k nearest Neighbour using Petals is: 0.911111111111
Elapsed time: 0.007362842559814453 seconds
The accuracy of the k nearest Neighbour using Sepals is: 0.844444444444
Elapsed time: 0.005990743637084961 seconds

Confusion Matrix for k nearest Neighbour using petals:
 [[13  0  0]
 [ 0 16  3]
 [ 0  1 12]]

Confusion Matrix for k nearest Neighbour sepals:
 [[13  0  0]
 [ 0 13  5]
 [ 0  2 12]]
```

Sepals and Petals

All features

C

Visualised Result for Machine Learning Algorithms



B

Summary

- SVM and Logistic Regression were the best ones in predicting.
- Using only Sepal is difficult for prediction.
- Choosing good K value in KNN.(8 in our case)
- Decision tree perhaps performed better with different hyperparameter.

Deep Learning - 1

(With Iris CSV Dataset)

E

Callbacks Used in Models:

1 – Model Checkpoint

```
In [164…  mc = ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min', save_best_only=True)

In [165…  early_stopping = EarlyStopping(monitor='val_loss', mode='min', patience=25, verbose=1)
```

Saves the best model based on the validation loss.

2 – Early Stopping

E

The Model 1 (All Features) Architecture

```
In [166…
#Creating a model for csv dataset
model_both = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

Model Compilation

```
In [167…
model_both.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
                   optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
                   metrics=['accuracy'])
```

Model 1 (All Features) Prediction:

```
# Training the model
history = model_both.fit(X_train, y_train,
                         epochs=50,
                         validation_data=(val_X, val_Y),
                         callbacks=[mc, early_stopping],
                         verbose=1)
```

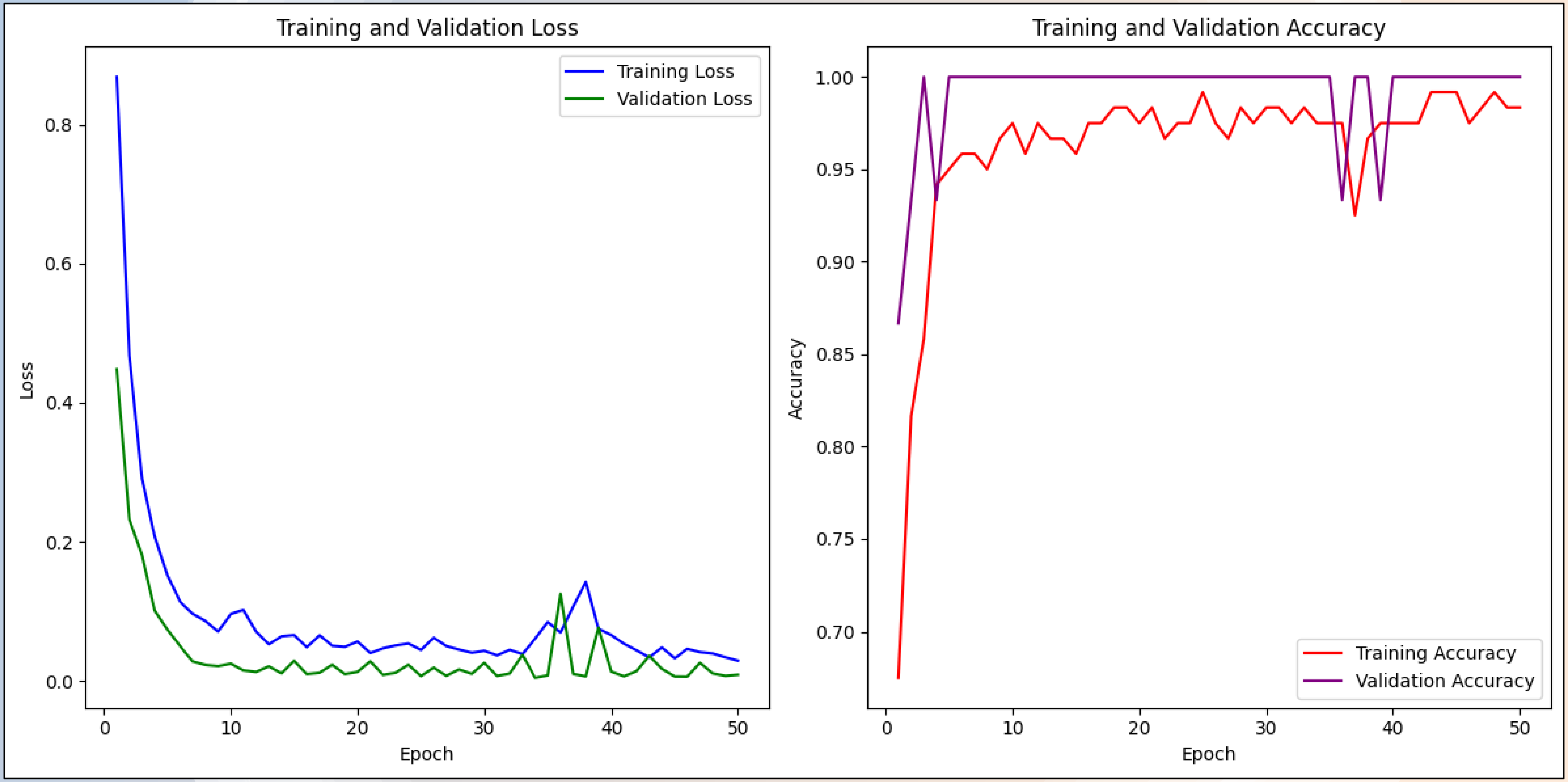
```
[ ] start_time = time.time()
    predictions = model_both.predict(test_X, verbose=1)
    predicted_classes = np.argmax(predictions, axis=-1)

    # Decoding the test set labels for accuracy calculation
    true_classes = np.argmax(test_Y, axis=-1)
    end_time = time.time()
    # Calculating and printing the accuracy score
    accuracy_CNN = accuracy_score(true_classes, predicted_classes)
    print(f'Accuracy: {accuracy_CNN:.4f}')

    time_CNN_CSV_both = end_time - start_time
    print(f"Elapsed time: {time_CNN_CSV_both} seconds")
```

```
WARNING:tensorflow:6 out of the last 8 calls to <function Model.make_predict_function
1/1 [==============================] - 0s 97ms/step
Accuracy: 1.0000
Elapsed time: 0.19290971755981445 seconds
```

Accuracy = 100%

E

Model 1 (All Features) - Visualized Result



E

The Model 2 (Petals only) Architecture

In [179…

```
model_2 = tf.keras.Sequential([
          tf.keras.layers.Dense(64, activation='relu', input_shape=(train_x_petal.shape[1],)),
          tf.keras.layers.Dense(32, activation='relu'),
          tf.keras.layers.Dense(16, activation='relu'),
          tf.keras.layers.Dense(3, activation='softmax')
])
```

Model Compilation

In [180…

```
model_2.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                metrics=['accuracy'])
```

Model 2 (Petals only) Prediction:

```
history_petal = model_2.fit(train_x_petal, train_y_petal_categorical,
                      validation_data=(val_x_petal, val_y_petal_categorical),
                      epochs=40,
                      batch_size=16,
                      callbacks=[mc, early_stopping])
```

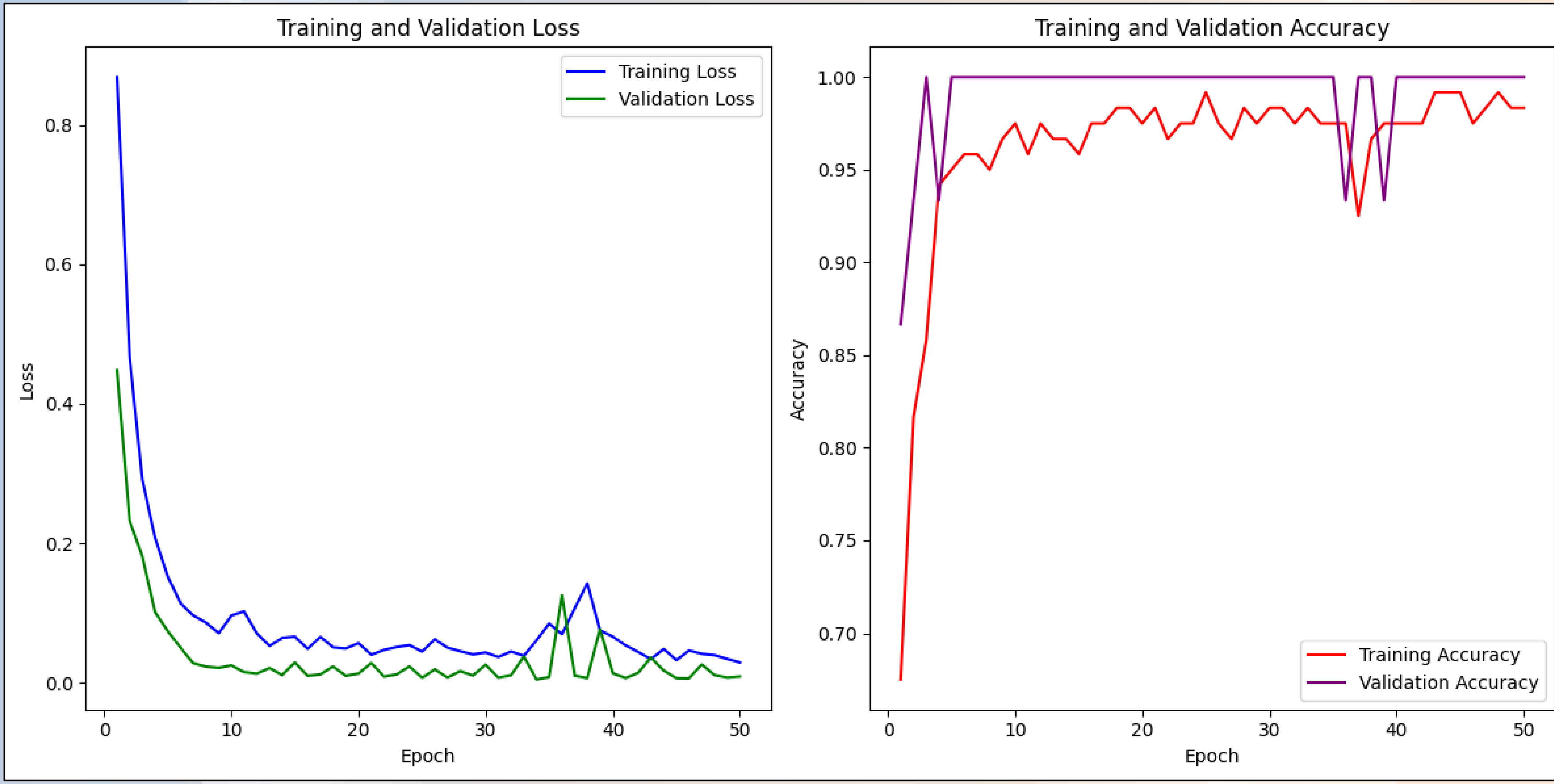
```
[ ] start_time = time.time()
    predictions = model_2.predict(test_x_petal, verbose=1)
    predicted_classes = np.argmax(predictions, axis=-1)
    predicted_classes = predicted_classes.flatten()
    end_time = time.time()

    decoded_predictions = label_encoder.inverse_transform(predicted_classes)
    test_y_petal_decoded = label_encoder.inverse_transform(np.argmax(test_y_petal_categorical, axis=-1))

    accuracy_CNN_petal = accuracy_score(test_y_petal_decoded, decoded_predictions)
    print(f'Accuracy: {accuracy_CNN_petal:.4f}')

    time_CNN_CSV_p = end_time - start_time
    print(f"Elapsed time: {time_CNN_CSV_p} seconds")
```

```
1/1 [==============================] - 0s 84ms/step
Accuracy: 1.0000
Elapsed time: 0.14316606521606445 seconds
```

Accuracy = 100%

E

Model 2 (Petals only) - Visualized Result



E

The Model 3 (Sepals only) Architecture

```
In [288…

#Sepal model
model_3 = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=(train_x_sepal.shape[1],)),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

Model Compilation

```
In [289…

#Sepal model

model_3.compile(loss=tf.keras.losses.CategoricalCrossentropy(),
                optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                metrics=['accuracy'])
```

E

Model 3 (Sepals only) Prediction:

```
# Training the model
history_sepal = model_3.fit(train_x_sepal, train_y_sepal_categorical,
                            validation_data=(val_x_sepal, val_y_sepal_categorical),
                            epochs=100,
                            batch_size=16,
                            callbacks=[mc, early_stopping])
```

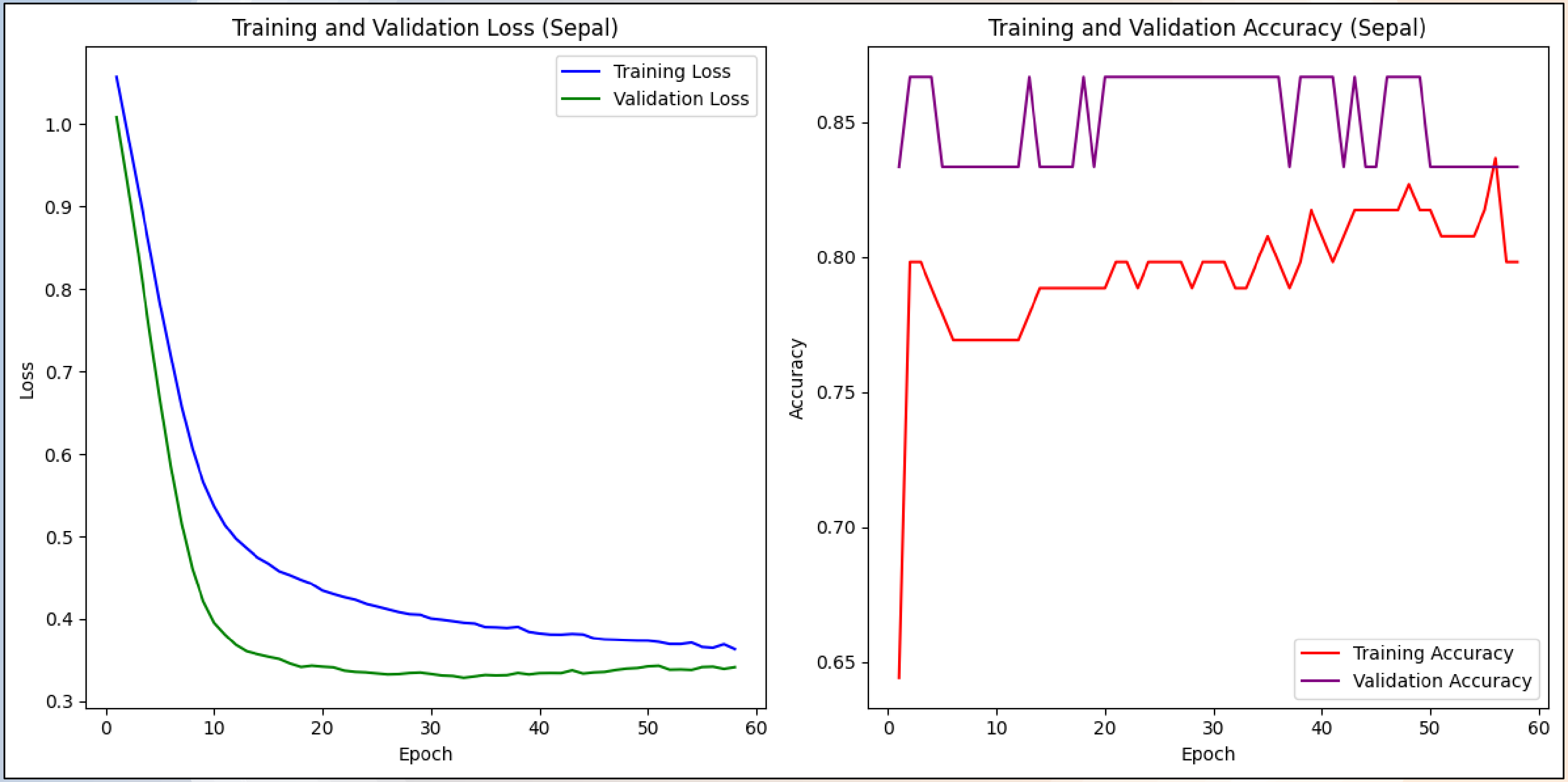
```
[ ] start_time = time.time()
    predictions_sepal = model_3.predict(test_x_sepal, verbose=1)
    predicted_classes_sepal = np.argmax(predictions_sepal, axis=-1)
    predicted_classes_sepal = predicted_classes_sepal.flatten()
    end_time = time.time()

    decoded_predictions_sepal = label_encoder.inverse_transform(predicted_classes_sepal)
    test_y_sepal_decoded = label_encoder.inverse_transform(np.argmax(test_y_sepal_categorical, axis=-1))

    # Calculate and print the accuracy score
    accuracy_CNN_sepal = accuracy_score(test_y_sepal_decoded, decoded_predictions_sepal)
    print(f'Sepal Accuracy: {accuracy_CNN_sepal:.4f}')

    time_CNN_CSV_s = end_time - start_time
    print(f"Elapsed time: {time_CNN_CSV_s} seconds")
```

```
1/1 [==============================] - 0s 31ms/step
Sepal Accuracy: 0.7500
Elapsed time: 0.12656140327453613 seconds
```

Accuracy = 75%

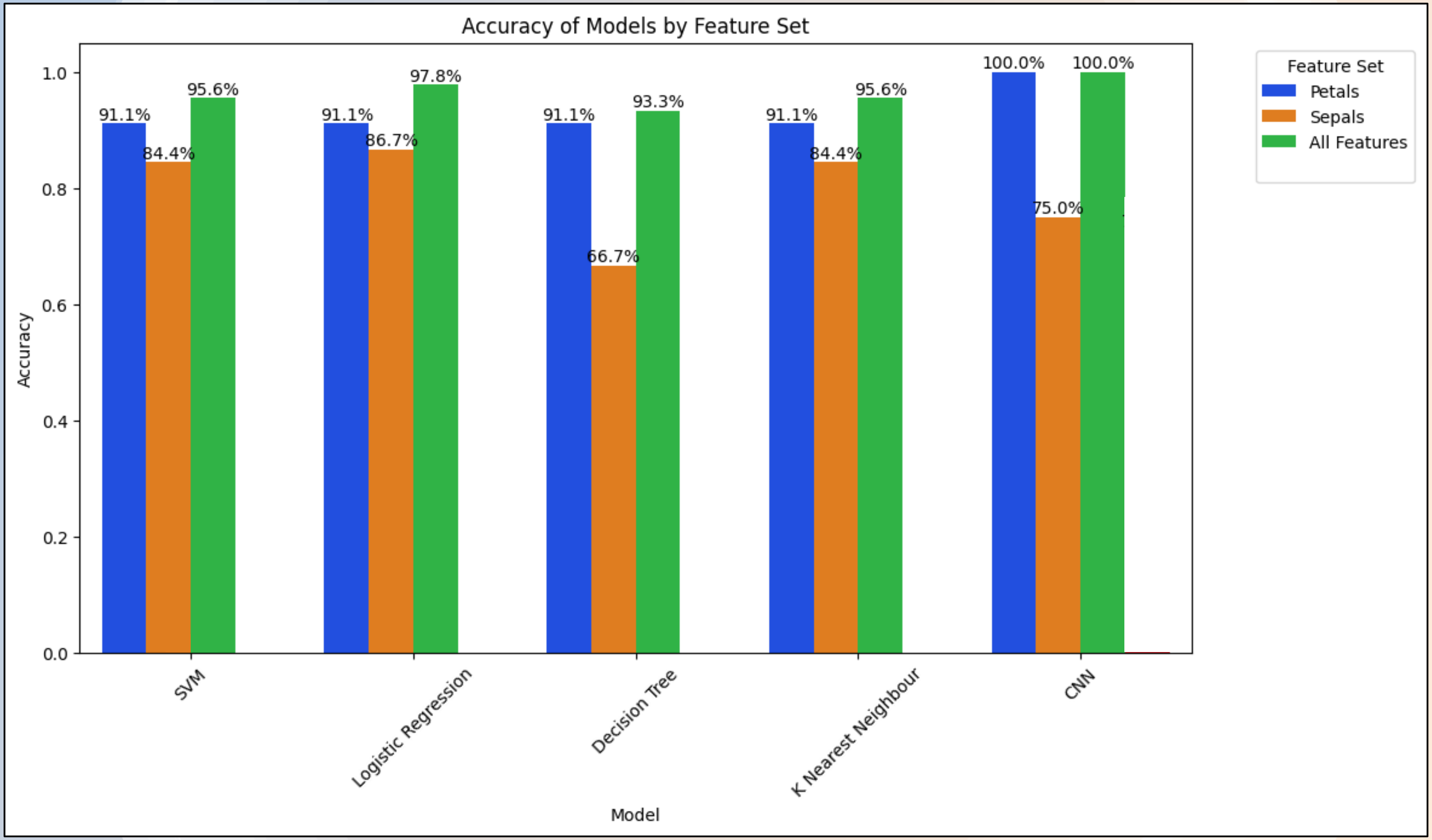
E

Model 3 (Sepals only) - Visualized Result



E

Visualised Result for ML Algorithms & CNN Models with Iris CSV Dataset



C

Summary

Machine Learning algorithm models were easy to implement and required less computational power.

Deep learning models, despite their complexity and intensive computational demands, provided superior performance.

Highest overall accuracy was performed by deep learning model, however; Logistic regression was the highest from machine learning

Deep Learning - 2

(With Iris Image Dataset)

M

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
```

M

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

In [39]:

```python
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
```

M

Splitting the Image Dataset for Train, Test & Validation

```
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,))
```

Training = 80%, Validation = 5% and Testing = 15%

M

The Model 1 Architecture (4 layers)

M

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)
```

M

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 (25% 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(0.001))  # Adding L2 regularization
    ])
```

M

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)
```

M

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
```

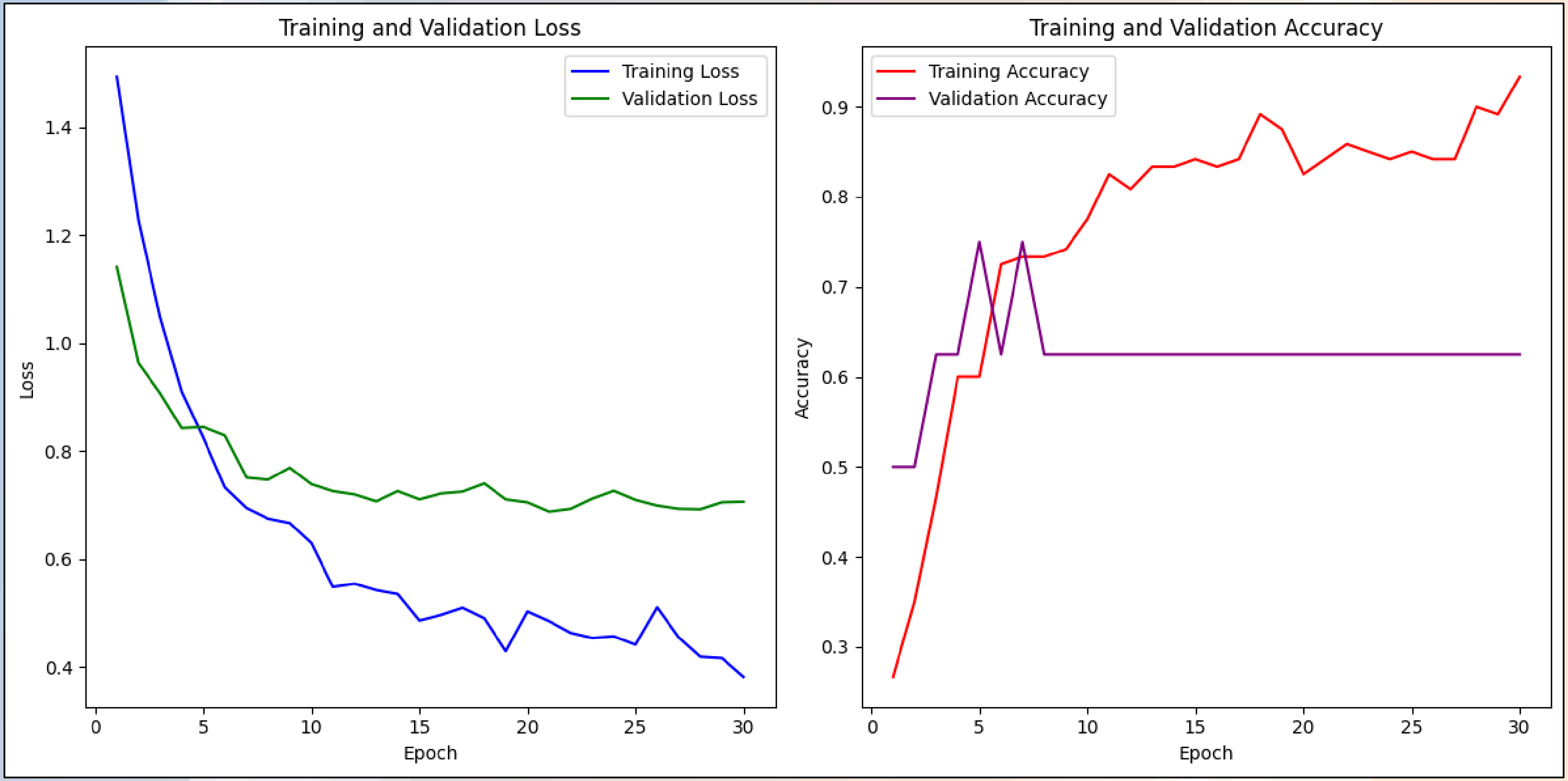
```
[ ] model.evaluate(X_test,y_test_cat)
```

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

Accuracy = 72.7%

M

Model 1 - Visualized Result



M

Model 1 – Results with different Hyperparameters

Dropout Value	Accuracy
0.25	72.7%
0.5	59.5%
0.75	59.5%

L2 Regularization Value	Accuracy
0.01	63.6%
0.001	72.7%
0.0001	59.5%

Batch Size	Accuracy
4	68.6%
8	63.6%
12	72.7%
16	63.6%
24	64.3%

Epochs	Accuracy
10	68.6%
20	63.6%
30	72.7%

Early Stopping Patience	Accuracy
15 (with 100 epochs)	63.6%
25 (with 100 epochs)	63.6%
No early stopping	72.7%

M

```
[ ] 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 ep

```
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.

E

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])
```

E

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:

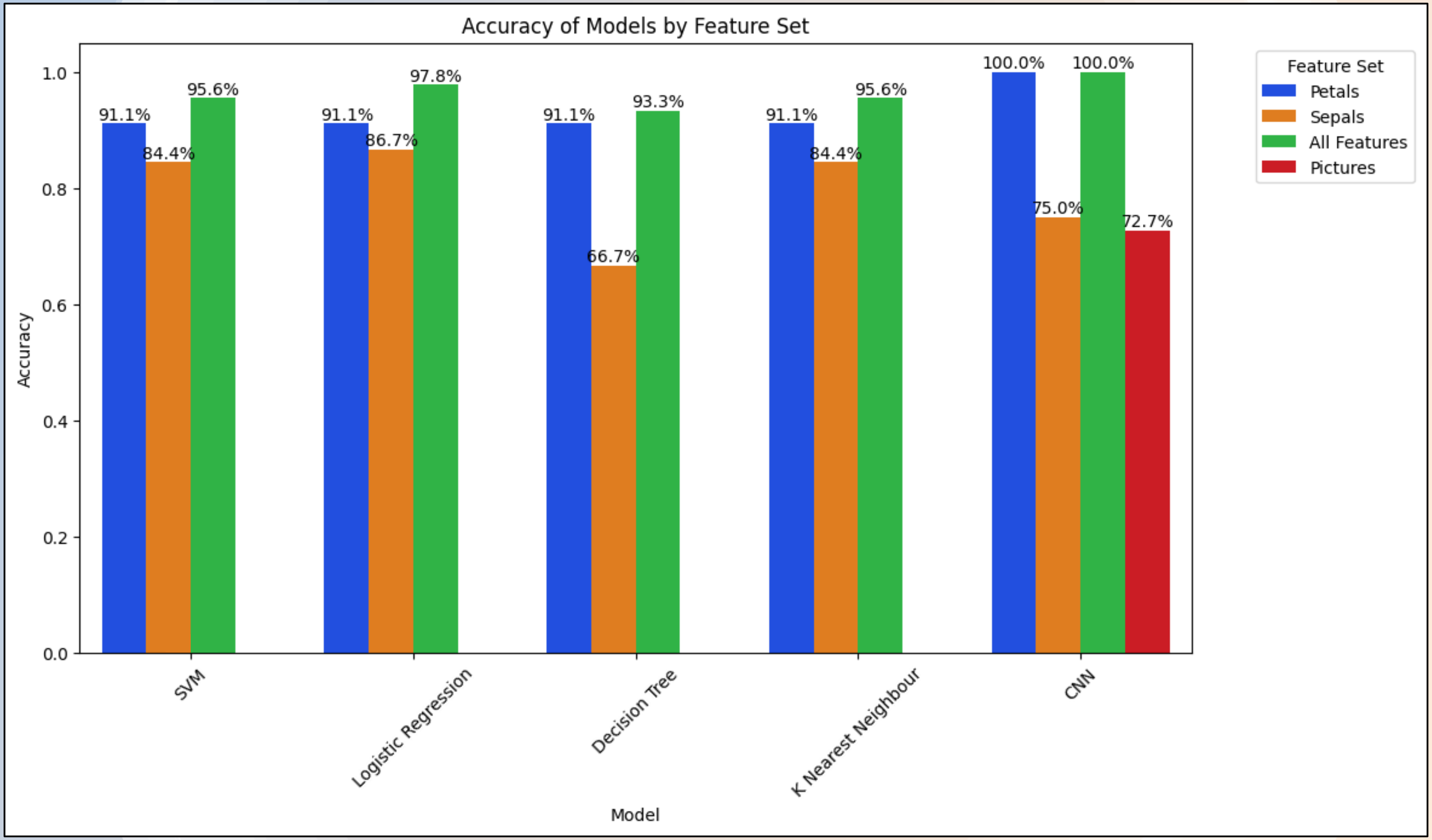
Both models have identical architectures, except for the dropout rate. Model 1 uses a dropout rate of 0.25, whereas Model 2 uses a dropout rate of 0.5.

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.

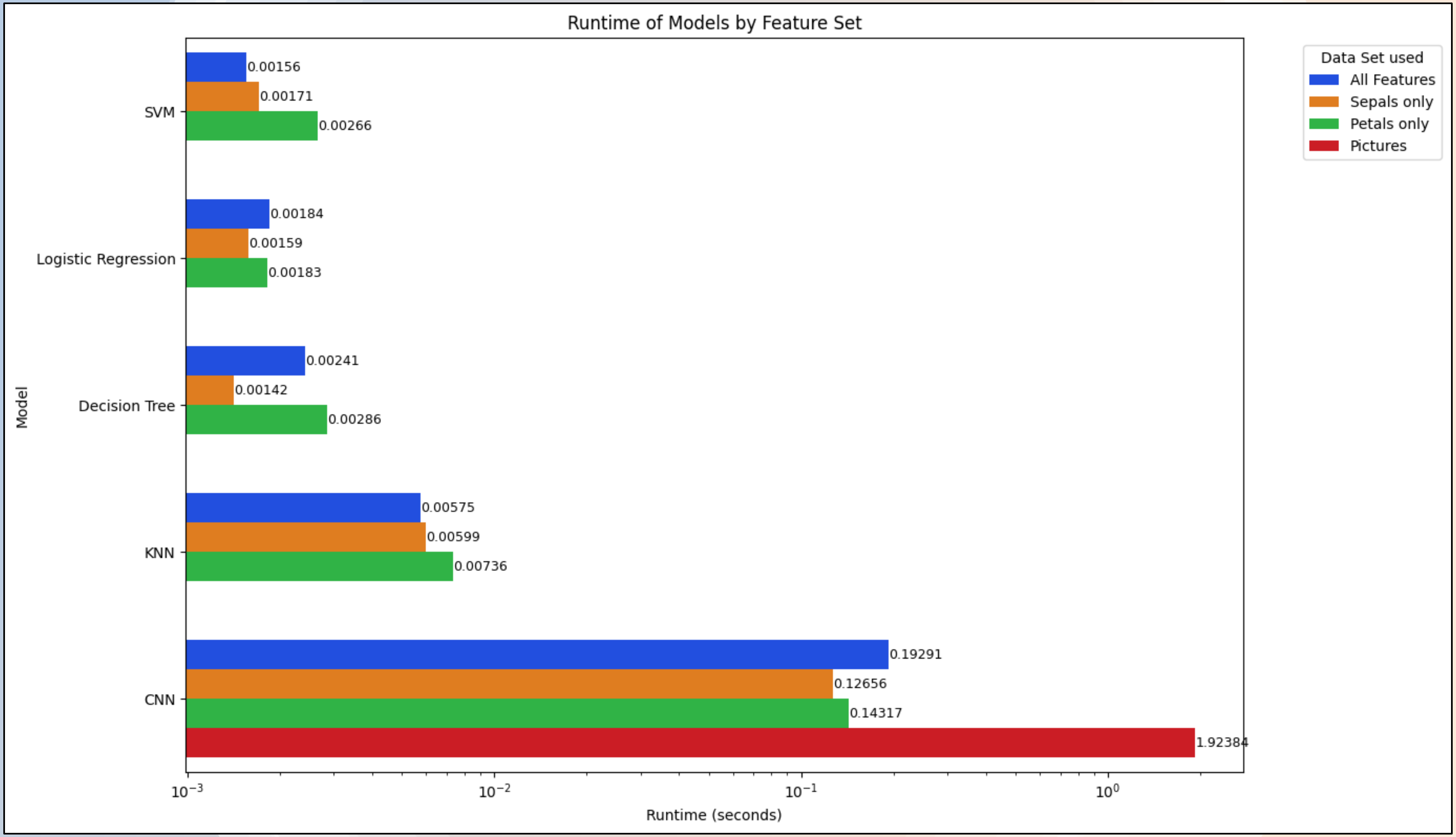
C

Visualised Result for ML Algorithms & CNN Models with both types of datasets



M

Visualised Runtimes of different Models with both types of datasets



E

Conclusion

Project Exploratory Data Analysis (EDA) was valuable experience.

Enhanced understanding of ML and DL algorithms.

Fun and quite simple project :P

Questions & Answers

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