In [1]:

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

In [2]:

```python
df=pd.read_csv("iris.data",header=None,names=["Sepal_length","Sepal_width"
df
```

Out[2]:

	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

150 rows × 5 columns

Exploring the data

In [3]:

```python
df.head() # display 1st five rows
```

Out[3]:

	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

In [4]:
```
df.tail(10) # display last 10 rows
```

Out[4]:

	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

In [5]:
```
df.head(10)
```

Out[5]:

	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

Getting Information about the Dataset

In [6]:
```
df.shape
```

Out[6]:
```
(150, 5)
```

In [7]:
```
df.info()# information about the columns and its datatypes
```

```
<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
```

In [8]:
```
df.describe() #  function gives a good picture of the distribution of data
```

Out[8]:

	Sepal_length	Sepal_width	Petal_length	Petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Checking Missing Values

In [9]:
```
df.isna()
```

Out[9]:

	Sepal_length	Sepal_width	Petal_length	Petal_width	Species
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
145	False	False	False	False	False
146	False	False	False	False	False
147	False	False	False	False	False
148	False	False	False	False	False
149	False	False	False	False	False

150 rows × 5 columns

In [10]:
```
df.isna().sum() # check if our data contains any missing values or not.
```

Out[10]:
```
Sepal_length    0
Sepal_width     0
Petal_length    0
Petal_width     0
Species         0
dtype: int64
```

In [11]:
```
df["Species"].value_counts() # no missing values
```

Out[11]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: Species, dtype: int64
```

In [12]:
```
# took input data as feature,output data as target
feature=df.iloc[:,:-1]
target=df["Species"]
```

```
In [13]: feature
```

Out[13]:

	Sepal_length	Sepal_width	Petal_length	Petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

```
In [14]: target
```

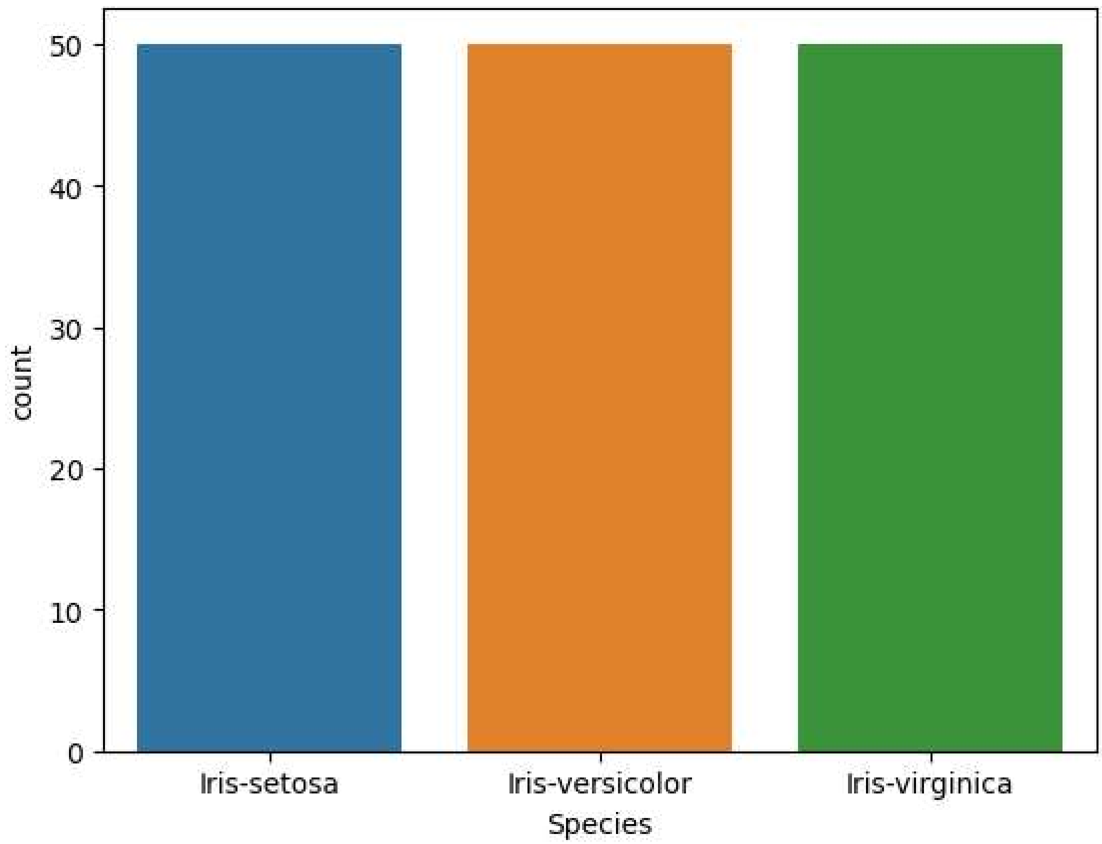
Out[14]:

```
0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
             ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 150, dtype: object
```

Data Visualization

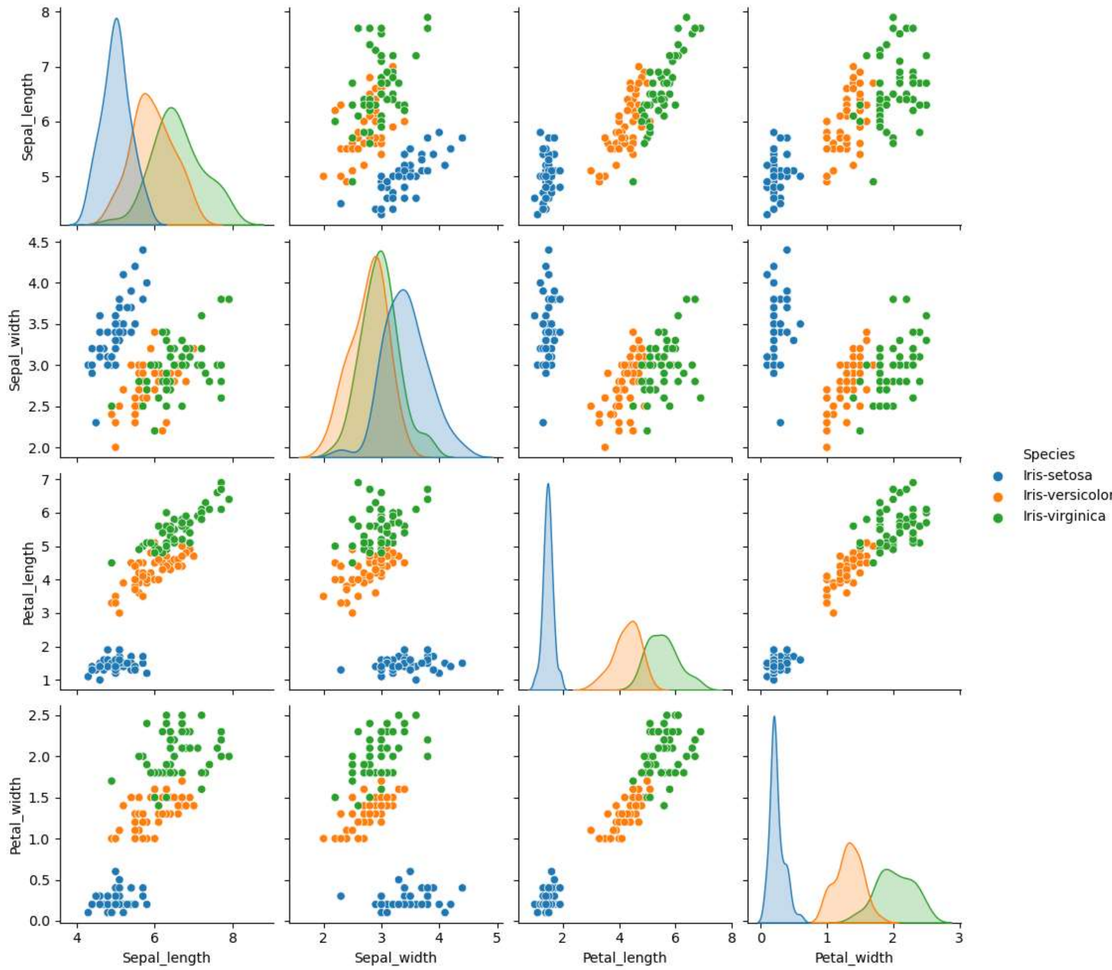
In [15]:

```
sns.countplot(x="Species",data=df);
```



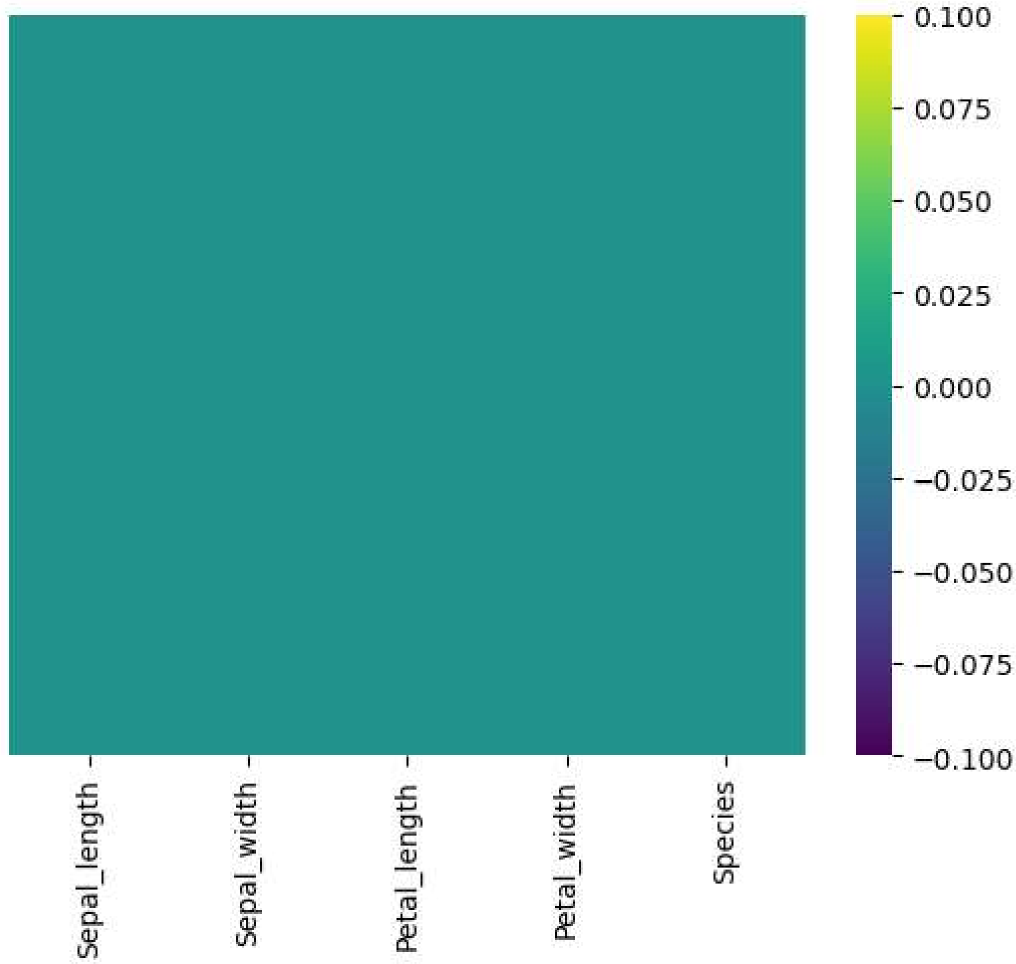
In [16]:

```
sns.pairplot(data=df,hue="Species");
```

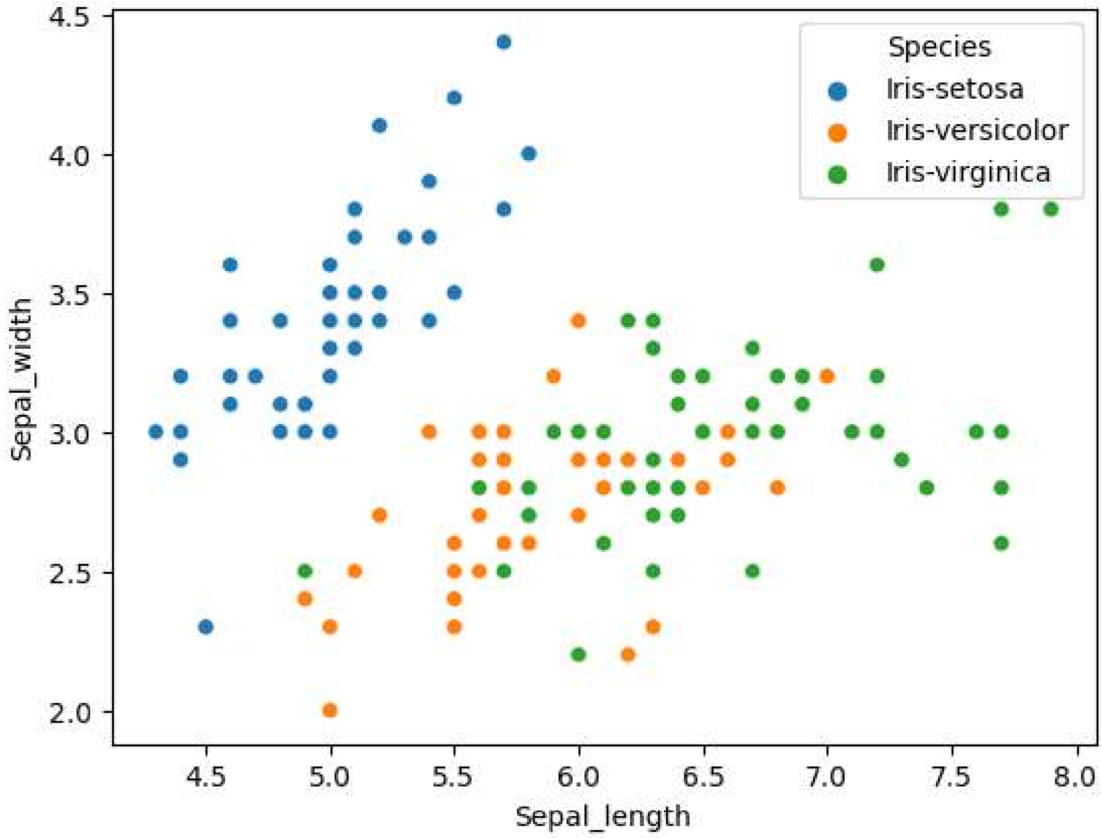


In [17]:

```
sns.heatmap(df.isnull(),yticklabels=False,cbar=True,cmap="viridis");
```



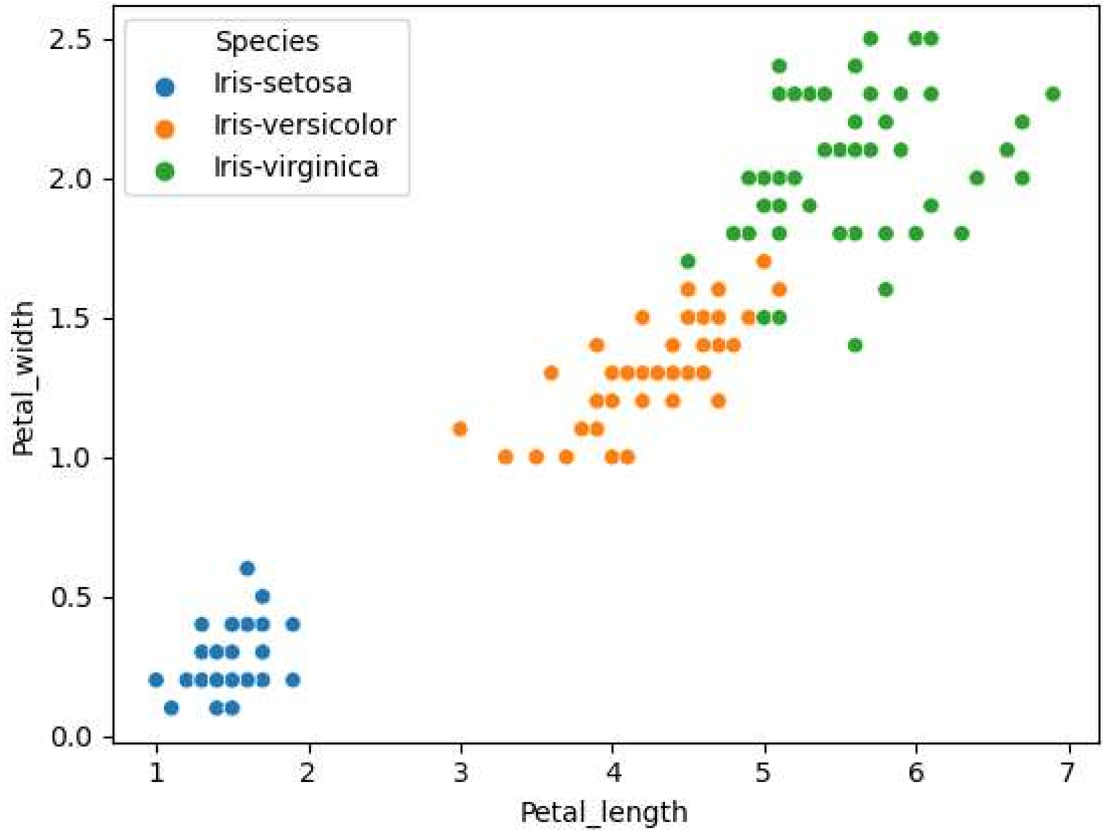
In [18]:
```
sns.scatterplot(x='Sepal_length', y='Sepal_width',hue='Species', data=df);
```



from fig:-

Setosa has smaller sepal lengths but larger sepal widths. versicolor lies in the middle,in the case of sepal length and sepal width. virginica has larger sepal length and smaller sepal width.

In [19]:
```
sns.scatterplot(x='Petal_length', y='Petal_width',hue='Species', data=df);
```



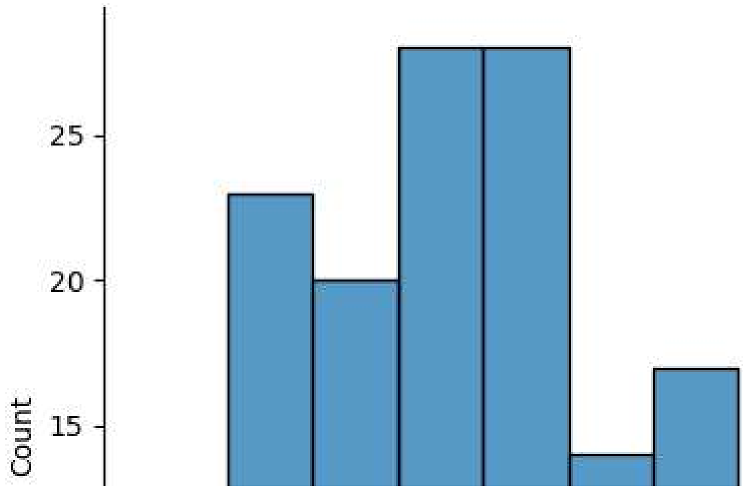
from fig:-

Setosa has smaller petal lengths and widths. Versicolor lies in the middle of the other two species in terms of petal length and width Virginica has the largest of petal lengths and widths.

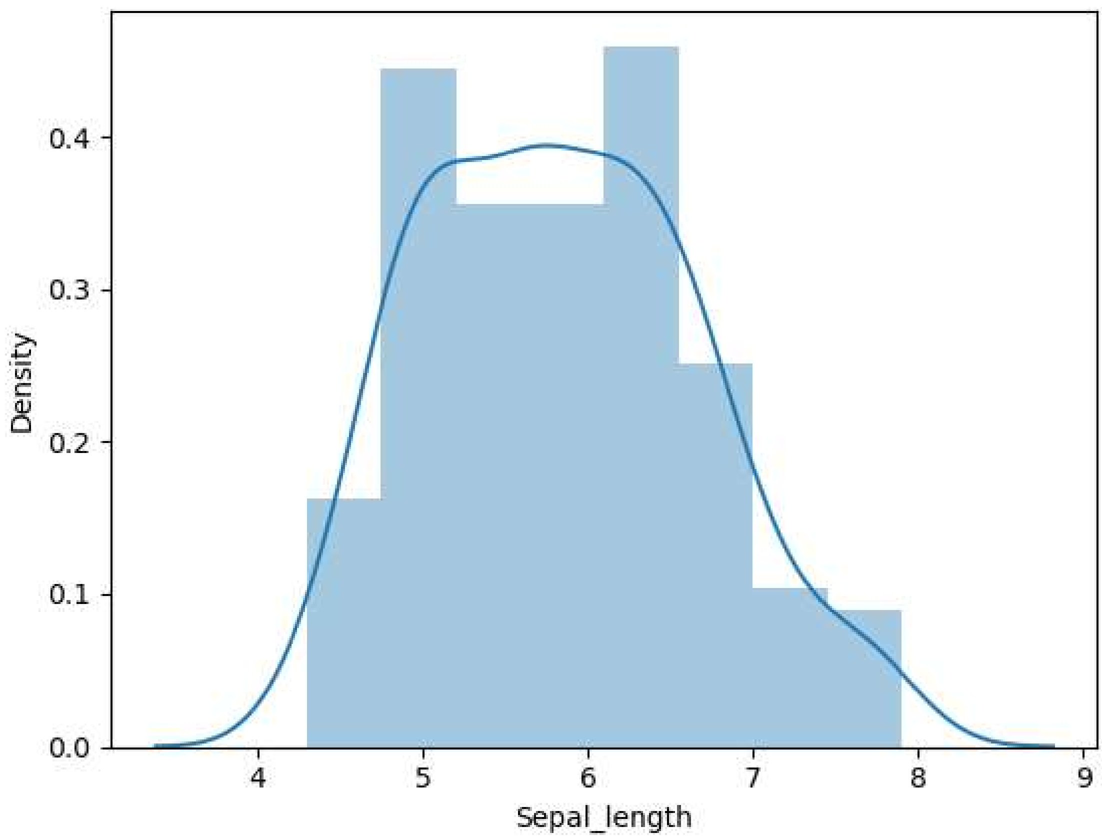
In [20]:
```
colname=feature.columns

for i in feature[colname]:
    print(i)
    sns.displot(feature[i]);
#sns.distplot(df["Sepal_length"]);
```

```
Sepal_length
Sepal_width
Petal_length
Petal_width
```



In [21]:
```
sns.distplot(df["Sepal_length"]);
```

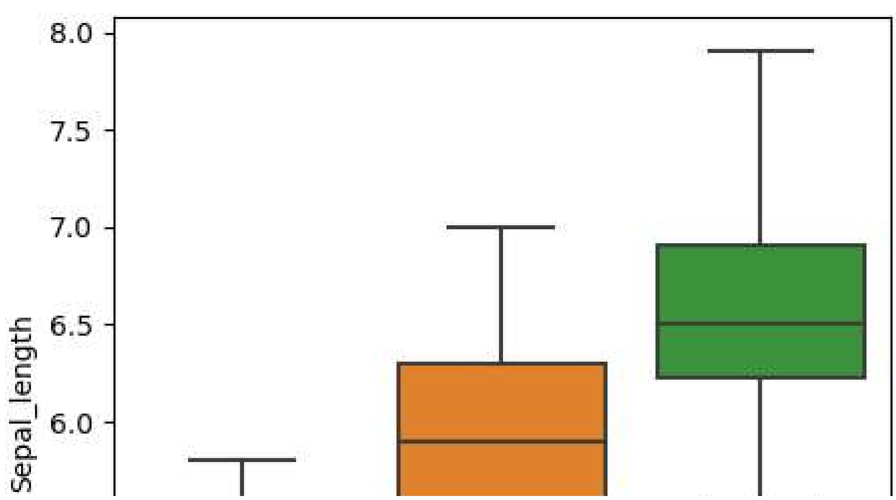


In [22]:

```
col=feature.columns

for i in feature[col]:
    print(i)
    plt.figure(figsize=(5,5))
    sns.boxplot(x="Species", y=i, data=df);
```

```
Sepal_length
Sepal_width
Petal_length
Petal_width
```



In [23]:

```
df.describe()
```

Out[23]:

	Sepal_length	Sepal_width	Petal_length	Petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [24]:

```
df.skew()
```

Out[24]:

```
Sepal_length    0.314911
Sepal_width     0.334053
Petal_length   -0.274464
Petal_width    -0.104997
dtype: float64
```

In [25]:
```
feature=df.iloc[:,:-1]
target=df["Species"]
```

In [26]:
```
feature
```

Out[26]:

	Sepal_length	Sepal_width	Petal_length	Petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

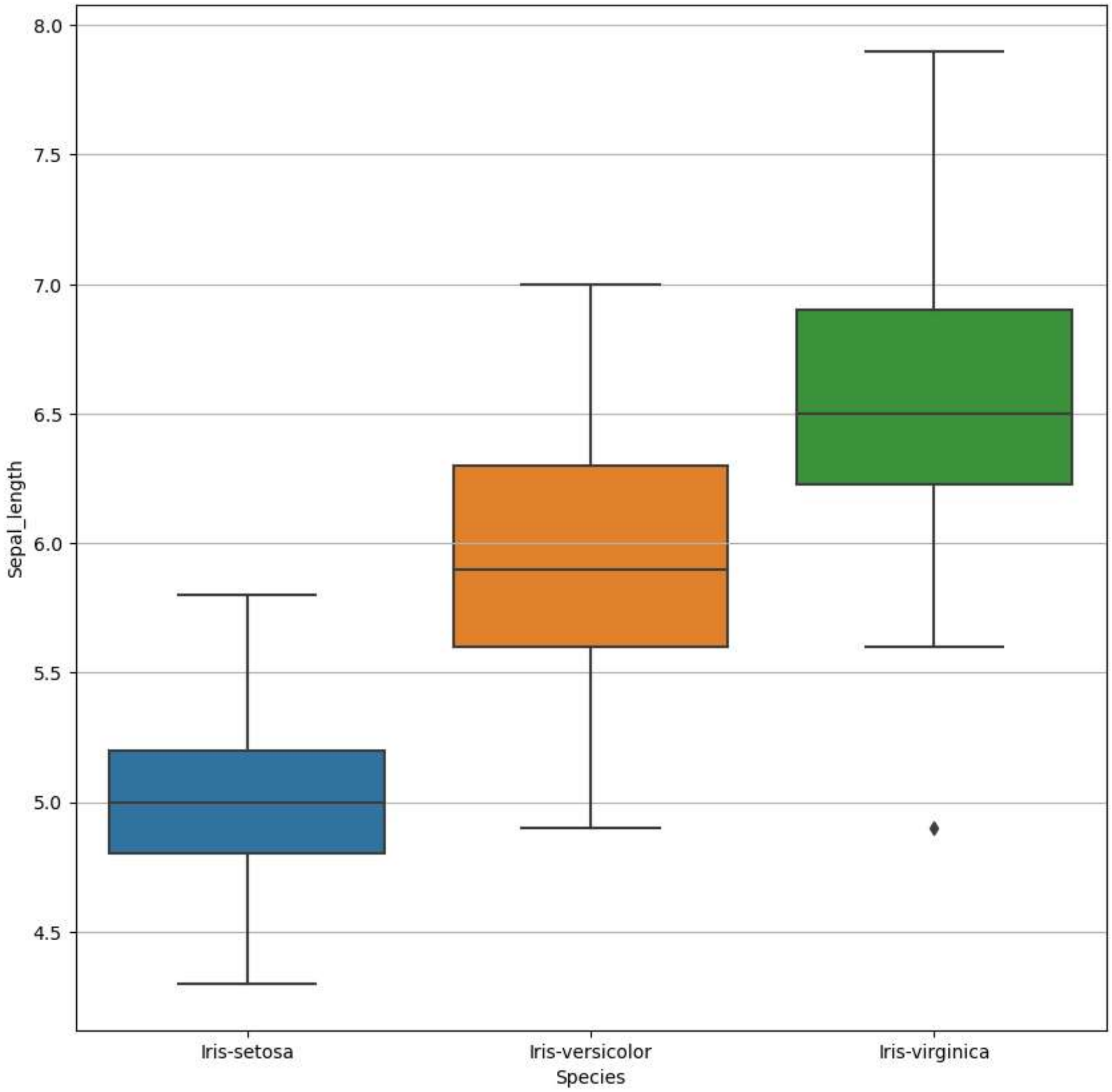
In [27]:
```
target
```

Out[27]:
```
0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
             ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 150, dtype: object
```

Handling Outliers

In [28]:

```
plt.figure(figsize=(10,10))
plt.grid()
sns.boxplot(data=feature,x=target,y="Sepal_length");
```



In [29]:

```
df[(df["Species"]=="Iris-virginica")&(df["Sepal_length"]<5.0)]
```

Out[29]:

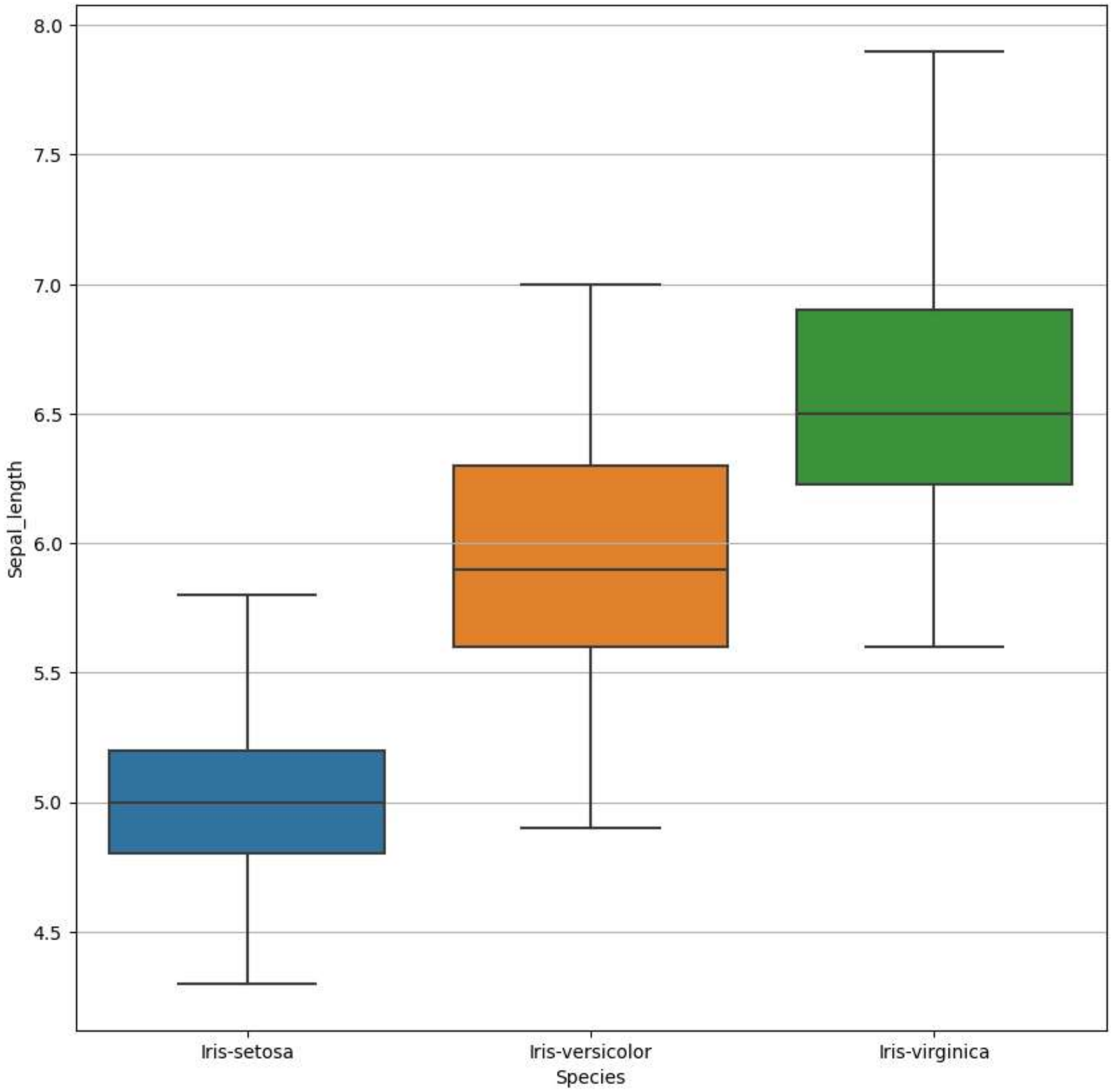
	Sepal_length	Sepal_width	Petal_length	Petal_width	Species
106	4.9	2.5	4.5	1.7	Iris-virginica

In [30]:

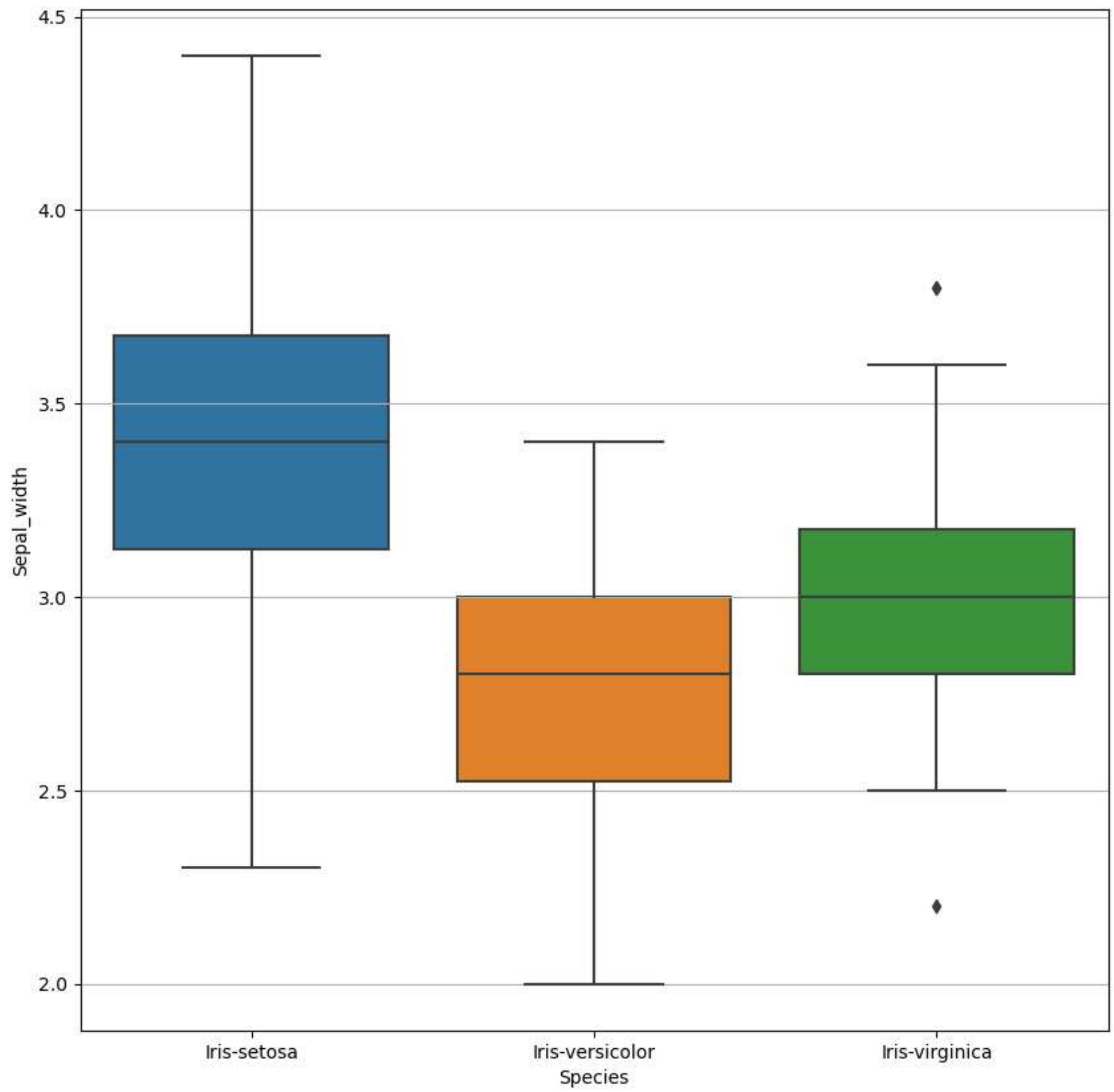
```
feature.loc[106,"Sepal_length"]=5.7
```

In [31]:

```
plt.figure(figsize=(10,10))
plt.grid()
sns.boxplot(data=feature,x=target,y="Sepal_length");
```



In [32]:
```
plt.figure(figsize=(10,10))
plt.grid()
sns.boxplot(data=feature,x=target,y="Sepal_width");
```



In [33]:
```
df[(df["Species"]=="Iris-virginica")&(df["Sepal_width"]<2.5)]
```

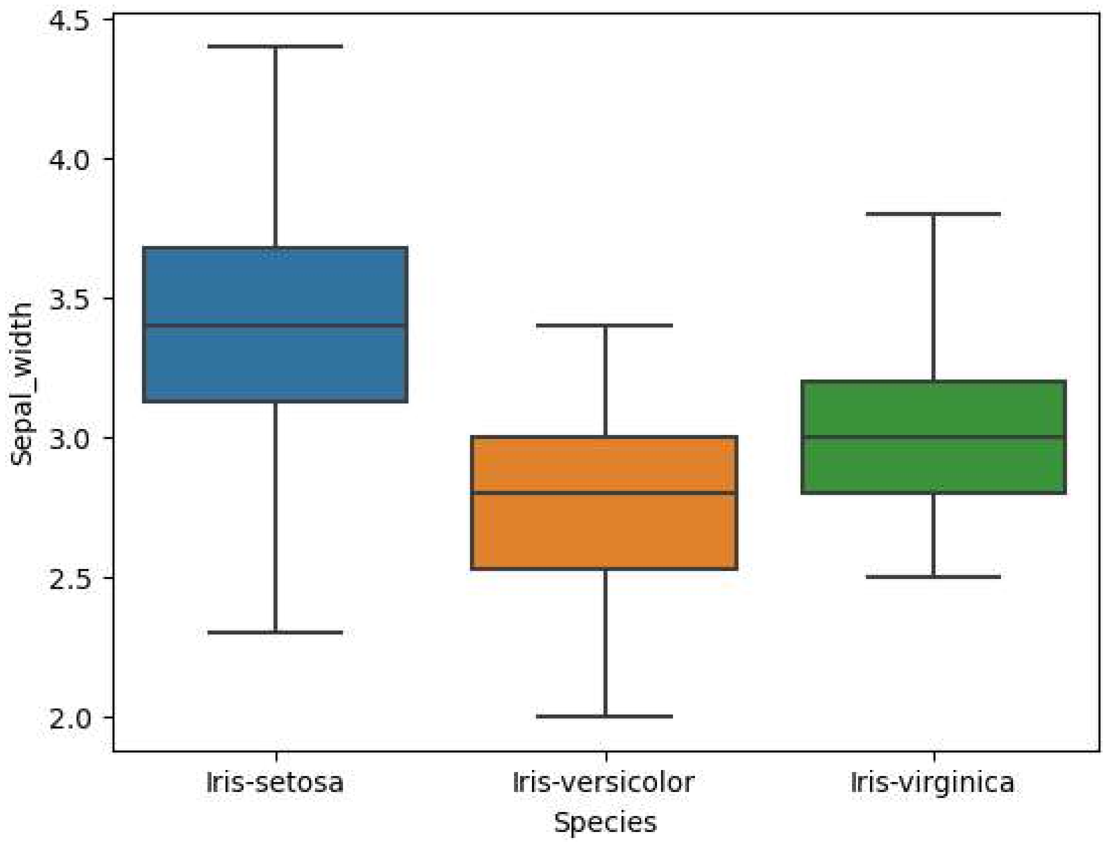
Out[33]:

	Sepal_length	Sepal_width	Petal_length	Petal_width	Species
119	6.0	2.2	5.0	1.5	Iris-virginica

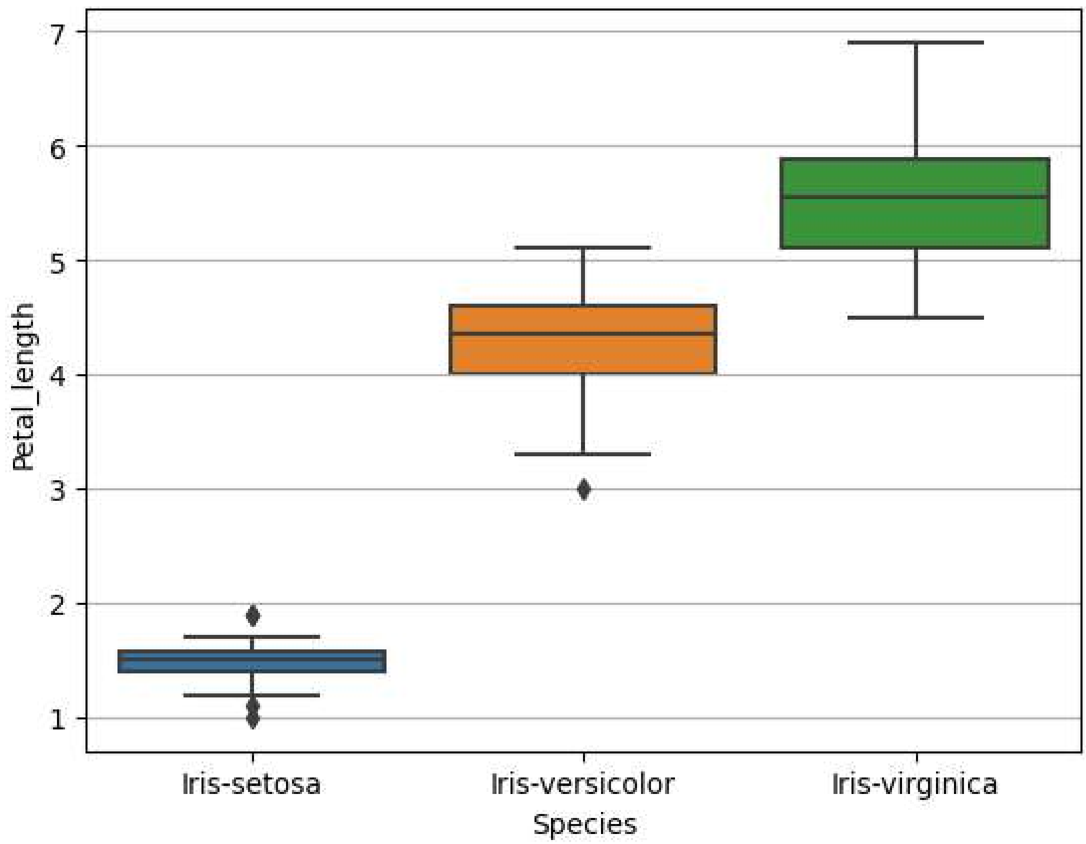
In [34]:
```
feature.loc[119,"Sepal_width"]=3.6
```

In [35]:

```
sns.boxplot(data=feature,x=target,y="Sepal_width");
```



In [36]:
```
plt.grid()
sns.boxplot(data=feature,x=target,y="Petal_length");
```



In [37]:
```
df[(df["Species"]=="Iris-versicolor")&(df["Petal_length"]<3.2)]
```

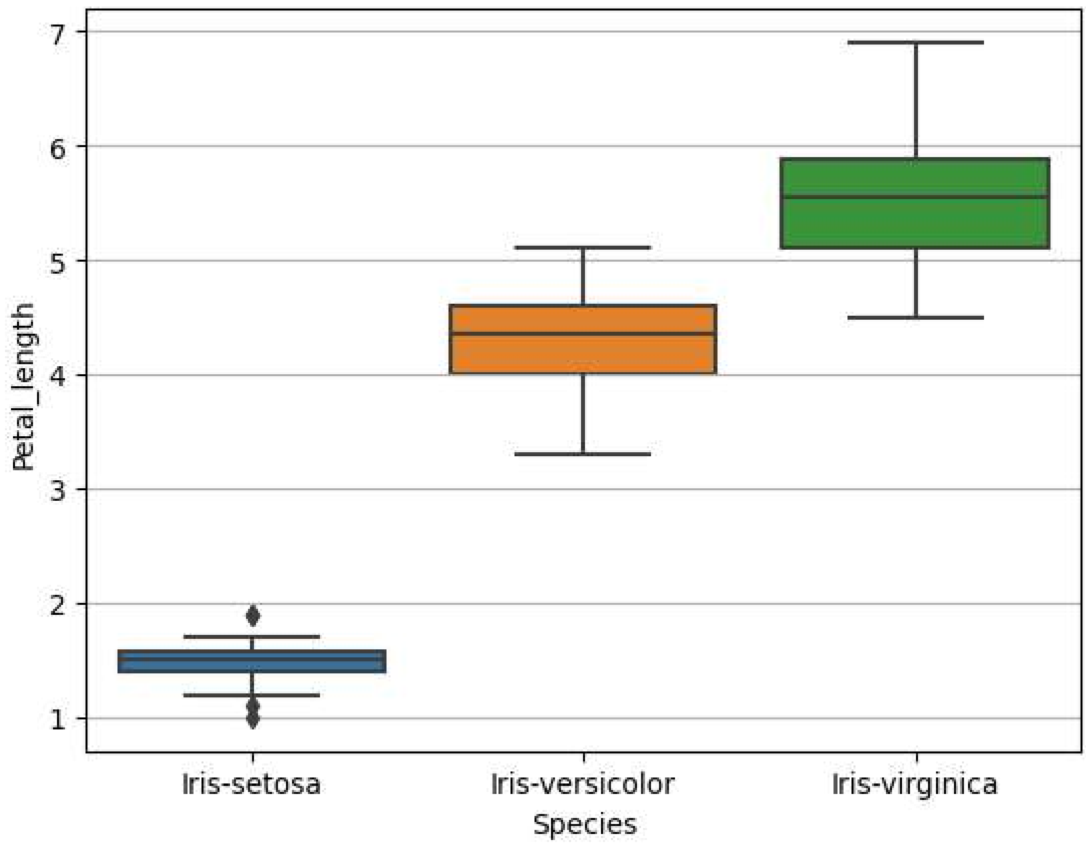
Out[37]:

	Sepal_length	Sepal_width	Petal_length	Petal_width	Species
98	5.1	2.5	3.0	1.1	Iris-versicolor

In [38]:
```
feature.loc[98,"Petal_length"]=3.3
```

In [39]:

```
plt.grid()
sns.boxplot(data=feature,x=target,y="Petal_length");
```



In [40]:

```
df[(df["Species"]=="Iris-setosa")&(df["Petal_length"]<1.2)]
```

Out[40]:

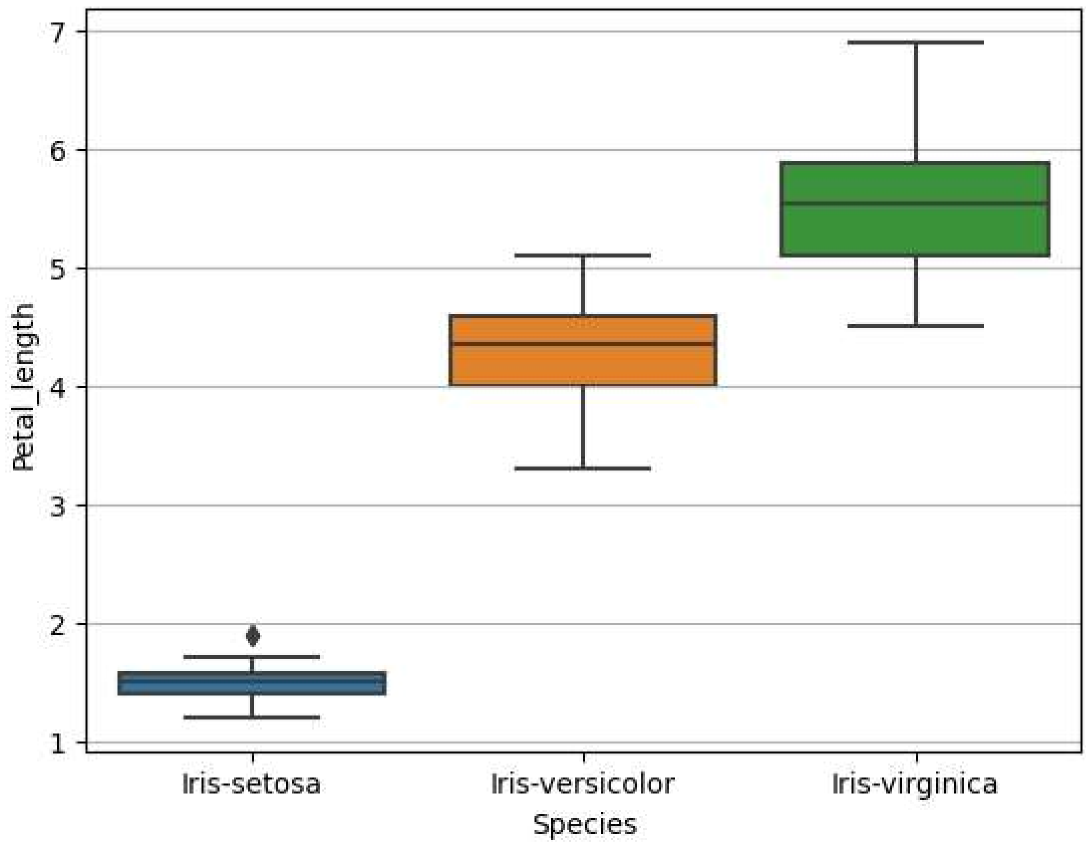
	Sepal_length	Sepal_width	Petal_length	Petal_width	Species
13	4.3	3.0	1.1	0.1	Iris-setosa
22	4.6	3.6	1.0	0.2	Iris-setosa

In [41]:

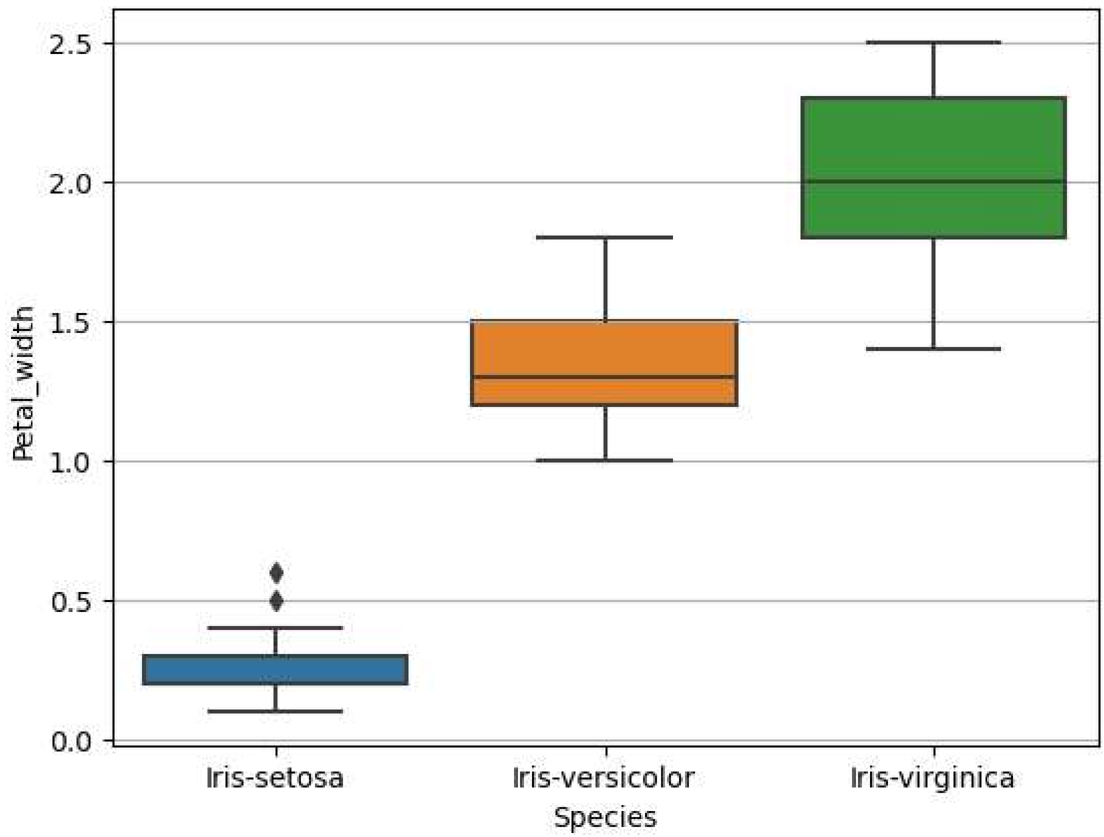
```
feature.loc[[13,22],"Petal_length"]=1.2
```

In [42]:

```
plt.grid()
sns.boxplot(data=feature,x=target,y="Petal_length");
```



In [43]:
```
plt.grid()
sns.boxplot(data=feature,x=target,y="Petal_width");
```



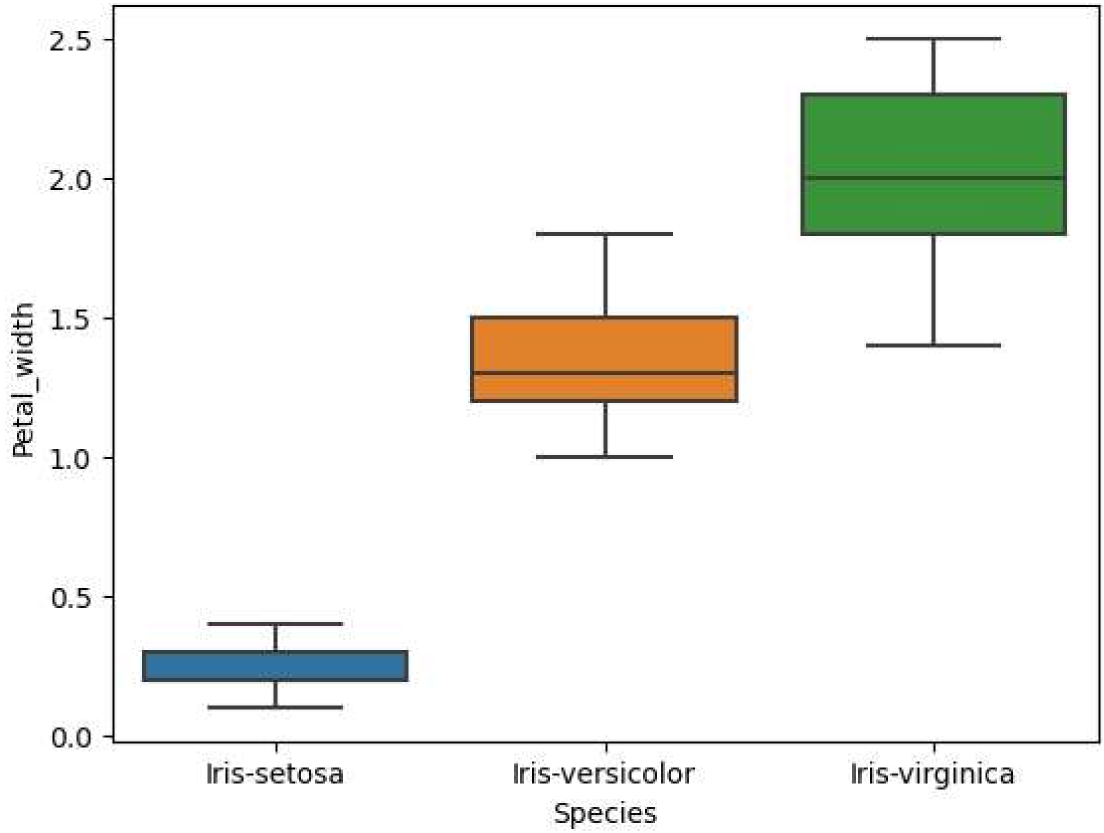
In [44]:
```
df[(df["Species"]=="Iris-setosa")&(df["Petal_width"]>0.4)]
```

Out[44]:

	Sepal_length	Sepal_width	Petal_length	Petal_width	Species
23	5.1	3.3	1.7	0.5	Iris-setosa
43	5.0	3.5	1.6	0.6	Iris-setosa

In [45]:
```
feature.loc[[23,43],"Petal_width"]=0.4
```

In [46]:
```
sns.boxplot(data=feature,x=target,y="Petal_width");
```



In [47]:
```
feature.head()
```

Out[47]:

	Sepal_length	Sepal_width	Petal_length	Petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Encoding

Target

In [48]:
```
from sklearn.preprocessing import LabelEncoder
```

In [49]:
```
le=LabelEncoder()
le.fit_transform(target)
```

Out[49]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Train and fit the model

In [50]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(feature,target,test_size=0.2,ra
```

In [51]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(xtrain, ytrain)
ypred = knn.predict(xtest)
```

In [52]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classificati

ac = accuracy_score(ytest, ypred)
cm = confusion_matrix(ytest, ypred)
cr = classification_report(ytest, ypred)

print(f"Accuracy -: {ac}\n{cm}\n\n{cr}")
```

```
Accuracy -: 0.9666666666666667
[[11  0  0]
 [ 0 12  1]
 [ 0  0  6]]

                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      0.92      0.96        13
 Iris-virginica       0.86      1.00      0.92         6

       accuracy                           0.97        30
      macro avg       0.95      0.97      0.96        30
   weighted avg       0.97      0.97      0.97        30
```

In [53]:

```
# KNN :-Training score and testing score
trainacc = knn.score(xtrain, ytrain)
testacc = knn.score(xtest, ytest)

print(f"Training Accuracy -: {trainacc}\nTesting Accuracy -: {testacc}")
```

```
Training Accuracy -: 0.95
Testing Accuracy -: 0.9666666666666667
```

In [54]:

```
final_k=[]
for i in range(1,31):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(xtrain, ytrain)
    pred=knn.predict(xtest)
    k=accuracy_score(ytest,pred,normalize=True)*float(100)
    final_k.append(k)
    print('\n ytest accuracy for k=%d is %d'%(i,k))
```

```
ytest accuracy for k=1 is 100

ytest accuracy for k=2 is 96

ytest accuracy for k=3 is 96

ytest accuracy for k=4 is 100

ytest accuracy for k=5 is 96

ytest accuracy for k=6 is 100

ytest accuracy for k=7 is 100

ytest accuracy for k=8 is 100

ytest accuracy for k=9 is 100

ytest accuracy for k=10 is 100

ytest accuracy for k=11 is 100

ytest accuracy for k=12 is 100

ytest accuracy for k=13 is 100

ytest accuracy for k=14 is 100

ytest accuracy for k=15 is 100

ytest accuracy for k=16 is 100

ytest accuracy for k=17 is 100

ytest accuracy for k=18 is 100

ytest accuracy for k=19 is 100

ytest accuracy for k=20 is 100

ytest accuracy for k=21 is 100

ytest accuracy for k=22 is 100

ytest accuracy for k=23 is 100

ytest accuracy for k=24 is 100

ytest accuracy for k=25 is 100

ytest accuracy for k=26 is 96

ytest accuracy for k=27 is 93

ytest accuracy for k=28 is 93
```

```
ytest accuracy for k=29 is 93

ytest accuracy for k=30 is 96
```

In [55]:

```
from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier(n_neighbors=1)

knn.fit(xtrain,ytrain)

ypred=knn.predict(xtest)
```

In [56]:

```
acc=accuracy_score(ytest,ypred)
cm=confusion_matrix(ytest,ypred)
cr=classification_report(ytest,ypred)

print(f"Accuracy :- {acc}\n{cm}\n{cr}")
```

```
Accuracy :- 1.0
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

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

Function for selecting model

In [57]:

```
def mymodel(model):
    #model creation
    model.fit(xtrain, ytrain)
    ypred = model.predict(xtest)

    #checking bias & variance
    train = model.score(xtrain, ytrain)
    test = model.score(xtest, ytest)
    print(f"Training Accuracy : {train}\nTesting Accuracy : {test}\n\n")

    #model evaluation
    print(classification_report(ytest, ypred))
    return model
```

Using Decision Tree .....

In [58]:
```python
from sklearn.tree import DecisionTreeClassifier
```

In [59]:
```python
decision_tree = mymodel(DecisionTreeClassifier())
```

```
Training Accuracy : 1.0
Testing Accuracy : 1.0


                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

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

In [ ]: