```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'iris:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F19%2F420%2Fbundle%2Farchive.zip%3FX-Goog-Algorithm%3DGO

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Downloading iris, 3687 bytes compressed
[==================================================] 3687 bytes downloaded
```

```
Downloaded and uncompressed: iris
Data source import complete.
```

IRIS FLOWER CLASSIFICATION

image.png

```
#import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


import warnings
warnings.filterwarnings('ignore')


plt.style.use("fivethirtyeight")
%matplotlib inline
```

```
df=pd.read_csv('/kaggle/input/iris/Iris.csv')
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Next steps: Generate code with df View recommended plots

```
#information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

```
#describing about the dataset
df.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

```
df.shape
```

```
(150, 6)
```

We drop the column id because it is not important.

```
df.drop('Id',axis=1,inplace=True)
```

```
df.head()
```

	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

Next steps: Generate code with df View recommended plots

```
#count the value
df['Species'].value_counts()
```

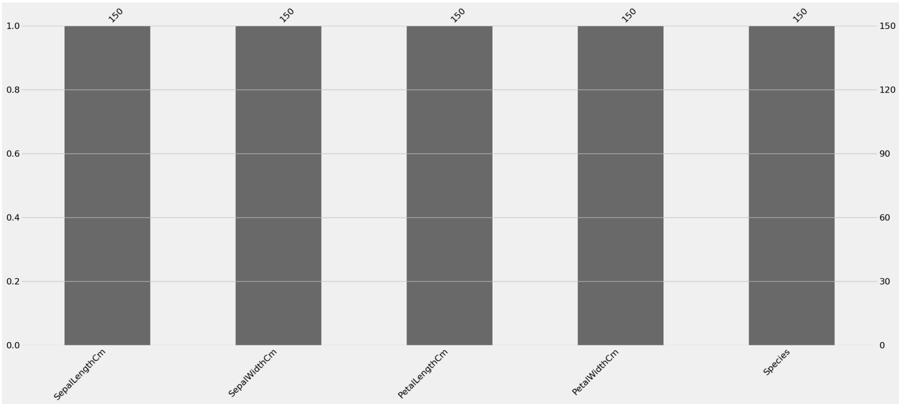
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

```
#finding the null value
df.isnull().sum()
```

```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

```
import missingno as msno
msno.bar(df)
```

```
<Axes: >
```



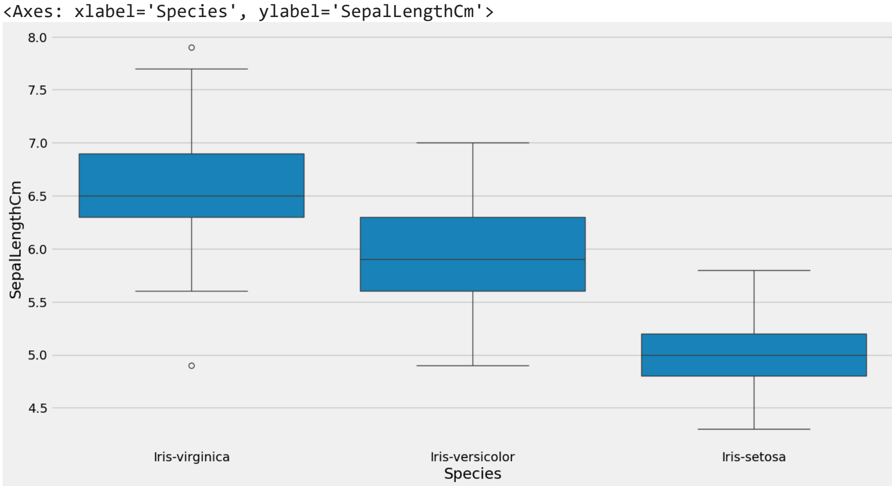

```
df.drop_duplicates(inplace=True)
```

EDA

1. Relationship between species and sepal length

```
plt.figure(figsize=(15,8))
sns.boxplot(x='Species',y='SepalLengthCm',data=df.sort_values('SepalLengthCm',ascending=False))
```

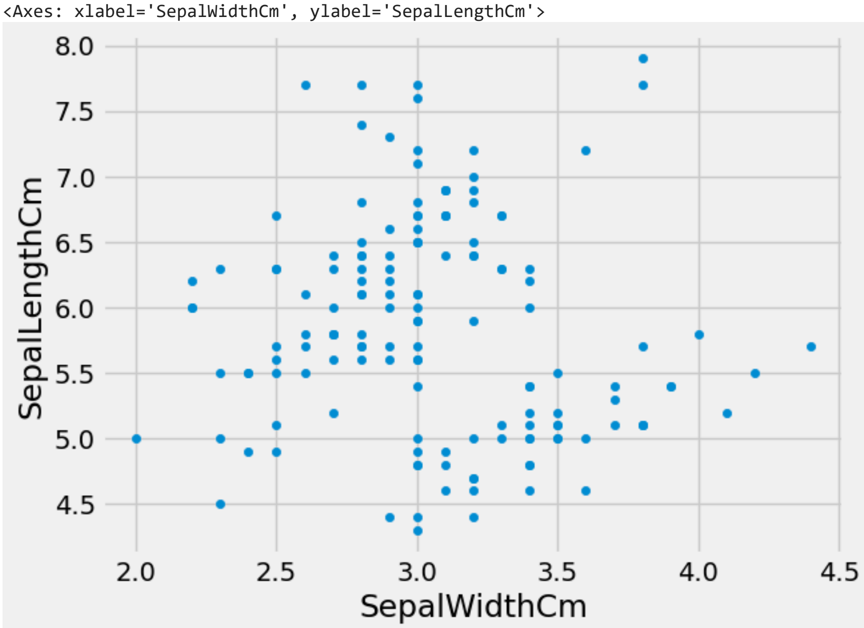
```
<Axes: xlabel='Species', ylabel='SepalLengthCm'>
```



2. Relationship between species and sepal width

```
df.plot(kind='scatter',x='SepalWidthCm',y='SepalLengthCm')
```

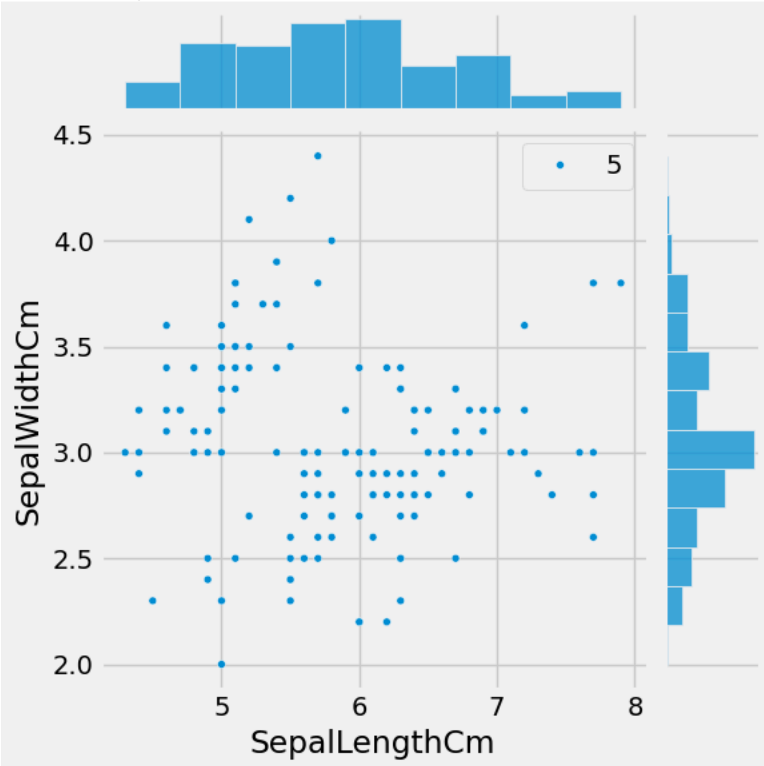
```
<Axes: xlabel='SepalWidthCm', ylabel='SepalLengthCm'>
```



3. Relationship between sepal width and sepal length

```
sns.jointplot(x="SepalLengthCm", y="SepalWidthCm", data=df, size=5)
```

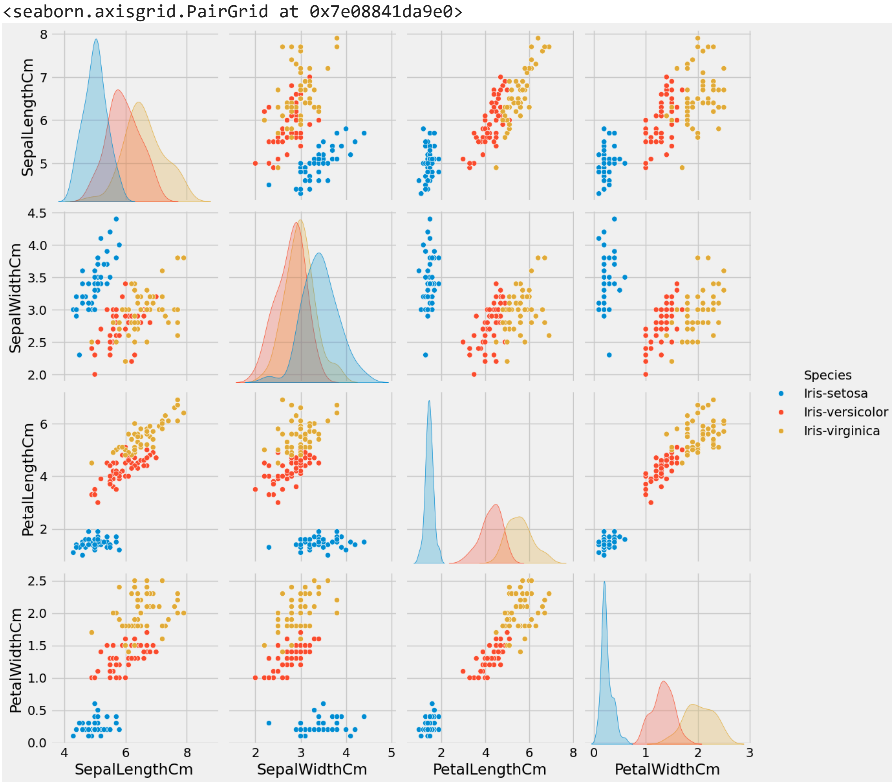
```
<seaborn.axisgrid.JointGrid at 0x7e08841dbdc0>
```



4.Pairplot

```
sns.pairplot(df, hue="Species", size=3)
```

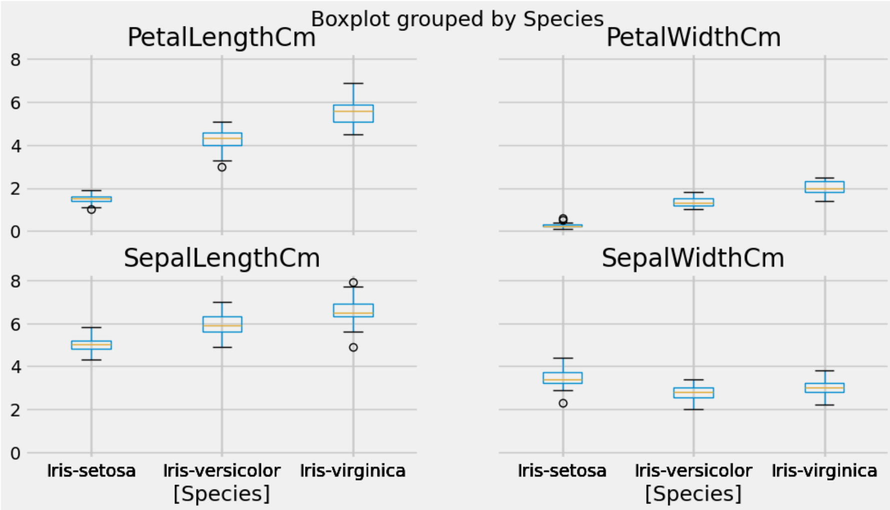
<seaborn.axisgrid.PairGrid at 0x7e08841da9e0>



5. Boxplot

```
df.boxplot(by="Species", figsize=(12, 6))
```

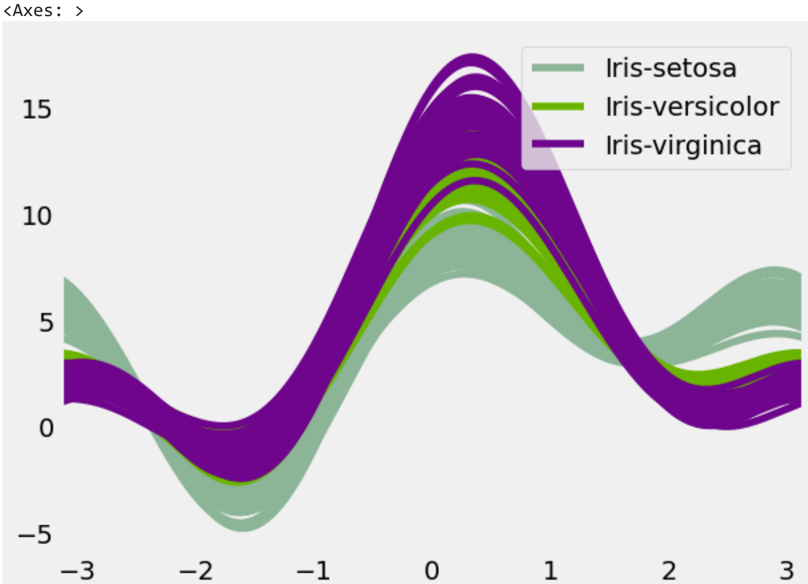
```
array([[<Axes: title={'center': 'PetalLengthCm'}, xlabel='[Species]'>,
        <Axes: title={'center': 'PetalWidthCm'}, xlabel='[Species]'>],
       [<Axes: title={'center': 'SepalLengthCm'}, xlabel='[Species]'>,
        <Axes: title={'center': 'SepalWidthCm'}, xlabel='[Species]'>]],
      dtype=object)
```



5. Andrews_curves

```
import pandas.plotting
from pandas.plotting import andrews_curves
andrews_curves(df, "Species")
```

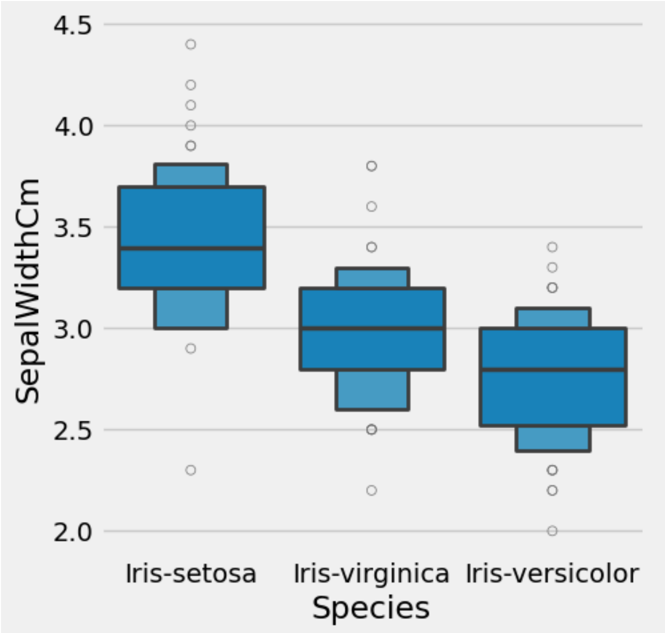
```
<Axes: >
```



6.CategoricalPlot

```
plt.figure(figsize=(15,15))
sns.catplot(x='Species',y='SepalWidthCm',data=df.sort_values('SepalWidthCm',ascending=False),kind='boxen')
```

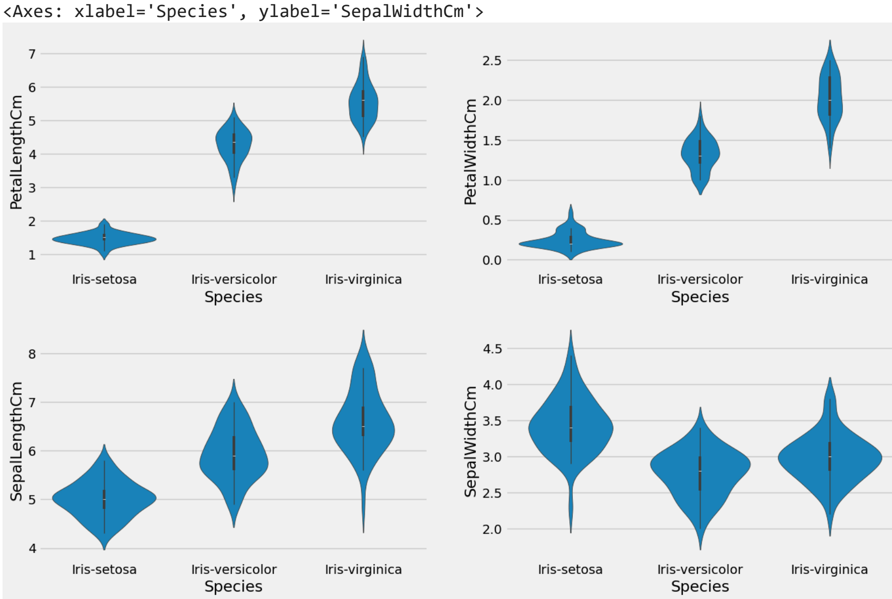
```
<seaborn.axisgrid.FacetGrid at 0x7e087cb907c0>
<Figure size 1500x1500 with 0 Axes>
```



7.Violinplot

```
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.violinplot(x='Species',y='PetalLengthCm',data=df)
plt.subplot(2,2,2)
sns.violinplot(x='Species',y='PetalWidthCm',data=df)
plt.subplot(2,2,3)
sns.violinplot(x='Species',y='SepalLengthCm',data=df)
plt.subplot(2,2,4)
sns.violinplot(x='Species',y='SepalWidthCm',data=df)
```

```
<Axes: xlabel='Species', ylabel='SepalWidthCm'>
```



Neural Network

```
X=df.drop('Species',axis=1)
y=df['Species']
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

```
df['Species'] = pd.Categorical(df.Species)
df['Species'] = df.Species.cat.codes
# Turn response variable into one-hot response vectory = to_categorical(df.response)
y = to_categorical(df.Species)
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30,stratify=y,random_state=123)
```

```
model=Sequential()
model.add(Dense(100,activation='relu',input_shape=(4,)))

model.add(Dense(3,activation='softmax'))


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


history=model.fit(X_train,y_train,epochs=45,validation_data=(X_test, y_test))
```

```
Epoch 1/45
4/4 [==============================] - 1s 86ms/step - loss: 1.5814 - accuracy: 0.0882 - val_loss: 1.4220 - val_accuracy: 0.0222
Epoch 2/45
4/4 [==============================] - 0s 17ms/step - loss: 1.3773 - accuracy: 0.0588 - val_loss: 1.2610 - val_accuracy: 0.2444
Epoch 3/45
4/4 [==============================] - 0s 17ms/step - loss: 1.2333 - accuracy: 0.2745 - val_loss: 1.1725 - val_accuracy: 0.3333
Epoch 4/45
4/4 [==============================] - 0s 17ms/step - loss: 1.1627 - accuracy: 0.1765 - val_loss: 1.1333 - val_accuracy: 0.3333
Epoch 5/45
4/4 [==============================] - 0s 17ms/step - loss: 1.1248 - accuracy: 0.3235 - val_loss: 1.1033 - val_accuracy: 0.3333
Epoch 6/45
4/4 [==============================] - 0s 12ms/step - loss: 1.0972 - accuracy: 0.3333 - val_loss: 1.0696 - val_accuracy: 0.3333
Epoch 7/45
4/4 [==============================] - 0s 12ms/step - loss: 1.0647 - accuracy: 0.3333 - val_loss: 1.0346 - val_accuracy: 0.3333
Epoch 8/45
4/4 [==============================] - 0s 17ms/step - loss: 1.0257 - accuracy: 0.3333 - val_loss: 0.9974 - val_accuracy: 0.3333
Epoch 9/45
4/4 [==============================] - 0s 17ms/step - loss: 0.9916 - accuracy: 0.3333 - val_loss: 0.9629 - val_accuracy: 0.4000
Epoch 10/45
4/4 [==============================] - 0s 18ms/step - loss: 0.9620 - accuracy: 0.5098 - val_loss: 0.9345 - val_accuracy: 0.6889
Epoch 11/45
4/4 [==============================] - 0s 12ms/step - loss: 0.9371 - accuracy: 0.6569 - val_loss: 0.9093 - val_accuracy: 0.8000
Epoch 12/45
4/4 [==============================] - 0s 12ms/step - loss: 0.9149 - accuracy: 0.7157 - val_loss: 0.8882 - val_accuracy: 0.8667
Epoch 13/45
4/4 [==============================] - 0s 19ms/step - loss: 0.8951 - accuracy: 0.7451 - val_loss: 0.8661 - val_accuracy: 0.8667
Epoch 14/45
4/4 [==============================] - 0s 13ms/step - loss: 0.8696 - accuracy: 0.7745 - val_loss: 0.8422 - val_accuracy: 0.6889
Epoch 15/45
4/4 [==============================] - 0s 13ms/step - loss: 0.8499 - accuracy: 0.6569 - val_loss: 0.8247 - val_accuracy: 0.6667
Epoch 16/45
4/4 [==============================] - 0s 17ms/step - loss: 0.8321 - accuracy: 0.6569 - val_loss: 0.8054 - val_accuracy: 0.6667
Epoch 17/45
4/4 [==============================] - 0s 17ms/step - loss: 0.8127 - accuracy: 0.6569 - val_loss: 0.7833 - val_accuracy: 0.6667
Epoch 18/45
4/4 [==============================] - 0s 17ms/step - loss: 0.7909 - accuracy: 0.6667 - val_loss: 0.7648 - val_accuracy: 0.8222
Epoch 19/45
4/4 [==============================] - 0s 17ms/step - loss: 0.7820 - accuracy: 0.8137 - val_loss: 0.7558 - val_accuracy: 0.8000
Epoch 20/45
4/4 [==============================] - 0s 17ms/step - loss: 0.7663 - accuracy: 0.7647 - val_loss: 0.7357 - val_accuracy: 0.8889
Epoch 21/45
4/4 [==============================] - 0s 18ms/step - loss: 0.7433 - accuracy: 0.8824 - val_loss: 0.7137 - val_accuracy: 0.7778
Epoch 22/45
4/4 [==============================] - 0s 18ms/step - loss: 0.7238 - accuracy: 0.6961 - val_loss: 0.7010 - val_accuracy: 0.6667
Epoch 23/45
4/4 [==============================] - 0s 18ms/step - loss: 0.7126 - accuracy: 0.6667 - val_loss: 0.6873 - val_accuracy: 0.6667
Epoch 24/45
4/4 [==============================] - 0s 12ms/step - loss: 0.6968 - accuracy: 0.6863 - val_loss: 0.6706 - val_accuracy: 0.8000
Epoch 25/45
4/4 [==============================] - 0s 17ms/step - loss: 0.6816 - accuracy: 0.8529 - val_loss: 0.6566 - val_accuracy: 0.8889
Epoch 26/45
4/4 [==============================] - 0s 18ms/step - loss: 0.6669 - accuracy: 0.9314 - val_loss: 0.6413 - val_accuracy: 0.9111
Epoch 27/45
4/4 [==============================] - 0s 18ms/step - loss: 0.6508 - accuracy: 0.9608 - val_loss: 0.6253 - val_accuracy: 0.9111
Epoch 28/45
4/4 [==============================] - 0s 17ms/step - loss: 0.6346 - accuracy: 0.9216 - val_loss: 0.6112 - val_accuracy: 0.8444
Epoch 29/45
4/4 [==============================] - 0s 17ms/step - loss: 0.6221 - accuracy: 0.8039 - val_loss: 0.6008 - val_accuracy: 0.7556
```

```
model.evaluate(X_test,y_test)
```

```
2/2 [==============================] - 0s 8ms/step - loss: 0.4600 - accuracy: 0.9556
[0.4600415527820587, 0.9555555582046509]
```

```
pred = model.predict(X_test[:10])
print(pred)
```

```
1/1 [==============================] - 0s 147ms/step
[[0.01529051 0.36611608 0.6185935 ]
 [0.01422953 0.3689779  0.6167926 ]
 [0.11658246 0.52779883 0.3556187 ]
```

```
 [0.07879135 0.5133856  0.4078232 ]
 [0.8309323  0.14295849 0.02610906]
 [0.07750668 0.5431579  0.37933546]
 [0.01420907 0.32426172 0.6615291 ]
 [0.01555702 0.33452058 0.6499225 ]
 [0.83395183 0.14095311 0.02509506]
 [0.02674764 0.39094636 0.58230597]]
```

```
p=np.argmax(pred,axis=1)
print(p)
print(y_test[:10])
```

```
[2 2 1 1 0 1 2 2 0 2]
[[0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]]
```

```
history.history['accuracy']
```

```
[0.0882352963089943,
 0.05882352963089943,
 0.27450981736183167,
 0.1764705926179886,
 0.3235294222831726,
 0.3333333432674408,
```