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

```
iris = pd.read_csv('/content/IRIS.csv')
```

```
iris.head(10)
```

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| **5** | 5.4 | 3.9 | 1.7 | 0.4 | Iris-setosa |
| **6** | 4.6 | 3.4 | 1.4 | 0.3 | Iris-setosa |
| **7** | 5.0 | 3.4 | 1.5 | 0.2 | Iris-setosa |
| **8** | 4.4 | 2.9 | 1.4 | 0.2 | Iris-setosa |
| **9** | 4.9 | 3.1 | 1.5 | 0.1 | Iris-setosa |

```
iris.tail(10)
```

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| **140** | 6.7 | 3.1 | 5.6 | 2.4 | Iris-virginica |
| **141** | 6.9 | 3.1 | 5.1 | 2.3 | Iris-virginica |
| **142** | 5.8 | 2.7 | 5.1 | 1.9 | Iris-virginica |
| **143** | 6.8 | 3.2 | 5.9 | 2.3 | Iris-virginica |
| **144** | 6.7 | 3.3 | 5.7 | 2.5 | Iris-virginica |
| **145** | 6.7 | 3.0 | 5.2 | 2.3 | Iris-virginica |
| **146** | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| **147** | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| **148** | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| **149** | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

```
iris.shape
```

```
(150, 5)
```

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
iris.isna().sum()
```

| | 0 |
|---|---|
| **sepal_length** | 0 |
| **sepal_width** | 0 |
| **petal_length** | 0 |
| **petal_width** | 0 |
| **species** | 0 |

**dtype:** int64

```
iris.duplicated().sum()
```

```
3
```

```
iris= iris.drop_duplicates()
```

```
iris.shape
```

```
(147, 5)
```

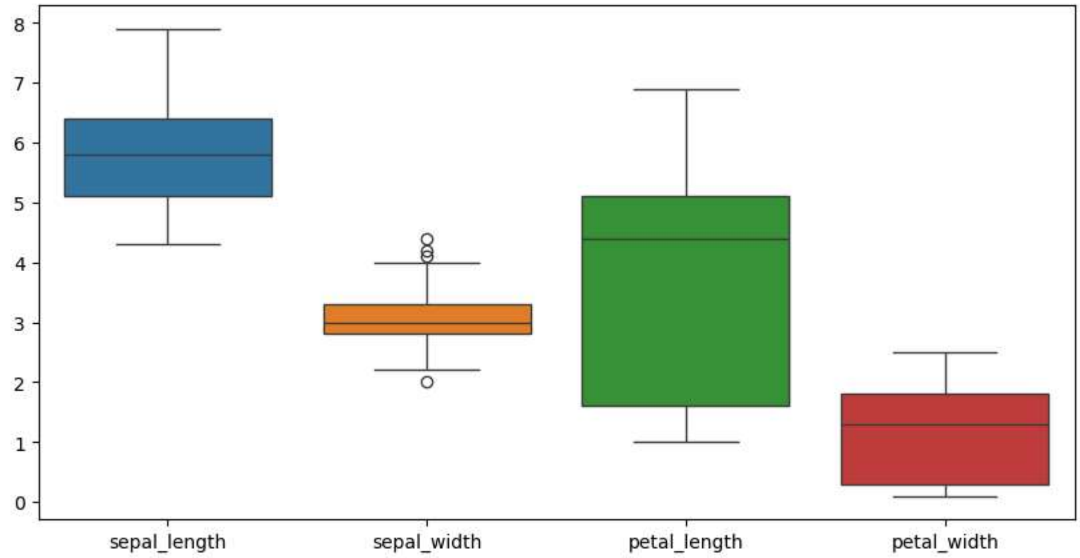
```
iris
```

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |
| **...** | ... | ... | ... | ... | ... |
| **145** | 6.7 | 3.0 | 5.2 | 2.3 | Iris-virginica |
| **146** | 6.3 | 2.5 | 5.0 | 1.9 | Iris-virginica |
| **147** | 6.5 | 3.0 | 5.2 | 2.0 | Iris-virginica |
| **148** | 6.2 | 3.4 | 5.4 | 2.3 | Iris-virginica |
| **149** | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |

147 rows × 5 columns

```
fig,ax = plt.subplots(figsize=(10,5))
sns.boxplot(iris,ax=ax)
```

```
<Axes: >
```




```
iris.describe()
```

| | sepal_length | sepal_width | petal_length | petal_width |
|---|---|---|---|---|
| count | 147.000000 | 147.000000 | 147.000000 | 147.000000 |
| mean | 5.856463 | 3.055782 | 3.780272 | 1.208844 |
| std | 0.829100 | 0.437009 | 1.759111 | 0.757874 |
| min | 4.300000 | 2.000000 | 1.000000 | 0.100000 |
| 25% | 5.100000 | 2.800000 | 1.600000 | 0.300000 |
| 50% | 5.800000 | 3.000000 | 4.400000 | 1.300000 |
| 75% | 6.400000 | 3.300000 | 5.100000 | 1.800000 |
| max | 7.900000 | 4.400000 | 6.900000 | 2.500000 |

```
iris.species.unique()
```

```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

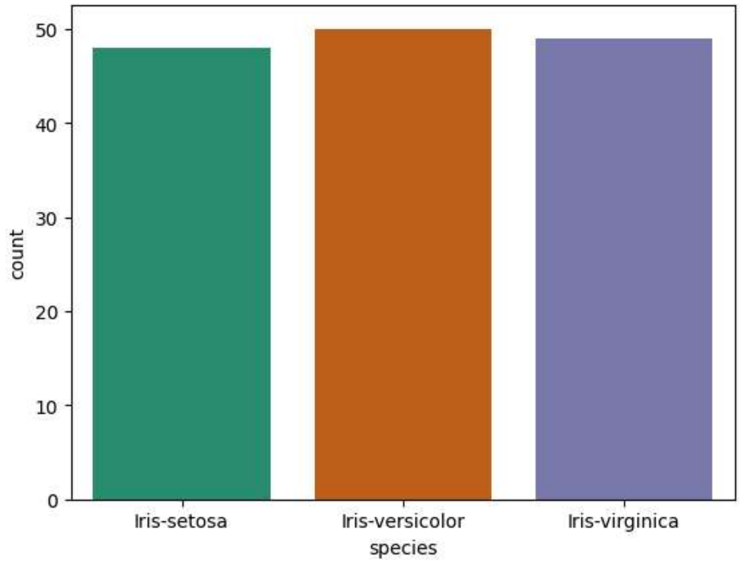
```
iris['species'].value_counts()
```

| species | count |
|---|---|
| Iris-versicolor | 50 |
| Iris-virginica | 49 |
| Iris-setosa | 48 |

**dtype:** int64

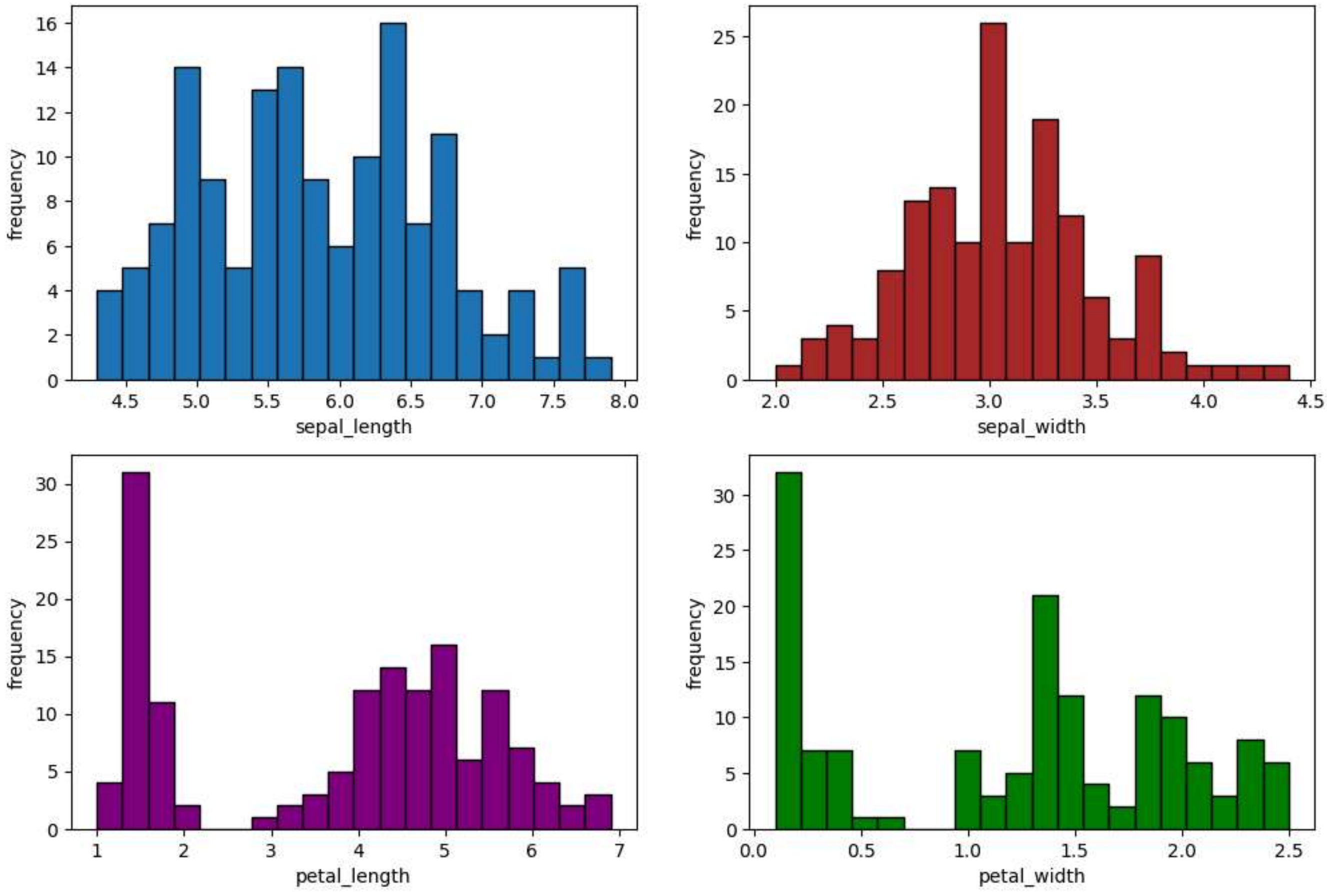
```
sns.countplot(x = "species",data=iris,palette="Dark2")
```

```
<ipython-input-18-481d8148ff53>:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.countplot(x = "species",data=iris,palette="Dark2")
<Axes: xlabel='species', ylabel='count'>
```

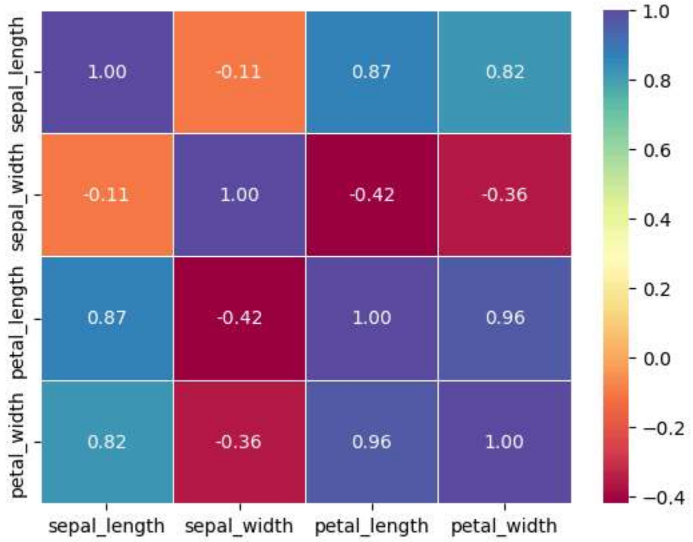



```
fig,ax=plt.subplots(2,2, figsize=(12,8))

ax[0,0].hist(iris['sepal_length'],bins=20,edgecolor='black')
ax[0,0].set_xlabel('sepal_length')
ax[0,0].set_ylabel('frequency')

ax[0,1].hist(iris['sepal_width'],bins=20,edgecolor='black',color='brown')
ax[0,1].set_xlabel('sepal_width')
ax[0,1].set_ylabel('frequency')

ax[1,0].hist(iris['petal_length'],bins=20,edgecolor='black',color='purple')
ax[1,0].set_xlabel('petal_length')
ax[1,0].set_ylabel('frequency')

ax[1,1].hist(iris['petal_width'],bins=20,edgecolor='black',color='green')
```

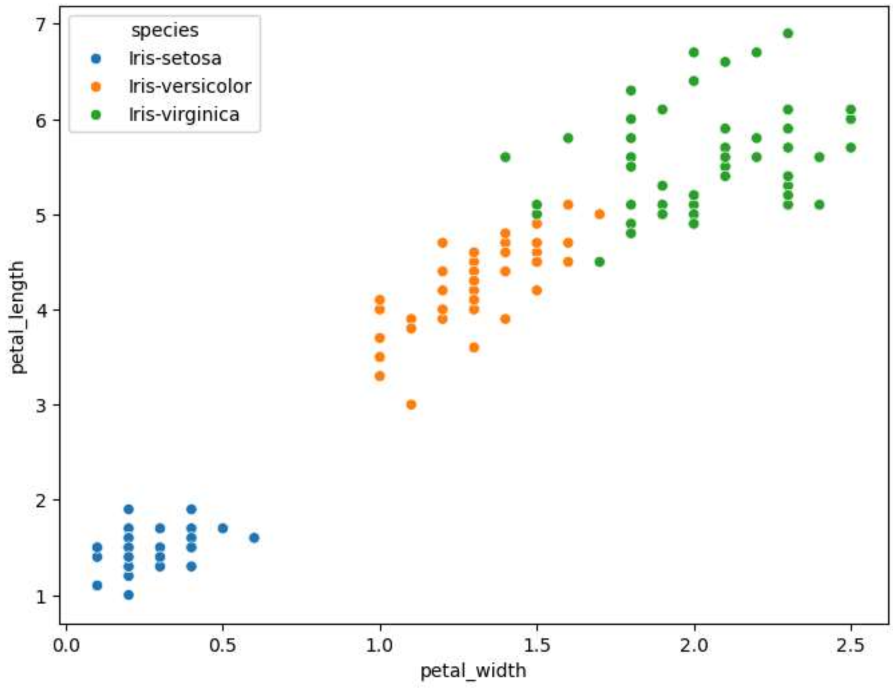
```
ax[1,1].set_xlabel('petal_width')
ax[1,1].set_ylabel('frequency')
plt.show()
```



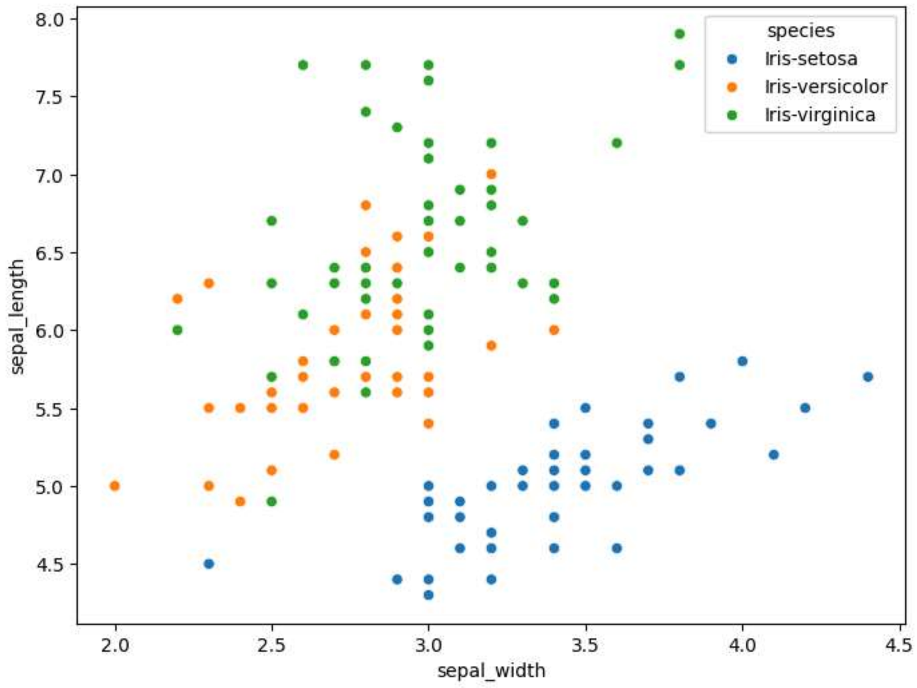

```
fig,ax=plt.subplots()
sns.heatmap(iris.drop(columns='species').corr(), annot=True,cmap='Spectral',fmt=".2f",ax=ax,linewidth=0.5)
plt.show()
```



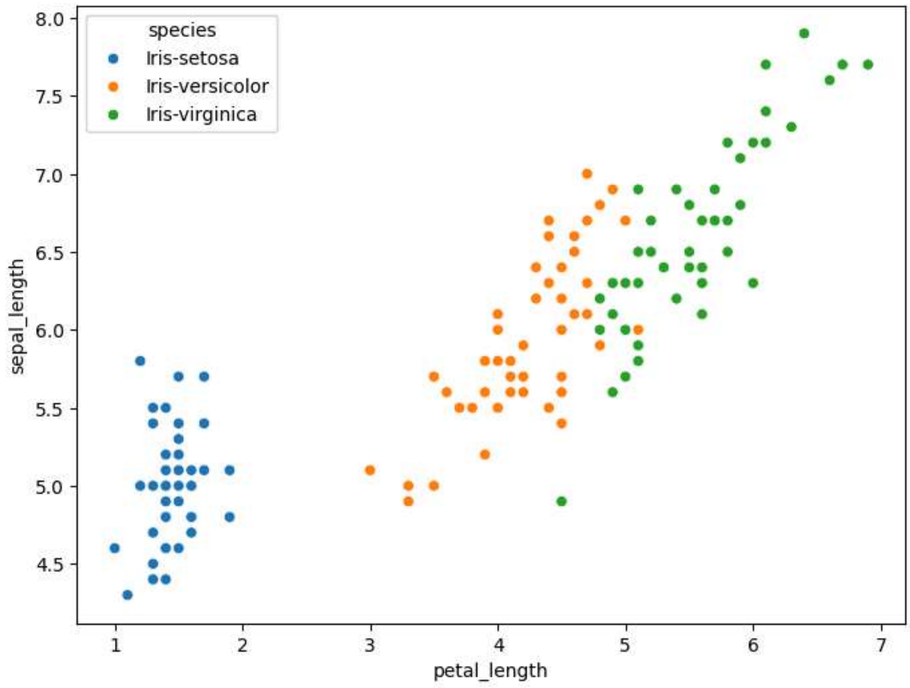

```
fig,ax=plt.subplots(figsize=(8,6))
sns.scatterplot(x="petal_width",y="petal_length",hue="species",data=iris)
plt.show()
```




```
fig,ax=plt.subplots(figsize=(8,6))
sns.scatterplot(x="sepal_width",y="sepal_length",hue="species",data=iris)
plt.show()
```

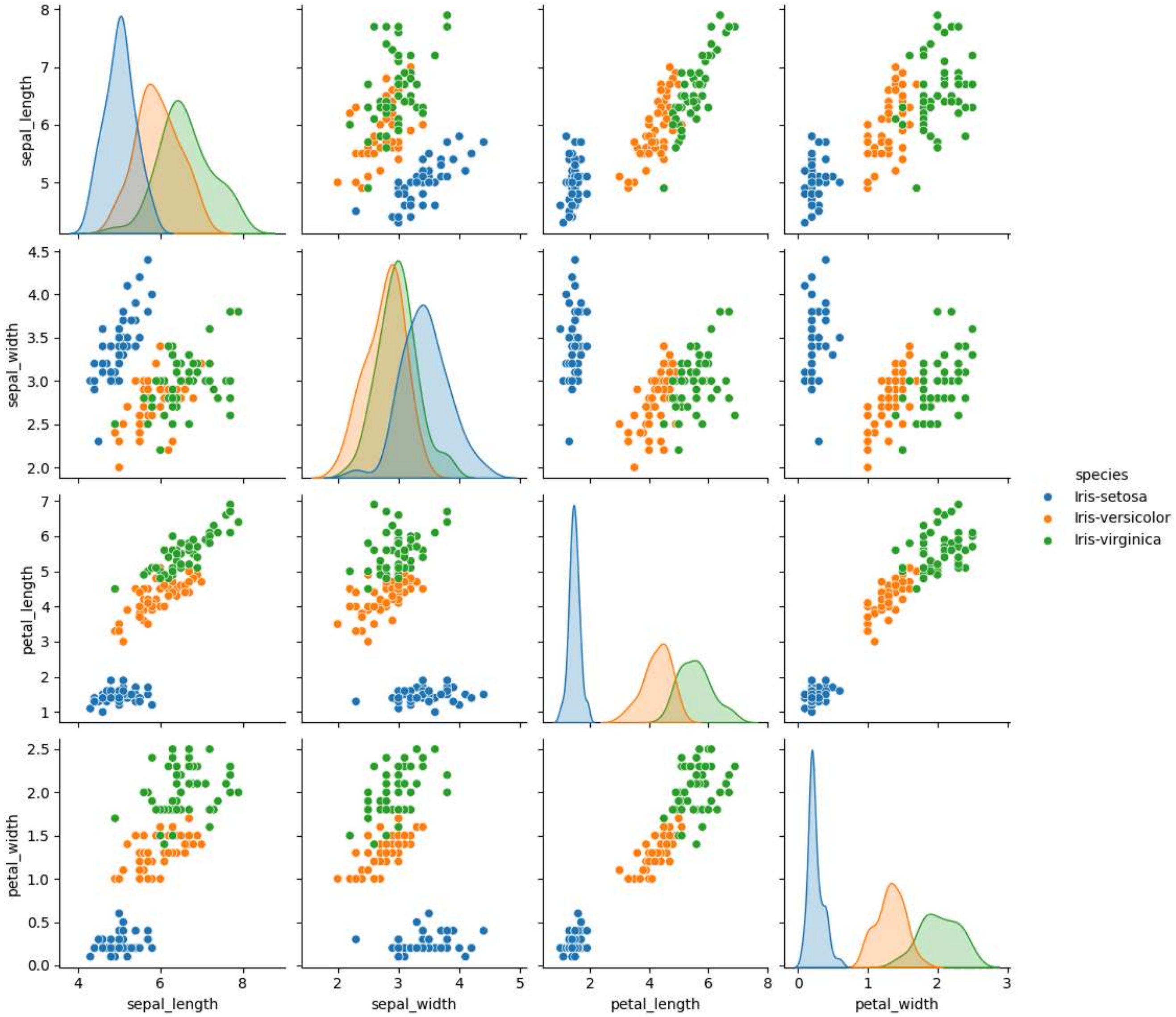



```
# Scatterplot for sepal length and petal length by species
fig,ax=plt.subplots(figsize=(8,6))
sns.scatterplot(x="petal_length",y="sepal_length",hue="species",data=iris)
plt.show()
```




```
sns.pairplot(iris, hue='species')
```

```
<seaborn.axisgrid.PairGrid at 0x7be91af42b90>
```




```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
```

```
iris['species']=label_encoder.fit_transform(iris['species'])
iris
```

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | 0 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | 0 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | 0 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | 0 |
| **...** | ... | ... | ... | ... | ... |
| **145** | 6.7 | 3.0 | 5.2 | 2.3 | 2 |
| **146** | 6.3 | 2.5 | 5.0 | 1.9 | 2 |
| **147** | 6.5 | 3.0 | 5.2 | 2.0 | 2 |
| **148** | 6.2 | 3.4 | 5.4 | 2.3 | 2 |
| **149** | 5.9 | 3.0 | 5.1 | 1.8 | 2 |

147 rows × 5 columns

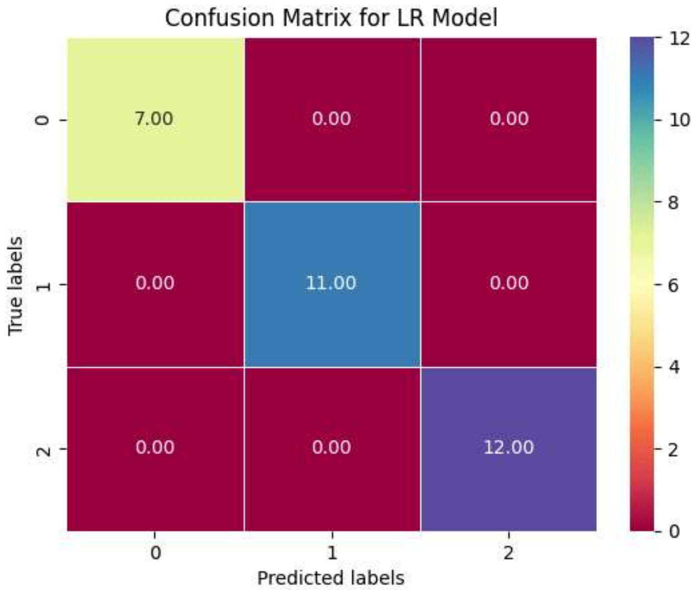
```
from sklearn.model_selection import train_test_split

# Define feature and target variables
x= iris.drop('species',axis=1)
y=iris['species']

#Split dataset into training and testing sets
x_train,x_test,y_train,y_test = train_test_split(x,y, test_size=0.2)

train_data= x_train.join(y_train)
train_data
```

| | sepal_length | sepal_width | petal_length | petal_width | species |
|---|---|---|---|---|---|
| **8** | 4.4 | 2.9 | 1.4 | 0.2 | 0 |
| **47** | 4.6 | 3.2 | 1.4 | 0.2 | 0 |
| **33** | 5.5 | 4.2 | 1.4 | 0.2 | 0 |
| **143** | 6.8 | 3.2 | 5.9 | 2.3 | 2 |
| **97** | 6.2 | 2.9 | 4.3 | 1.3 | 1 |
| **...** | ... | ... | ... | ... | ... |
| **115** | 6.4 | 3.2 | 5.3 | 2.3 | 2 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | 0 |
| **118** | 7.7 | 2.6 | 6.9 | 2.3 | 2 |
| **99** | 5.7 | 2.8 | 4.1 | 1.3 | 1 |
| **44** | 5.1 | 3.8 | 1.9 | 0.4 | 0 |

117 rows × 5 columns

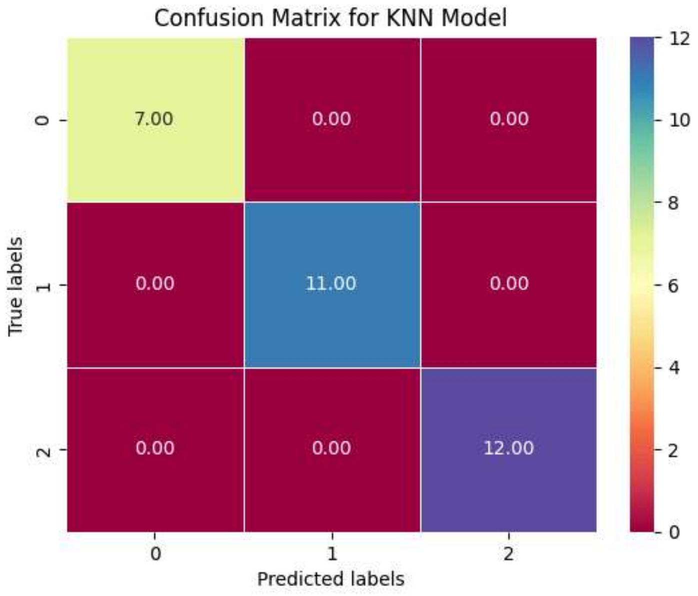
```
from sklearn.linear_model import LogisticRegression

#Build a Logisitc Regression model
fitted_model_lr = LogisticRegression()
#Train the model
fitted_model_lr.fit(x_train,y_train)
#Make predictions
y_pred_lr = fitted_model_lr.predict(x_test)
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Model's performance evaluating
accuracy = accuracy_score(y_test, y_pred_lr)
print(f' Accuracy for LR: {accuracy:.4f}')
print(classification_report(y_test, y_pred_lr))
cf=confusion_matrix(y_test,y_pred_lr)
sns.heatmap(cf,annot=True,fmt=".2f",cmap="Spectral",linewidth=0.5)
plt.title('Confusion Matrix for LR Model')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:460: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
 Accuracy for LR: 1.0000
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        12

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```




```
from sklearn.neighbors import KNeighborsClassifier

#Build a KNN model
fitted_model_knn = KNeighborsClassifier()
#Train the model
fitted_model_knn.fit(x_train,y_train)
#Make predictions
y_pred_knn = fitted_model_knn.predict(x_test)
# Model's performance evaluating
accuracy1 = accuracy_score(y_test, y_pred_knn)
print(f' Accuracy for KNN: {accuracy1:.4f}')
print(classification_report(y_test, y_pred_knn))
cf1=confusion_matrix(y_test,y_pred_knn)
sns.heatmap(cf1,annot=True,fmt=".2f",cmap="Spectral",linewidth=0.5)
plt.title('Confusion Matrix for KNN Model')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
Accuracy for KNN: 1.0000
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        12

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```




```
from sklearn.svm import SVC

#Build a KNN model
fitted_model_svm = SVC()
#Train the model
fitted_model_svm.fit(x_train,y_train)
#Make predictions
y_pred_svm = fitted_model_svm.predict(x_test)
# Model's performance evaluating
accuracy2 = accuracy_score(y_test, y_pred_svm)
print(f' Accuracy for SVM: {accuracy2:.4f}')
print(classification_report(y_test, y_pred_svm))
cf2=confusion_matrix(y_test,y_pred_svm)
sns.heatmap(cf2,annot=True,fmt=".2f",cmap="Spectral",linewidth=0.5)
plt.title('Confusion Matrix for SVM Model')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
Accuracy for SVM: 1.0000
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        12

    accuracv                           1.00        30
```

```
from sklearn.tree import DecisionTreeClassifier

#Build a KNN model
fitted_model_dt = DecisionTreeClassifier()
#Train the model
fitted_model_dt.fit(x_train,y_train)
#Make predictions
y_pred_dt = fitted_model_dt.predict(x_test)
# Model's performance evaluating
accuracy3 = accuracy_score(y_test, y_pred_dt)
print(f' Accuracy for DT: {accuracy3:.4f}')
print(classification_report(y_test, y_pred_dt))
cf3=confusion_matrix(y_test,y_pred_dt)
sns.heatmap(cf3,annot=True,fmt=".2f",cmap="Spectral",linewidth=0.5)
plt.title('Confusion Matrix for DT Model')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.show()
```

```
Accuracy for DT: 1.0000
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      1.00      1.00        11
           2       1.00      1.00      1.00        12

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```