In [1]:
```
# Analysis On Iris Dataset
```

In [2]:
```
#Exploratory Data Analysis (EDA) is a critical step in the data analysis process
#that involves visually and statistically exploring the characteristics of a dataset.

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
iris=pd.read_excel("C:/Users/Admin/Downloads//iris.xlsx")
iris.head()
```

Out[3]:

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

In [4]:
```
iris.describe()
```

Out[4]:

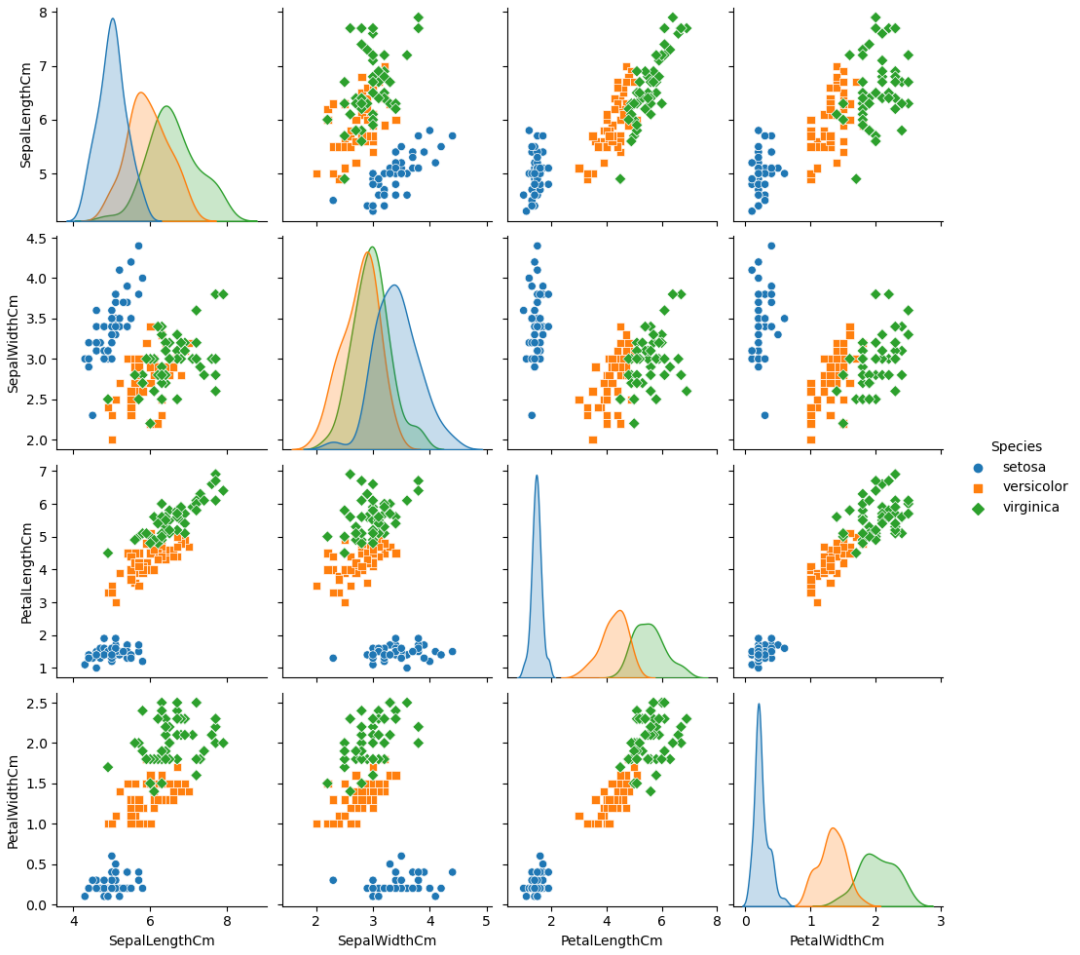
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	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

In [5]:
```
iris=iris.drop('Id',axis=1)
```

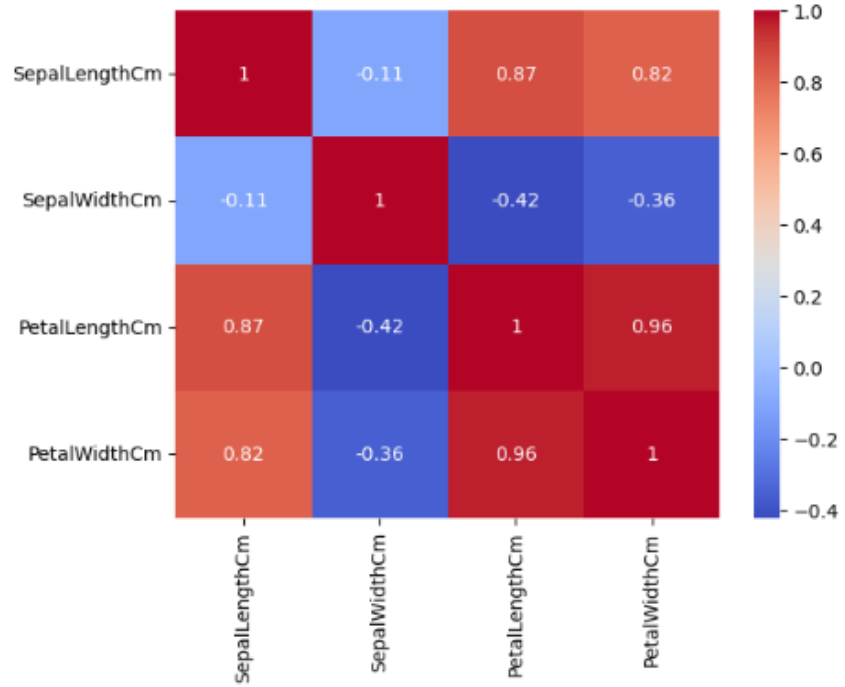
In [6]:
```
print(iris.info())
```

```
<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
None
```

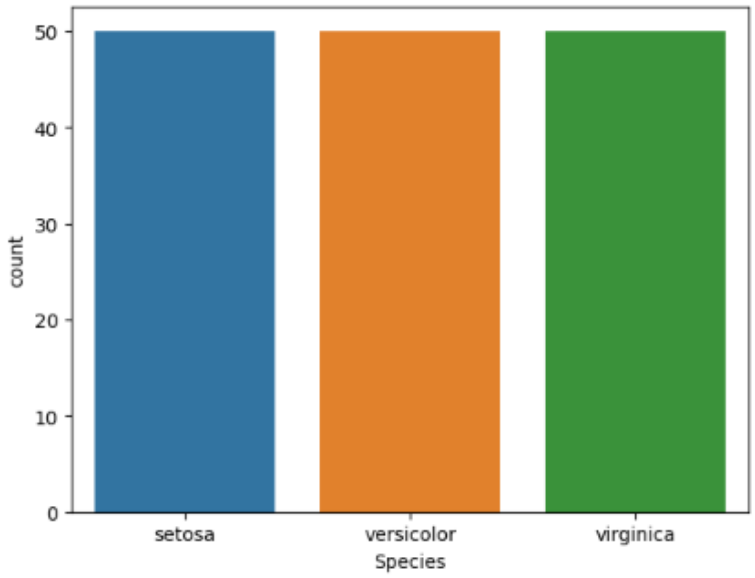
In [7]:
```
#Visualization
sns.pairplot(iris,hue='Species',markers=['o','s','D'])# "o" represents circles."s" represents squares.
plt.show()
```



In [8]:
```
# Correlation Analysis
correlation_matrix=iris.corr()
#Heatmap
sns.heatmap(correlation_matrix,annot=True,cmap='coolwarm')
plt.show()
```



In [9]:
```
# Categorical Data Analysis
sns.countplot(x='Species',data=iris)
plt.show()
```

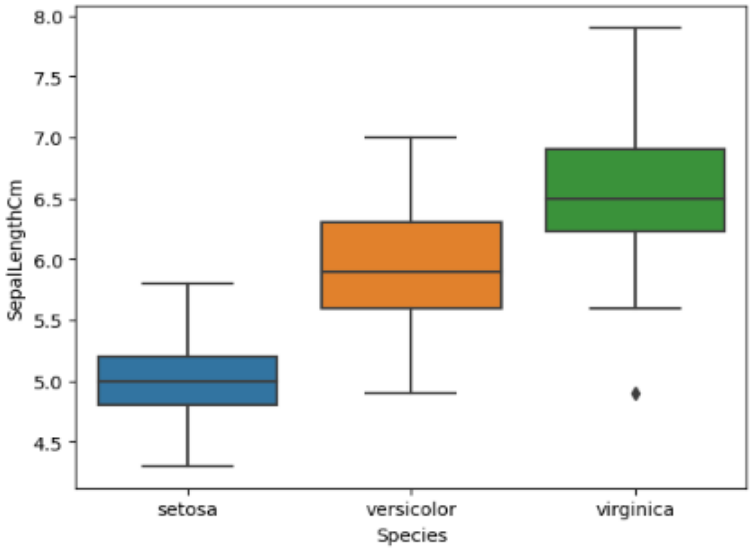


In [10]:
```
#Handling Missing Data
print(iris.isnull().sum())
```

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

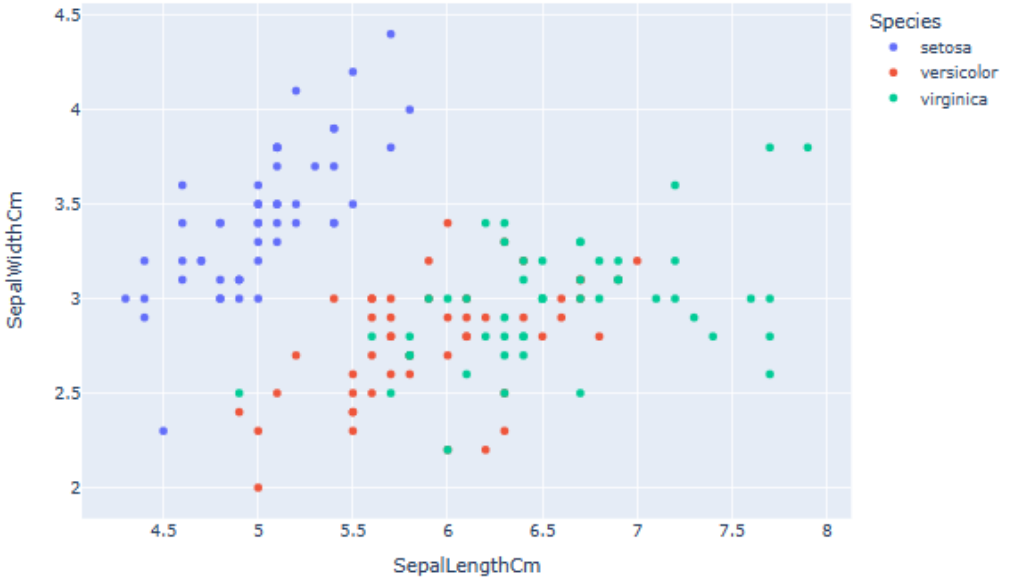
In [11]:
```
# Outliers Detection
#boxplot
sns.boxplot(x="Species",y="SepalLengthCm",data=iris)
plt.show()
```

In [11]:
```
# Outliers Detection
#boxplot
sns.boxplot(x="Species",y="SepalLengthCm",data=iris)
plt.show()
```



In [12]:
```
# Interactive Visualization
import plotly.express as px

# Interactive scatter plot
fig = px.scatter(iris, x='SepalLengthCm', y='SepalWidthCm', color='Species')
fig.show()
```



In [13]:
```
# Classification(Decision Tree)
from sklearn.metrics import accuracy_score,classification_report
from sklearn.model_selection import train_test_split as tts
from sklearn.tree import DecisionTreeClassifier
```

In [14]:
```
x=iris.drop("Species",axis=1)
y=iris[["Species"]]
x_train,x_test,y_train,y_test=tts(x,y,test_size=0.2,random_state=42)
x_train.shape,y_train.shape
```

Out[14]: ((120, 4), (120, 1))

In [15]:
```
# Create DT Model
from sklearn.tree import DecisionTreeClassifier
clt=DecisionTreeClassifier()

#Train the model
clt.fit(x_train,y_train)

# Prediction on the Test
y_pred=clt.predict(x_test)

#accuracy_score
#acc_score=sccuracy_score(y_test,y_pred)
#classification_report(y_test,y_pred)
print("Accuracy:",accuracy_score(y_test,y_pred))
print("Classification_Report:\n",classification_report(y_test,y_pred))
```

```
Accuracy: 1.0
Classification_Report:
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   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
```

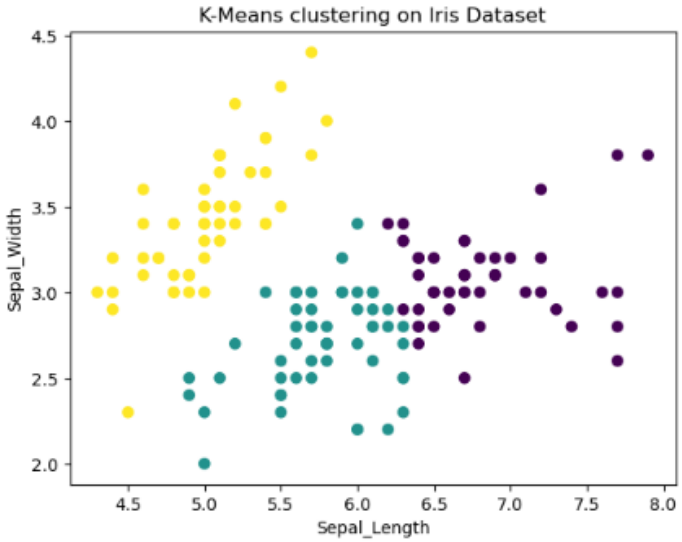
In [16]:
```
# Clustering(K-Means) # Unsupervised Learning
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns

# Only 2 features for visualization purpose
x_cluster=iris[["SepalLengthCm","SepalWidthCm"]]

# Apply KMeans Clustering
k_means=KMeans(n_clusters=3,random_state=42)
k_means.fit(x_cluster)

# Add cluster labels to data
iris["cluster"]=k_means.labels_

# Visualize the data
plt.scatter(iris["SepalLengthCm"],iris["SepalWidthCm"],c=iris['cluster'],cmap="viridis")
plt.title("K-Means clustering on Iris Dataset")
plt.xlabel("Sepal_Length")
plt.ylabel("Sepal_Width")
plt.show()
```



In [17]:
```
# Apriori Rule Mining
# Unsupervised Leaning technique
import pandas as pd
from mlxtend.frequent_patterns import apriori
from mlxtend.preprocessing import TransactionEncoder

# Convert the species column as one hot encodng
te = TransactionEncoder()
te_array=te.fit(iris[["Species"]]).transform(iris[["Species"]])
df_encoded=pd.DataFrame(te_array,columns=te.columns_)

# Apply apriori algorithm
freq_itemsets=apriori(df_encoded,min_support=0.005,use_colnames=True)
print(freq_itemsets)
```

```
     support           itemsets
0   0.006667                (S)
1   0.006667                (c)
2   0.006667                (e)
3   0.006667                (i)
4   0.006667                (p)
..       ...                ...
58  0.006667    (S, p, e, s, c)
59  0.006667    (S, p, s, c, i)
60  0.006667    (S, p, e, s, i)
61  0.006667    (p, e, s, c, i)
62  0.006667 (S, p, e, s, c, i)

[63 rows x 2 columns]
```

In [18]:
```
# Association Rule
# Unsupervised Leaning technique

from mlxtend.frequent_patterns import apriori, association_rules
df_encoded=pd.DataFrame(te_array,columns=te.columns_)
frequent_itemsets = apriori(df_encoded, min_support=0.005, use_colnames=True)

# Generate association rules
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.5)

# Display the association rules
print(rules)
```

```
    antecedents      consequents  antecedent support  consequent support  \
0           (S)              (c)            0.006667            0.006667
1           (c)              (S)            0.006667            0.006667
2           (S)              (e)            0.006667            0.006667
3           (e)              (S)            0.006667            0.006667
4           (S)              (i)            0.006667            0.006667
..          ...              ...                 ...                 ...
597         (p)  (S, e, s, c, i)            0.006667            0.006667
598         (e)  (S, p, s, c, i)            0.006667            0.006667
599         (s)  (S, p, e, c, i)            0.006667            0.006667
600         (c)  (S, p, e, s, i)            0.006667            0.006667
601         (i)  (S, p, e, s, c)            0.006667            0.006667

      support  confidence   lift  leverage  conviction  zhangs_metric
0    0.006667         1.0  150.0  0.006622         inf            1.0
1    0.006667         1.0  150.0  0.006622         inf            1.0
2    0.006667         1.0  150.0  0.006622         inf            1.0
3    0.006667         1.0  150.0  0.006622         inf            1.0
4    0.006667         1.0  150.0  0.006622         inf            1.0
..        ...         ...    ...       ...         ...            ...
597  0.006667         1.0  150.0  0.006622         inf            1.0
598  0.006667         1.0  150.0  0.006622         inf            1.0
599  0.006667         1.0  150.0  0.006622         inf            1.0
600  0.006667         1.0  150.0  0.006622         inf            1.0
601  0.006667         1.0  150.0  0.006622         inf            1.0

[602 rows x 10 columns]
```

In [19]:
```
# Random Forest
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [20]:
```
from sklearn.model_selection import train_test_split as tts
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,classification_report
```

In [21]:
```
x=iris.drop("Species",axis=1)
y=iris[["Species"]]
x_train,x_test,y_train,y_test=tts(x,y,test_size=0.2,random_state=42)
x_train.shape,y_train.shape
```

Out[21]:
```
((120, 5), (120, 1))
```

In [22]:
```
rf_classifier=RandomForestClassifier(random_state=42)
rf_classifier.fit(x_train,y_train)
y_pred=rf_classifier.predict(x_test)
print("Accuracy:",accuracy_score(y_test,y_pred))
print("Random Forest Classification Report:\n",classification_report(y_test,y_pred))
```

```
Accuracy: 1.0
Random Forest Classification Report:
               precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   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
```

In [23]:
```
# SVM
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score,classification_report
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split as tts
import warnings
warnings.filterwarnings("ignore")
```

In [24]:
```
# Load the Iris dataset
iris = sns.load_dataset('iris')

# Split the data into features (X) and target variable (y)
x = iris.drop('species', axis=1)
y = iris['species']

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = tts(x, y, test_size=0.2, random_state=42)

# SVM Classifier
svm=SVC()

#train the model
svm_classifier=svm.fit(x_train,y_train)

# Make predictons on the test set
y_pred=svm_classifier.predict(x_test)

# Evaluate the model
print("Accuracy:",accuracy_score(y_test,y_pred))
print("SVM Classification Report:\n",classification_report(y_test,y_pred))
```

```
Accuracy: 1.0
SVM Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   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
```

In [25]:
```
# Neural Network
!pip install tensorflow
```

```
es (from google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflo
w) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\admin\anaconda3\lib\site-packag
es (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (2.0.
4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\admin\anaconda3\lib\site-packages (from req
uests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\admin\anaconda3\lib\site-packages (fr
om requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\admin\anaconda3\lib\site-packages (fr
om requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (2023.7.22)
Requirement already satisfied: MarkupSafe>=2.1.1 in c:\users\admin\anaconda3\lib\site-packages (fro
m werkzeug>=1.0.1->tensorboard<2.16,>=2.15->tensorflow-intel==2.15.0->tensorflow) (2.1.1)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in c:\users\admin\anaconda3\lib\site-packages
(from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow-intel==2.1
5.0->tensorflow) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in c:\users\admin\anaconda3\lib\site-packages (from
requests-oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow-intel==
2.15.0->tensorflow) (3.2.2)
```

In [26]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
from sklearn.metrics import accuracy_score,classification_report
from sklearn.model_selection import train_test_split as tts
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings("ignore")

# LabelEncoder
le=LabelEncoder()
y=le.fit_transform(y)

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = tts(x, y, test_size=0.2, random_state=42)

# Build a nn network
model=tf.keras.Sequential([
    tf.keras.layers.Dense(64,activation='relu',input_shape=(x_train.shape[1],)),
    tf.keras.layers.Dense(3,activation='softmax')
])

#complile the model
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=["accuracy"])

#Train the model
model.fit(x_train,y_train,epochs=50,batch_size=32,validation_split=0.1,verbose=0)

# Evaluate the model on test data
y_pred=model.predict(x_test)
y_pred_nn=tf.argmax(y_pred,axis=1).numpy()

# Evaluate the model
print("Neural Network Accuracy:", accuracy_score(y_test, y_pred_nn))
print("Neural Network Classification Report:\n", classification_report(y_test, y_pred_nn))
```

```
WARNING:tensorflow:From C:\Users\Admin\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name
tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_c
ross_entropy instead.

WARNING:tensorflow:From C:\Users\Admin\anaconda3\Lib\site-packages\keras\src\backend.py:873: The name
tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\Admin\anaconda3\Lib\site-packages\keras\src\optimizers\__init__.py:3
09: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

WARNING:tensorflow:From C:\Users\Admin\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: T
he name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue i
nstead.

WARNING:tensorflow:From C:\Users\Admin\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.
py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.execut
ing_eagerly_outside_functions instead.

1/1 [==============================] - 0s 107ms/step
Neural Network Accuracy: 0.9666666666666667
Neural Network Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.90      1.00      0.95         9
           2       1.00      0.91      0.95        11

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

In [27]:
```
# KNN
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score,classification_report
from sklearn.model_selection import train_test_split as tts
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings("ignore")

# Create a KNN Classifier
knn=KNeighborsClassifier(n_neighbors=3)

# train the model
knn.fit(x_train,y_train)

# Make predictions
y_pred=knn.predict(x_test)

# Evaluate the model
print("Accuracy:",accuracy_score(y_test,y_pred))
print("KNN Clssification Report:\n",classification_report(y_test,y_pred))
```

```
Accuracy: 1.0
KNN Clssification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       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
```

In [28]:
```
#PCA
#Unsupervised Learning

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score,classification_report
from sklearn.model_selection import train_test_split as tts
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings("ignore")

# Apply pca to reduce dimensionality
pca=PCA(n_components=2)
x_train_pca=pca.fit_transform(x_train)
x_test_pca=pca.transform(x_test)

# Create and train a classifier
clf=RandomForestClassifier(random_state=42)
clf.fit(x_train_pca,y_train)

# Make predictions
y_pred=clf.predict(x_test_pca)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9666666666666667
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       0.90      1.00      0.95         9
           2       1.00      0.91      0.95        11

    accuracy                           0.97        30
   macro avg       0.97      0.97      0.97        30
weighted avg       0.97      0.97      0.97        30
```

In [29]:

```
# Logistic Reg for Binary Classification

#import pandas as pd
#from sklearn.model_selection import train_test_split
#from sklearn.linear_model import LogisticRegression
#from sklearn.metrics import accuracy_score, classification_report
#import seaborn as sns

# Load the Iris dataset
#iris = sns.load_dataset('iris')

# Create a binary target variable
#iris['is_setosa'] = (iris['species'] == 'setosa').astype(int)

# Split the data into features (X) and binary target variable (y)
#X = iris.drop(['species', 'is_setosa'], axis=1)
#y = iris['is_setosa']

# Split the data into training and testing sets
#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a logistic regression model
#logreg_model = LogisticRegression()

# Train the model
#logreg_model.fit(X_train, y_train)

# Make predictions on the test set
#y_pred_logreg = logreg_model.predict(X_test)

# Evaluate the model
#print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_logreg))
#print("Logistic Regression Classification Report:\n", classification_report(y_test, y_pred_logreg))
```