In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
sns.set()
import matplotlib.pyplot as  plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df = pd.read_csv('IRIS.csv')
```

In [3]:
```
df.head()
```

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()
```

Out[4]:

	sepal_length	sepal_width	petal_length	petal_width	species
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

Basic information

In [5]:
```
df.shape
```

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

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

Out[6]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

In [7]:
```
df.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
```

In [8]:
```
df.describe().style.background_gradient(cmap='Reds')
```

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

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

Out[9]:
```
0      False
1      False
2      False
3      False
4      False
       ...  
145    False
146    False
147    False
148    False
149    False
Length: 150, dtype: bool
```

In [10]:
```
df.isnull().sum()
```

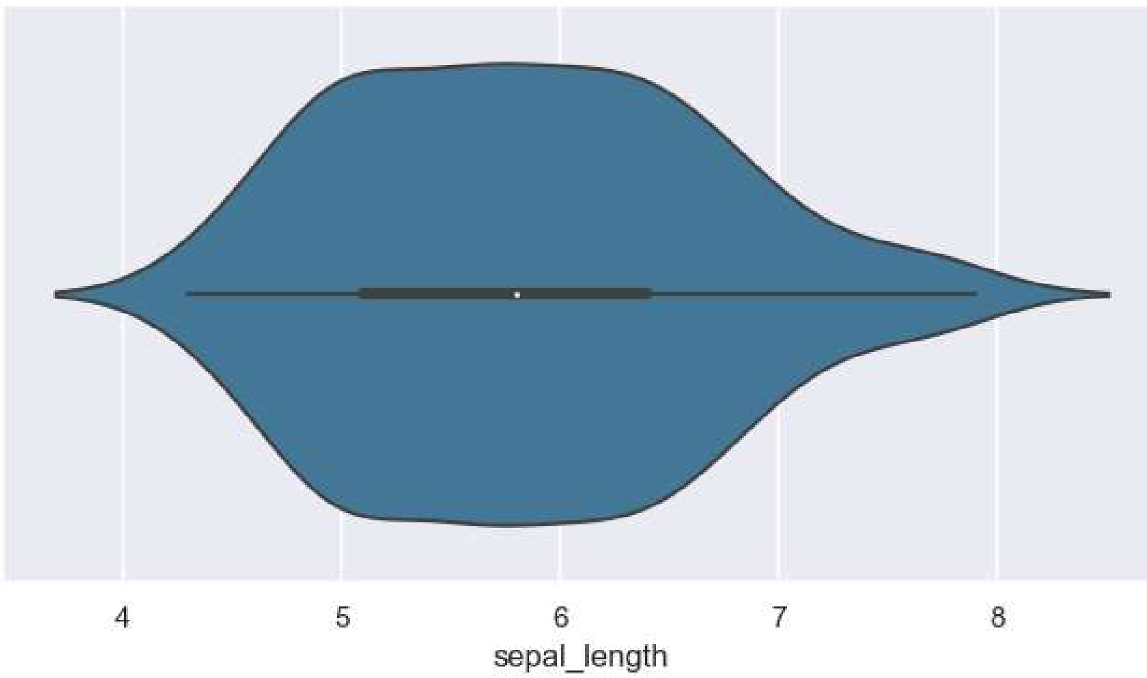
Out[10]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

In [11]:
```
df.isnull().sum()/len(df)*100
```

Out[11]:
```
sepal_length    0.0
sepal_width     0.0
petal_length    0.0
petal_width     0.0
species         0.0
dtype: float64
```

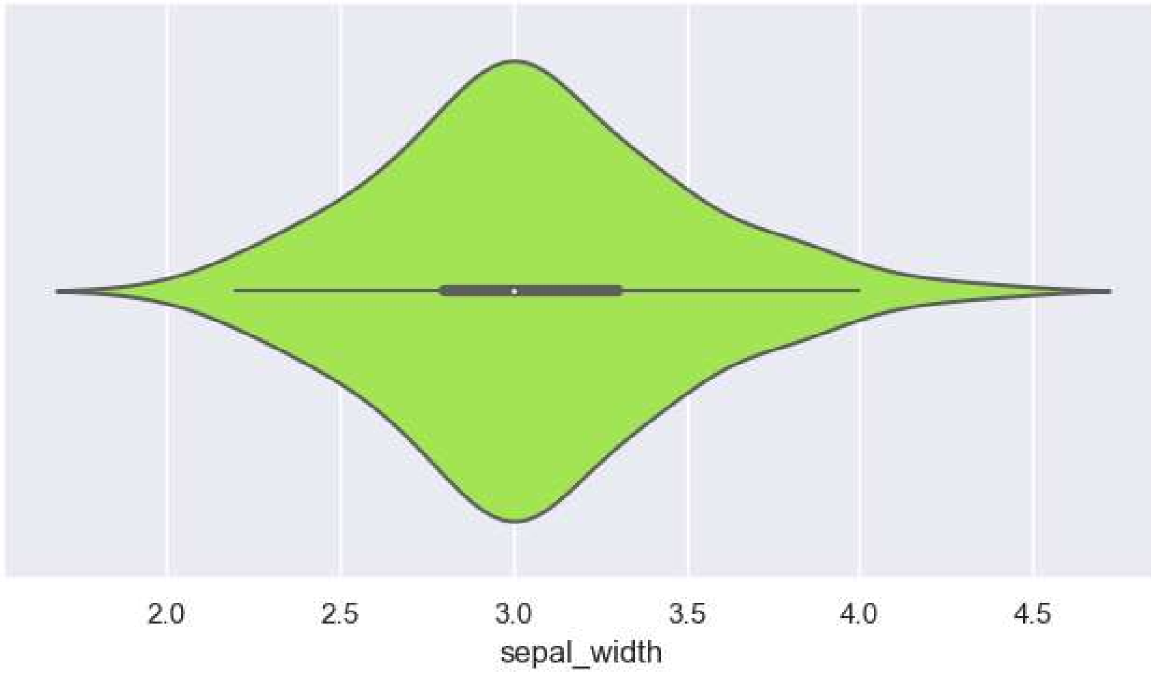
In [12]:
```
plt.figure(figsize=(8,4))
sns.violinplot(x=df["sepal_length"],palette='mako')
```

Out[12]:
```
<Axes: xlabel='sepal_length'>
```



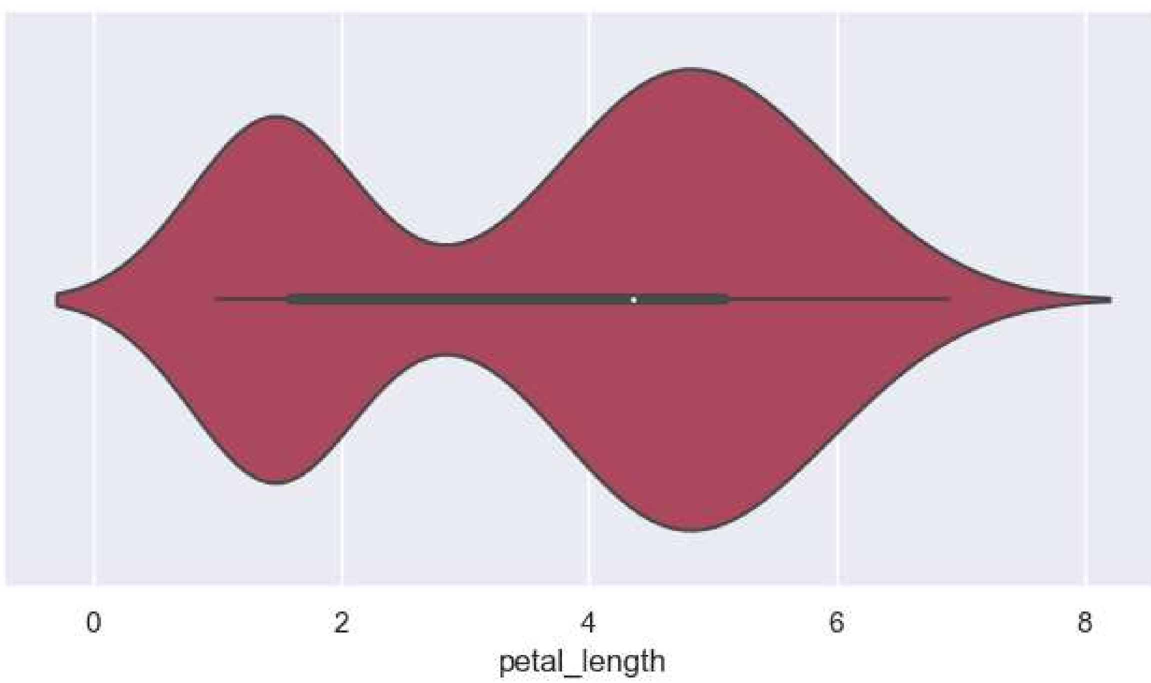
In [13]:
```
plt.figure(figsize=(8,4))
sns.violinplot(x=df["sepal_width"],palette='turbo')
```

Out[13]: <Axes: xlabel='sepal_width'>



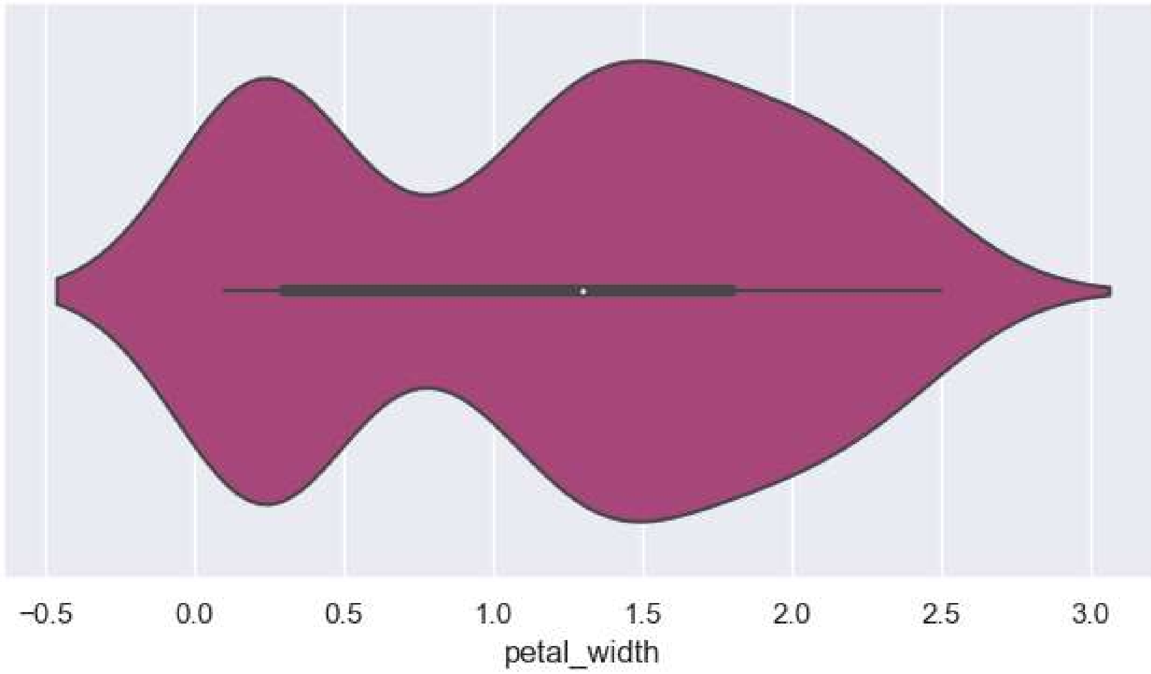
In [14]:
```
plt.figure(figsize=(8,4))
sns.violinplot(x=df["petal_length"],palette='inferno')
```

Out[14]: <Axes: xlabel='petal_length'>



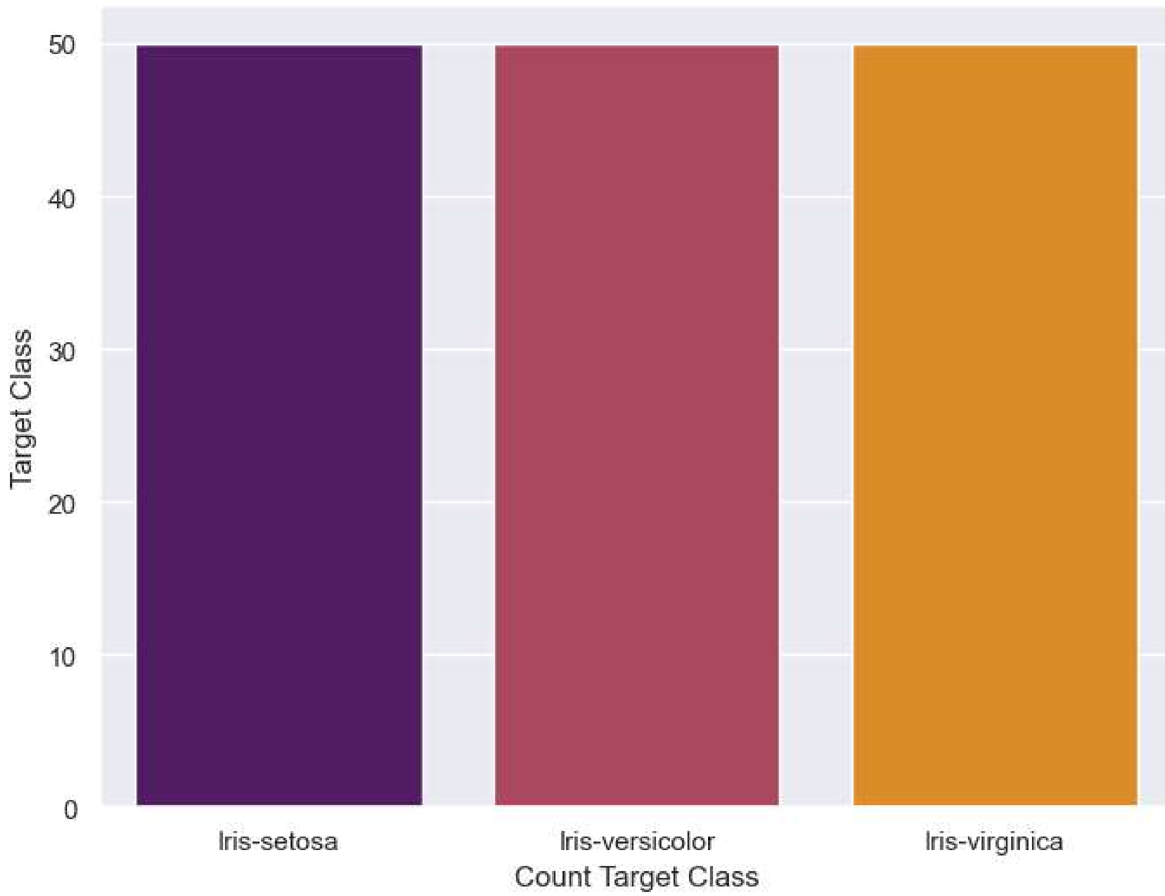
In [15]:
```
plt.figure(figsize=(8,4))
sns.violinplot(x=df["petal_width"],palette = 'magma')
```

Out[15]:
```
<Axes: xlabel='petal_width'>
```

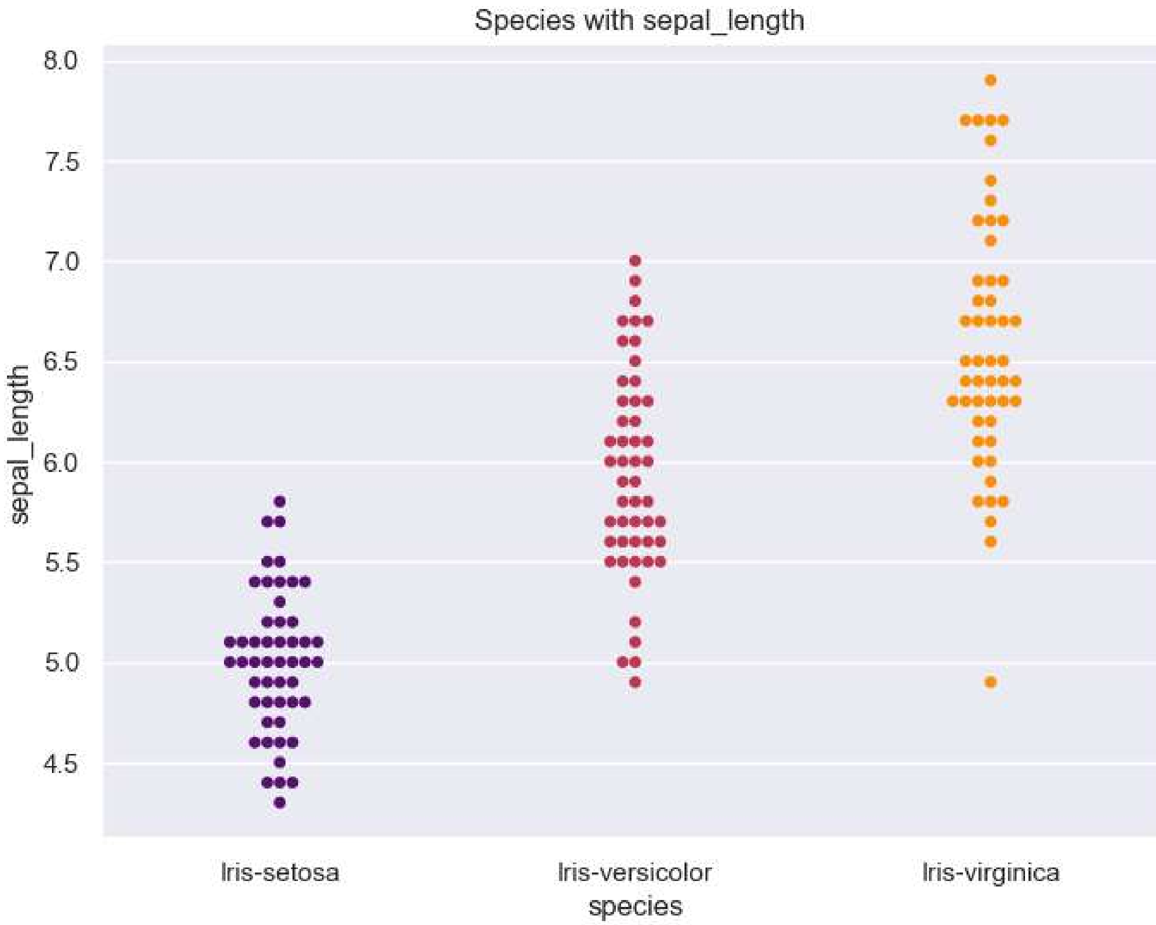


In [16]:
```
plt.figure(figsize=(8,6))
sns.countplot(df, x="species",palette='inferno')
plt.xlabel('Count Target Class')
plt.ylabel('Target Class')
```

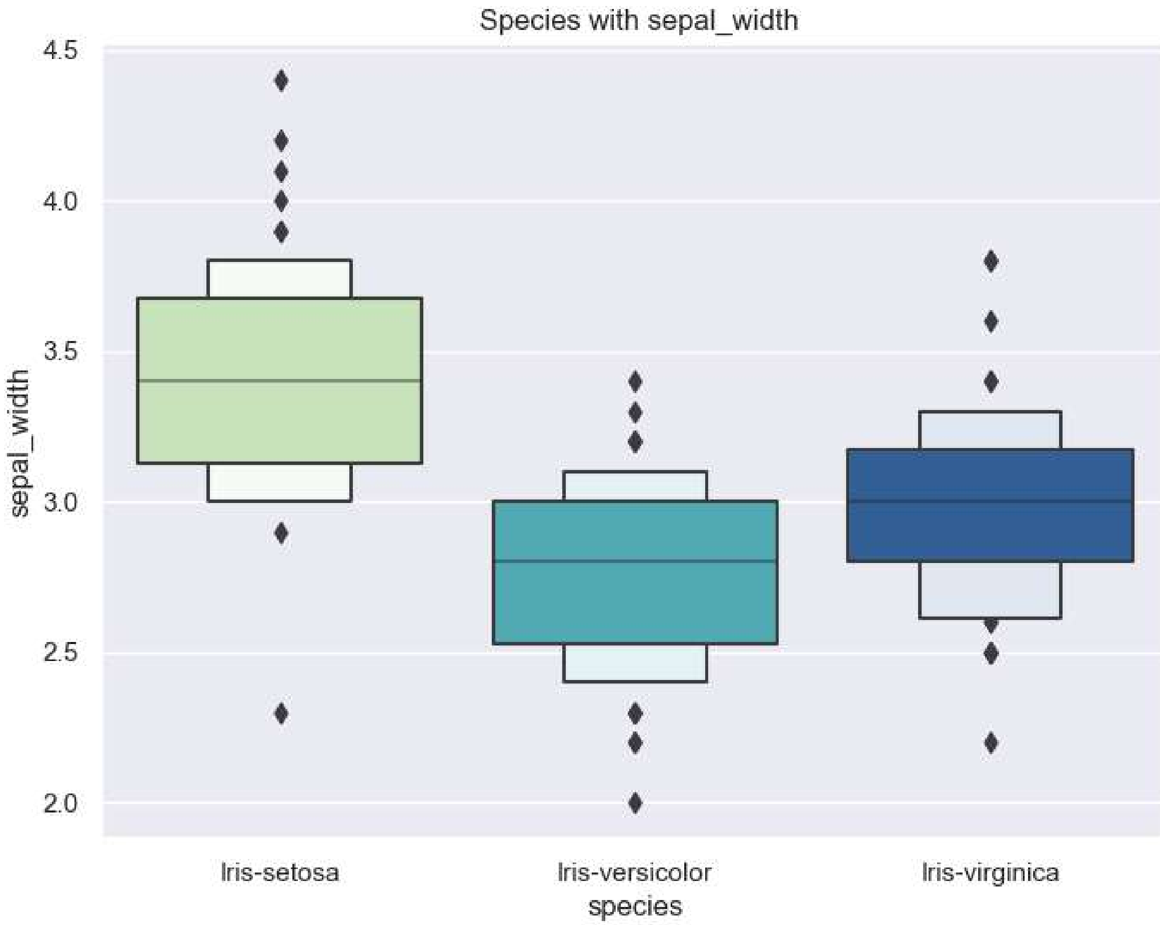
Out[16]:
```
Text(0, 0.5, 'Target Class')
```



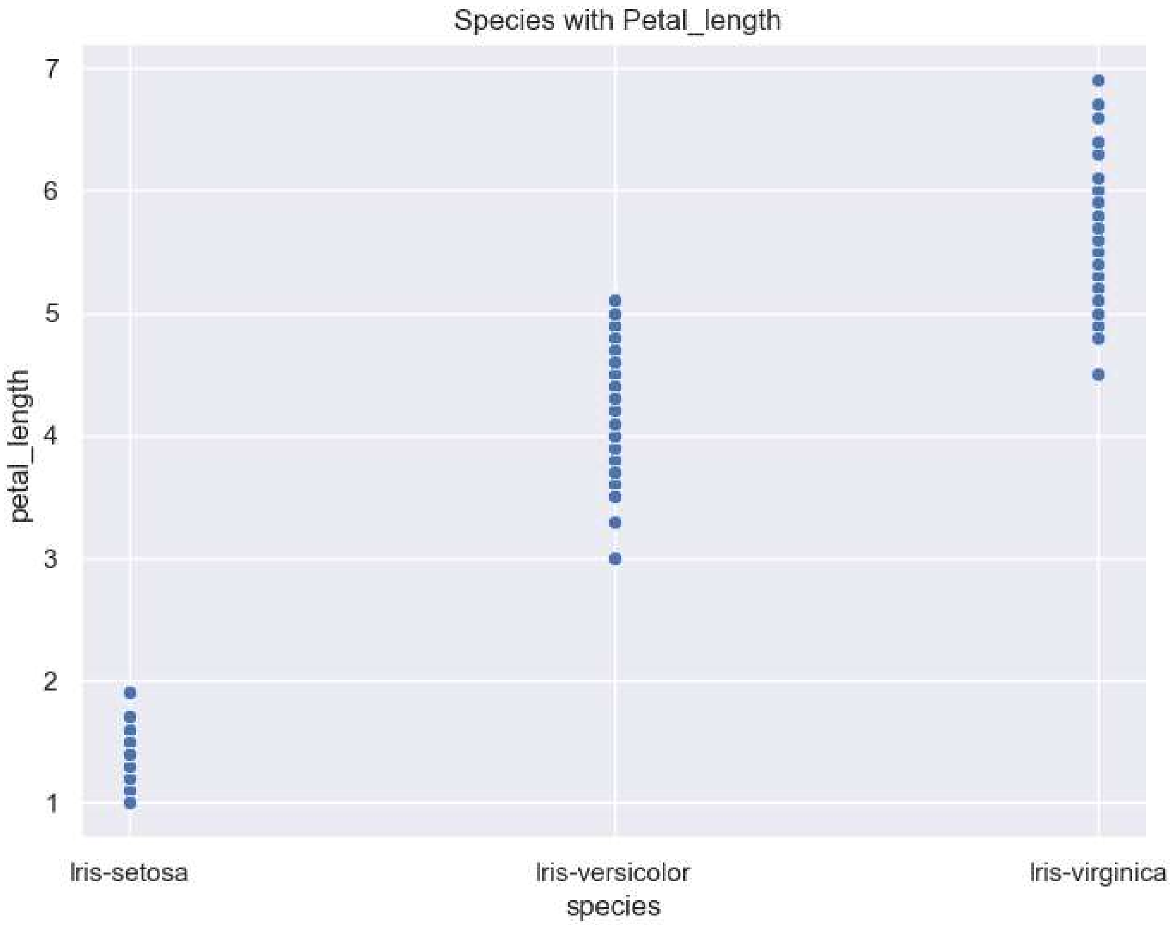
In [17]:
```
plt.figure(figsize=(8,6))
sns.swarmplot(data=df, x="species", y="sepal_length", hue=None,palette='inferno
plt.title(' Species with sepal_length')
plt.show()
```



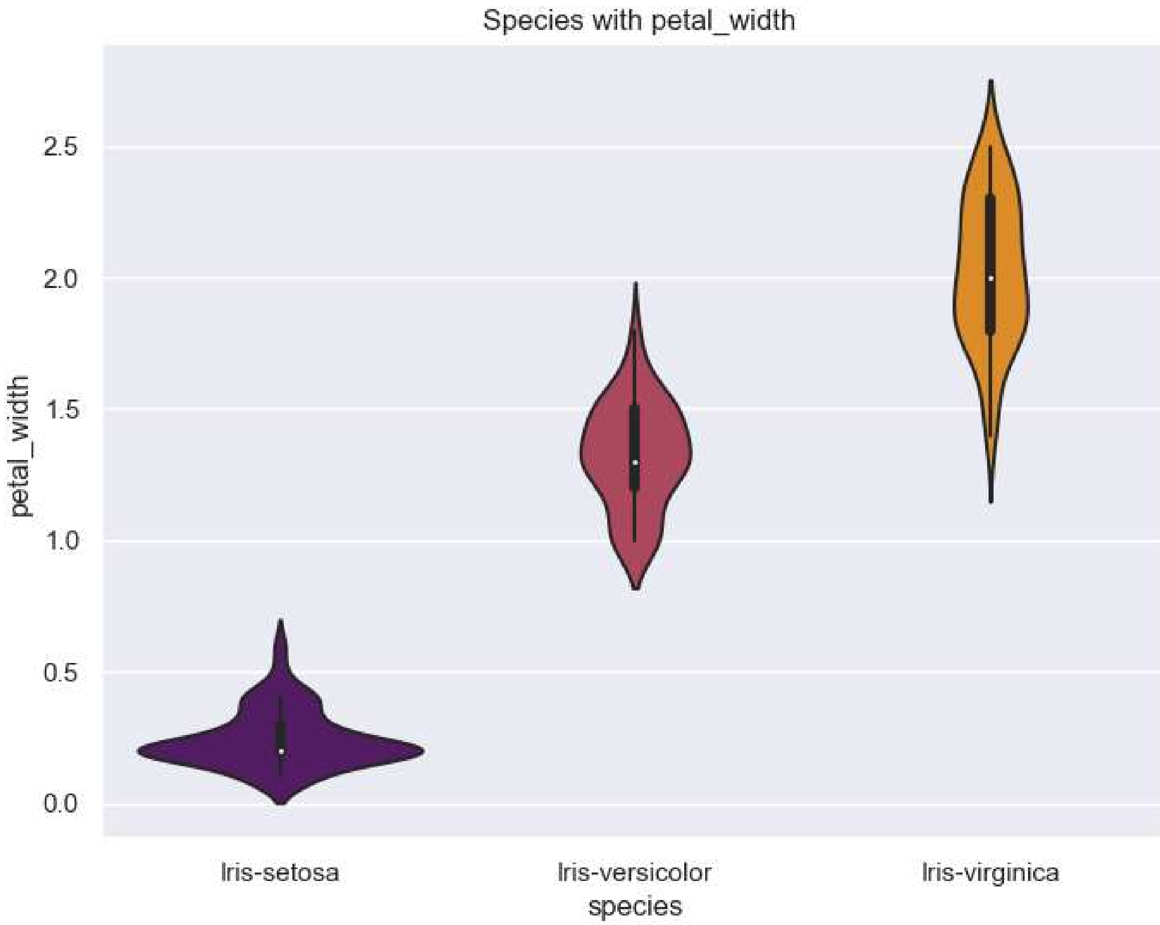
In [18]:
```
plt.figure(figsize=(8,6))
sns.boxenplot(data=df, x="species", y="sepal_width", hue=None, palette='YlGnBu
plt.title(' Species with sepal_width')
plt.show()
```



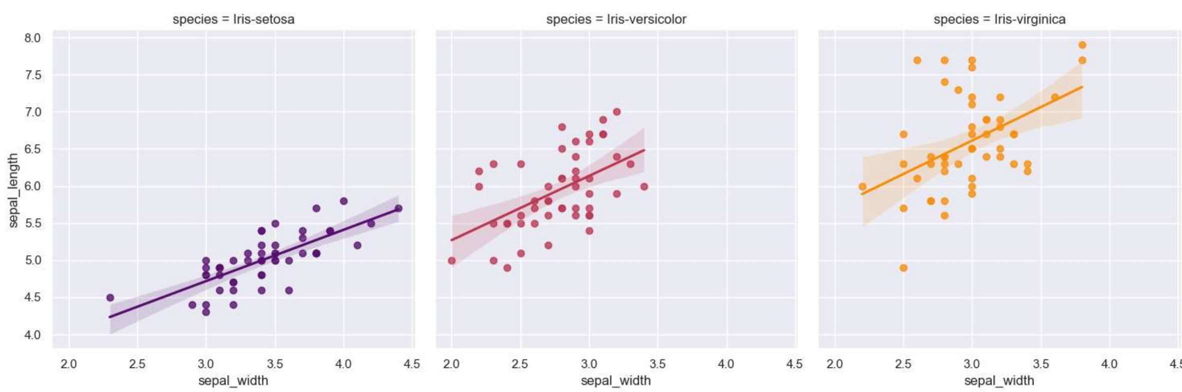
In [19]:
```
plt.figure(figsize=(8,6))
sns.scatterplot(data=df, x="species", y="petal_length", hue=None,palette = 'tu
plt.title(' Species with Petal_length')
plt.show()
```



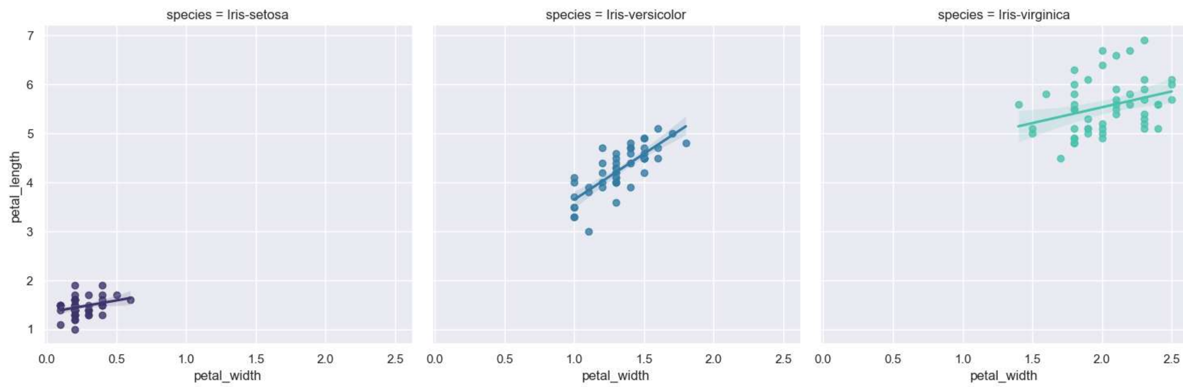
In [20]:
```
plt.figure(figsize=(8,6))
sns.violinplot(x='species', y='petal_width', data=df, palette='inferno')
plt.title(' Species with petal_width')
plt.show()
```



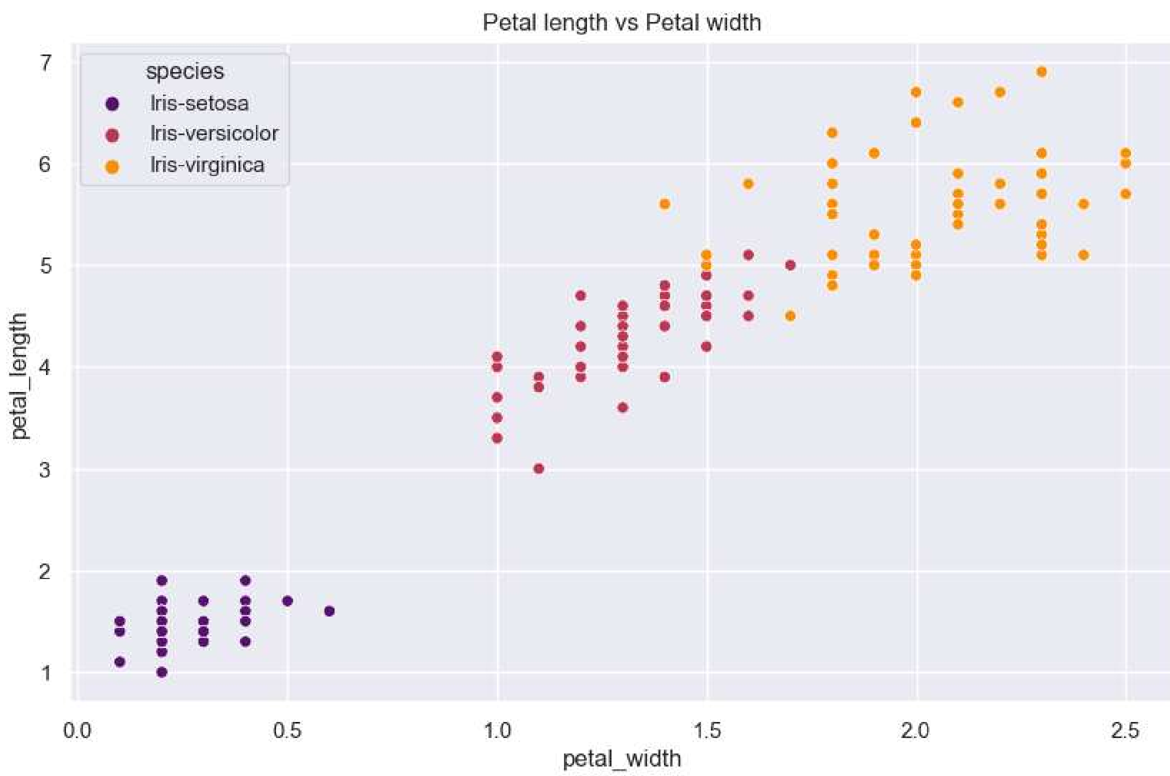
In [21]:
```
sns.lmplot(x = 'sepal_width', y = 'sepal_length', data = df, col = 'species',
plt.show()
```



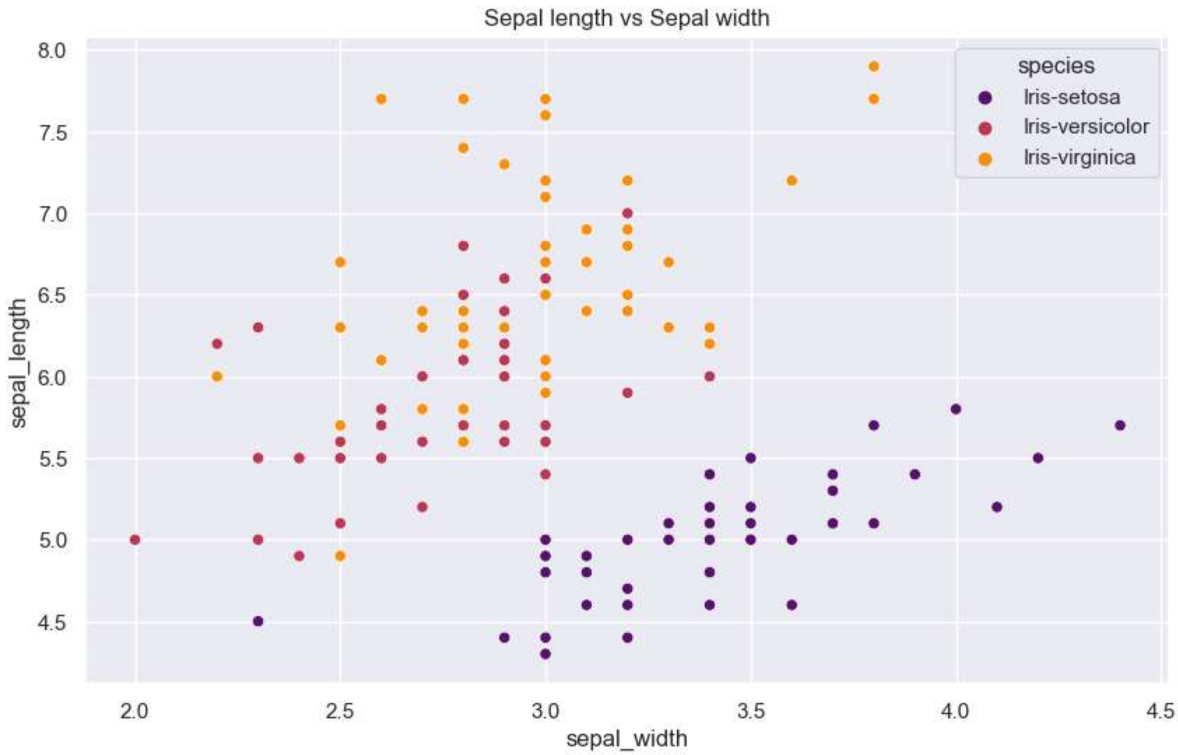
In [22]:
```
sns.lmplot(x = 'petal_width', y = 'petal_length', data = df, col = 'species', |
plt.show()
```



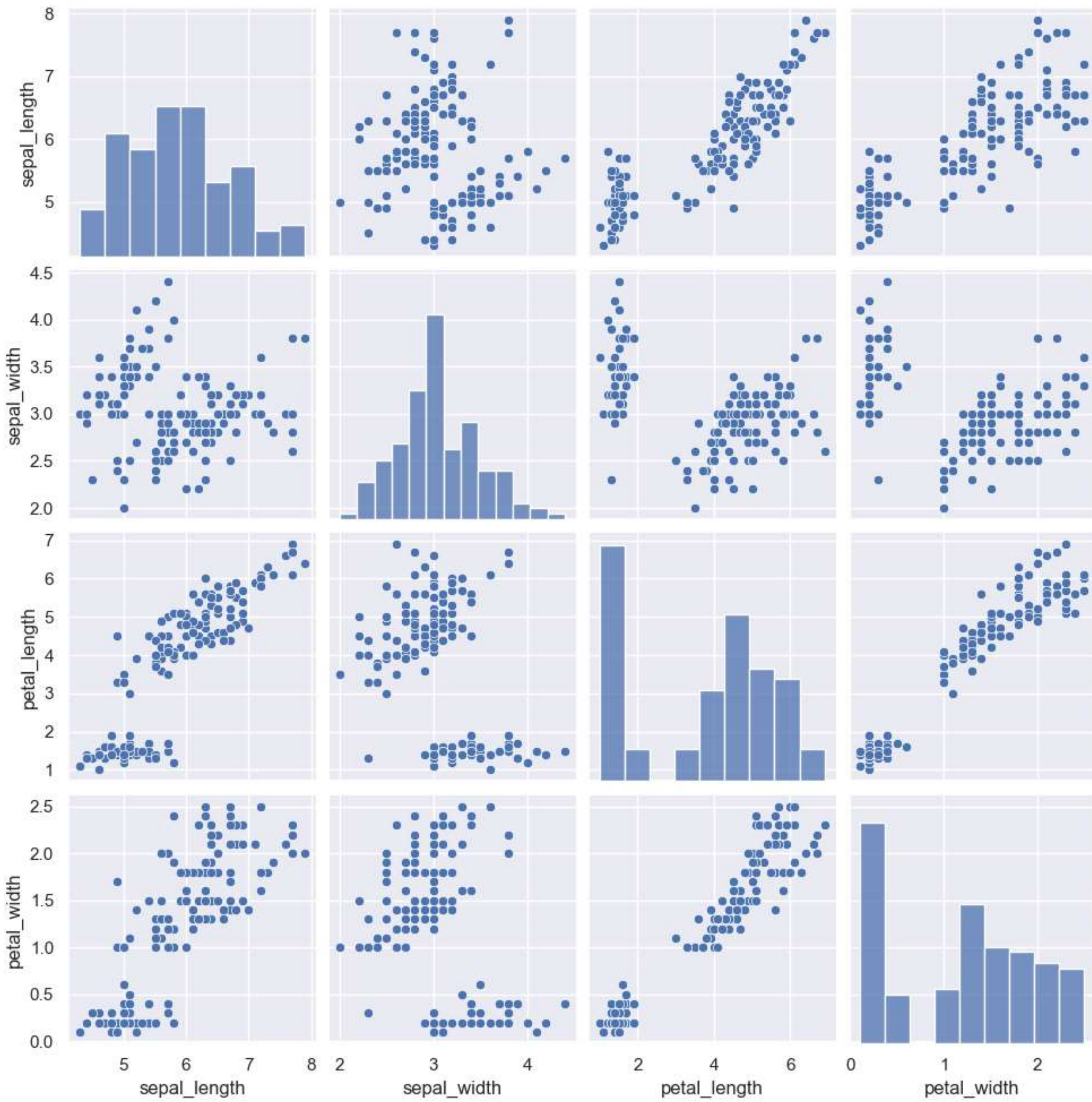
In [23]:
```
plt.figure(figsize=(10,6))
sns.scatterplot(data=df, x='petal_width', y='petal_length', hue = 'species', pa
plt.title('Petal length vs Petal width')
plt.show()
```



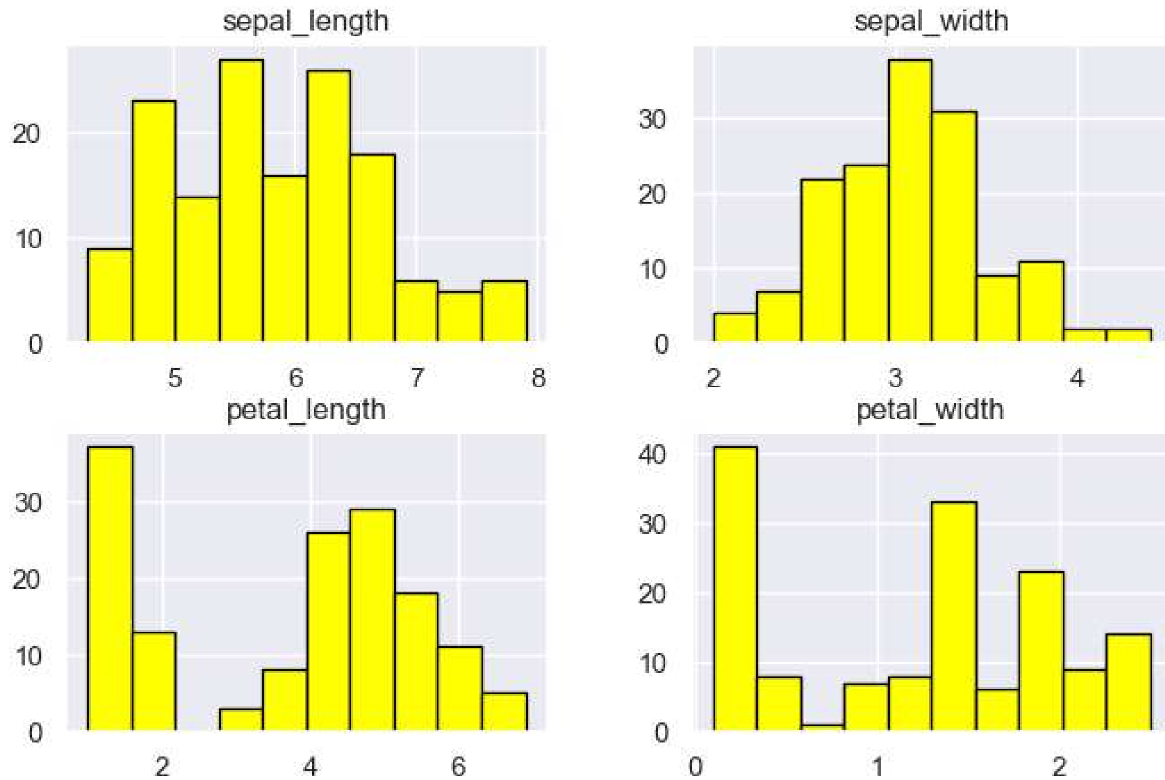
In [24]:
```
plt.figure(figsize=(10,6))
sns.scatterplot(data=df, x='sepal_width', y='sepal_length', hue = 'species', p
plt.title('Sepal length vs Sepal width')
plt.show()
```



In [25]:
```
sns.pairplot(df)
plt.show()
```

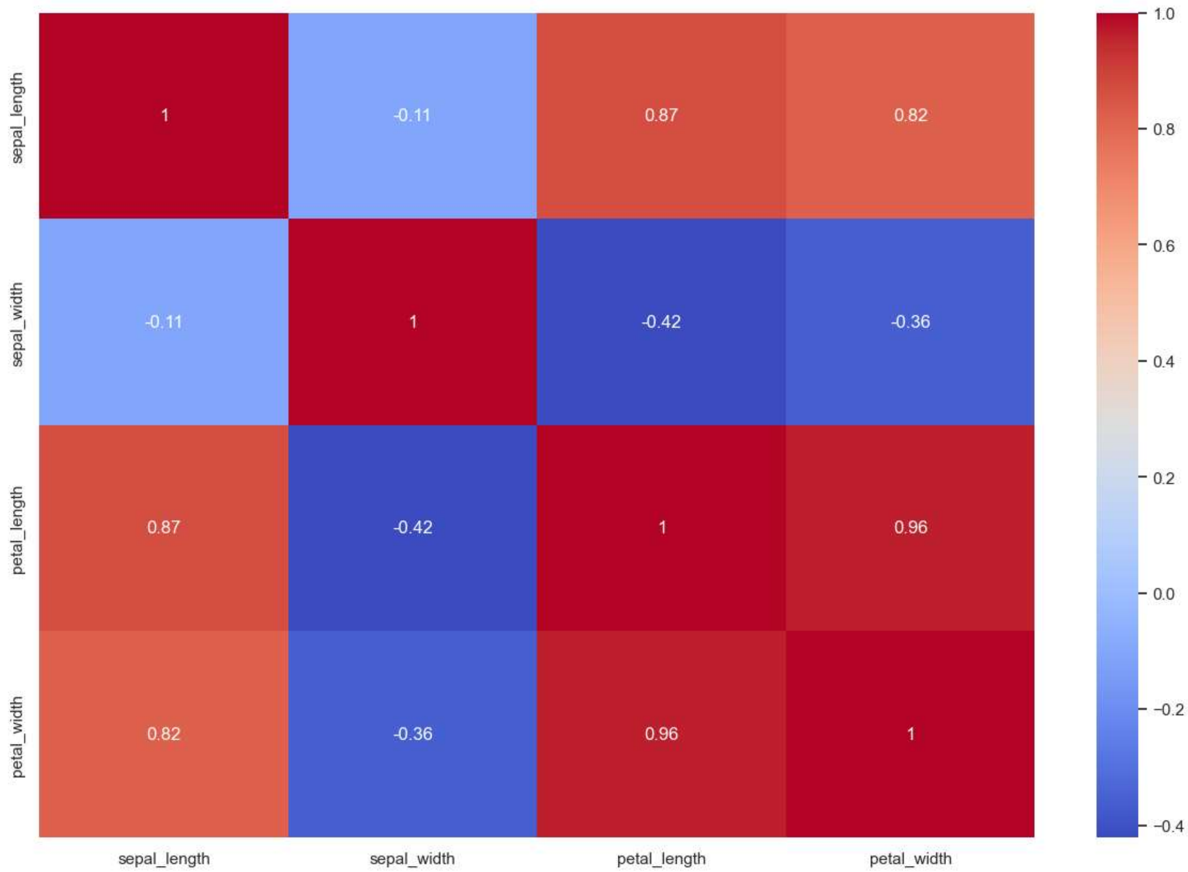


In [26]:
```
df.hist(bins=10, figsize=(8,5),color='yellow', edgecolor='black')
plt.show()
```

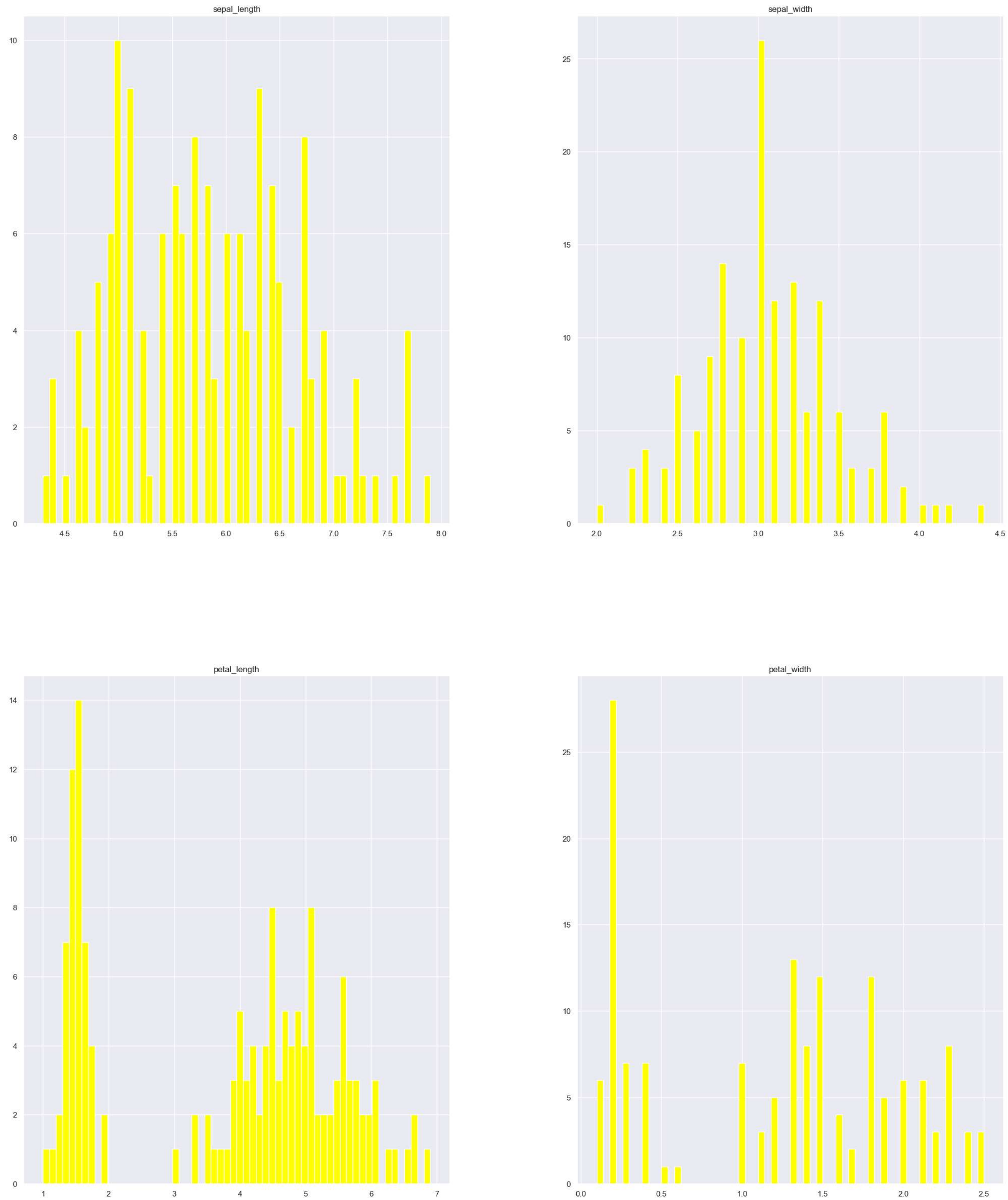


In [27]:
```
plt.figure(figsize=(15,10))
df_corr_matrix = df.corr()
print(df_corr_matrix)
sns.heatmap(df_corr_matrix, cmap='coolwarm', annot=True)
plt.show()
```

```
              sepal_length  sepal_width  petal_length  petal_width
sepal_length      1.000000    -0.109369      0.871754     0.817954
sepal_width      -0.109369     1.000000     -0.420516    -0.356544
petal_length      0.871754    -0.420516      1.000000     0.962757
petal_width       0.817954    -0.356544      0.962757     1.000000
```



In [28]:
```
df.hist(bins=60,figsize=(25,30),color='yellow')
plt.show()
```



Label encoding

In [29]:
```
df['species']=df['species'].astype('category')
df['species']=df['species'].cat.codes
```

In [30]:
```
df.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    int8   
dtypes: float64(4), int8(1)
memory usage: 5.0 KB
```

Feature scaling

In [31]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_absolute_percentage_error, mean_squa
from sklearn import metrics
```

In [32]:
```
x= df.drop(['species'],axis=1)
y= df[['species']]
```

In [33]:
```
x.head()
```

Out[33]:

	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

In [34]:
```
y.head()
```

Out[34]:

	species
0	0
1	0
2	0
3	0
4	0

In [35]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc_x = sc.fit_transform(x)
pd.DataFrame(sc_x)
```

Out[35]:

	0	1	2	3
0	-0.900681	1.032057	-1.341272	-1.312977
1	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.385353	0.337848	-1.398138	-1.312977
3	-1.506521	0.106445	-1.284407	-1.312977
4	-1.021849	1.263460	-1.341272	-1.312977
...	...	...	...	...
145	1.038005	-0.124958	0.819624	1.447956
146	0.553333	-1.281972	0.705893	0.922064
147	0.795669	-0.124958	0.819624	1.053537
148	0.432165	0.800654	0.933356	1.447956
149	0.068662	-0.124958	0.762759	0.790591

150 rows × 4 columns

VIF Variance Inflation Factor

In [36]:
```
variable = sc_x
variable.shape
```

Out[36]: (150, 4)

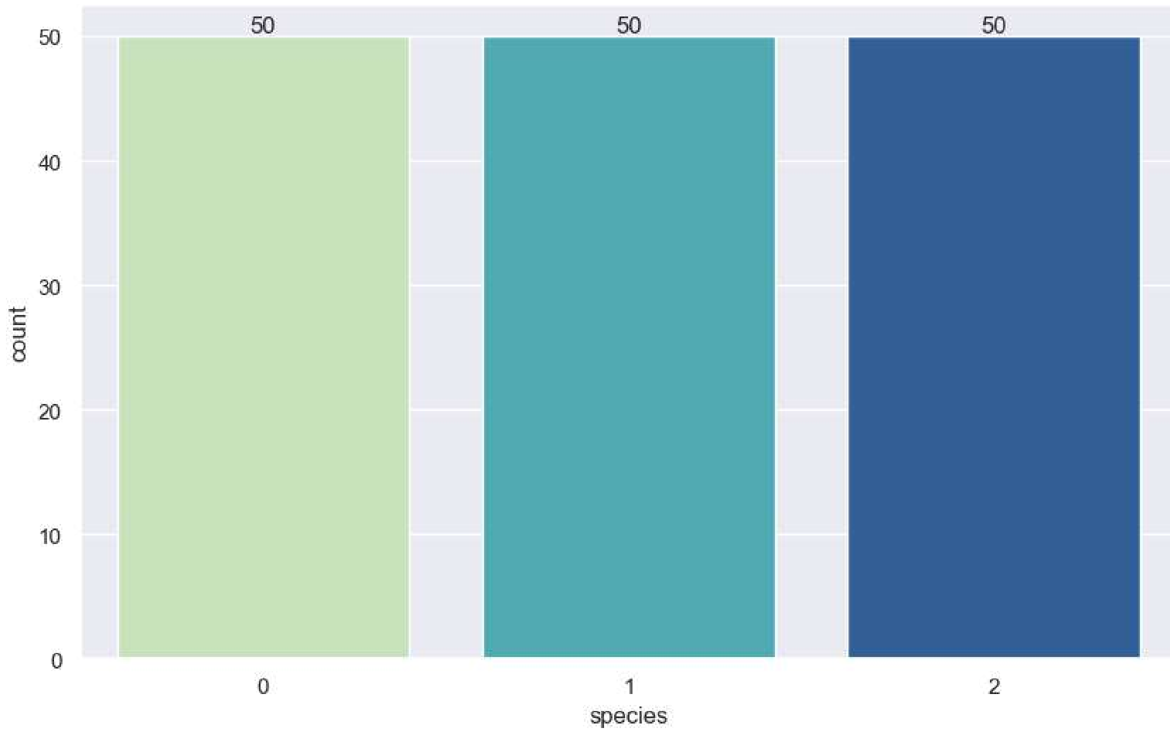
In [37]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
variable = sc_x

vif = pd.DataFrame()

vif['Variance Inflation Factor'] = [variance_inflation_factor(variable, i ) fo

vif['Features'] = x.columns
```

In [38]:
```
vif
```

Out[38]:

	Variance Inflation Factor	Features
0	7.103113	sepal_length
1	2.099039	sepal_width
2	31.397292	petal_length
3	16.141564	petal_width

In [39]:
```
plt.figure(figsize=(10,6),dpi=100)
ax=sns.countplot(x='species',data=df,palette='YlGnBu')

for i in ax.containers:
    ax.bar_label(i)
```



Split the data into training and test for building the model and for prediction

In [40]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, rando
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(105, 4) (45, 4) (105, 1) (45, 1)
```

In [41]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.ensemble import VotingClassifier
from sklearn.metrics import confusion_matrix, classification_report, accuracy_
```

In [42]:
```
# LogisticRegression
logistic = LogisticRegression()
lr = logistic.fit(x_train, y_train)
y_pred_lr = logistic.predict(x_test)
accuracy_lr = accuracy_score(y_test, y_pred_lr)

# DecisionTree
dtree = DecisionTreeClassifier()
dt = dtree.fit(x_train, y_train)
y_pred_dt = dtree.predict(x_test)
accuracy_dt = accuracy_score(y_test, y_pred_dt)

# RandomForest
rfmodel = RandomForestClassifier()
rf = rfmodel.fit(x_train, y_train)
y_pred_rf = rfmodel.predict(x_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)

# BaggingClassifier
bagg = BaggingClassifier()
bg = bagg.fit(x_train, y_train)
y_pred_bg = bagg.predict(x_test)
accuracy_bg = accuracy_score(y_test, y_pred_bg)

# AdaBoostClassifier
ada = AdaBoostClassifier()
ad = ada.fit(x_train, y_train)
y_pred_ad = ada.predict(x_test)
accuracy_ad = accuracy_score(y_test, y_pred_ad)

# GradientBoostingClassifier
gdb = GradientBoostingClassifier()
gd = gdb.fit(x_train, y_train)
y_pred_gd = gdb.predict(x_test)
accuracy_gd = accuracy_score(y_test, y_pred_gd)


# SVM
svc = SVC()
sv = svc.fit(x_train, y_train)
y_pred_sv = svc.predict(x_test)
accuracy_sv = accuracy_score(y_test, y_pred_sv)

# KNN
knn = KNeighborsClassifier()
kn = knn.fit(x_train, y_train)
y_pred_knn = knn.predict(x_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)

# GaussianNB
naive_gb = GaussianNB()
ngb = naive_gb.fit(x_train, y_train)
y_pred_ngb = naive_gb.predict(x_test)
accuracy_ngb = accuracy_score(y_test, y_pred_ngb)

# BernoulliNB
naive_bn = BernoulliNB()
```

```
nbr = naive_bn.fit(x_train, y_train)
y_pred_nbr = naive_bn.predict(x_test)
accuracy_nbr = accuracy_score(y_test, y_pred_nbr)


from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import StackingClassifier
```

In [43]:
```
evc = VotingClassifier(estimators=[('lr',lr),('dt',dt),('rf', rf),('bg', bg),(
                                   ('gd', gd),('sv', sv),('kn', kn),
                                   ('ngb', ngb),('nbr', nbr)], voting='hard')

model_evc = evc.fit(x_train, y_train)
pred_evc = evc.predict(x_test)
accuracy_evc = accuracy_score(y_test, pred_evc)
```

In [44]:
```
list1 = ['LogisticRegression','DecisionTree','RandomForest','Bagging','Adaboos
         'GradientBoosting','SupportVector','KNearestNeighbors',
         'NaiveBayesGaussian','NaiveBayesBernoullies','VotingClassifier']
```

In [45]:
```
list2 = [accuracy_lr, accuracy_dt, accuracy_rf, accuracy_bg,accuracy_ad, accura
         , accuracy_sv, accuracy_knn, accuracy_ngb, accuracy_nbr, accuracy_evc
```

In [46]:
```
list3 = [logistic, dtree, rfmodel, bagg, ada, gdb, svc, knn, naive_gb,naive_bn
```

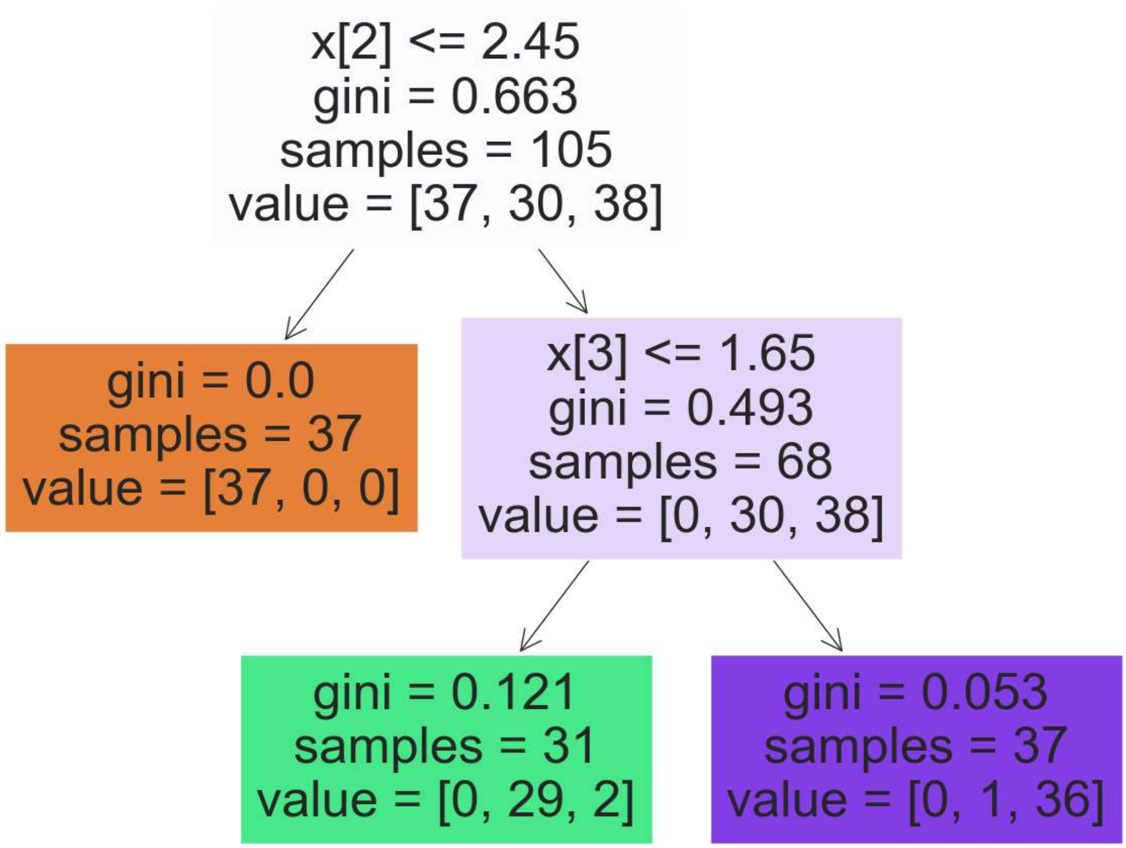
Classification Report

In [47]:
```
print('LogisticRegression: \n',classification_report(y_test,y_pred_lr))
print('DecisionTreeClassifier: \n',classification_report(y_test,y_pred_dt))
print('RandomForestClassifier: \n',classification_report(y_test,y_pred_rf))
print('BaggingClassifier: \n',classification_report(y_test,y_pred_bg))
print('AdaboostClassifier: \n',classification_report(y_test,y_pred_ad))
print('SupportVectorClassifier: \n',classification_report(y_test,y_pred_sv))
print('KNearestNeighborsClassifier: \n',classification_report(y_test,y_pred_kn
print('NaiveBayesGaussianClassifier: \n',classification_report(y_test,y_pred_n
print('NaiveBayesBernoulliesClassifier: \n',classification_report(y_test,y_pre
```

```
LogisticRegression:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      0.95      0.97        20
           2       0.92      1.00      0.96        12

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45

DecisionTreeClassifier:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       0.95      0.95      0.95        20
           2       0.92      0.92      0.92        12

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45

RandomForestClassifier:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       0.95      0.95      0.95        20
           2       0.92      0.92      0.92        12

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45

BaggingClassifier:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       0.95      0.95      0.95        20
           2       0.92      0.92      0.92        12

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45

AdaboostClassifier:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       0.90      0.95      0.93        20
           2       0.91      0.83      0.87        12

    accuracy                           0.93        45
   macro avg       0.94      0.93      0.93        45
weighted avg       0.93      0.93      0.93        45

SupportVectorClassifier:
               precision    recall  f1-score   support
```

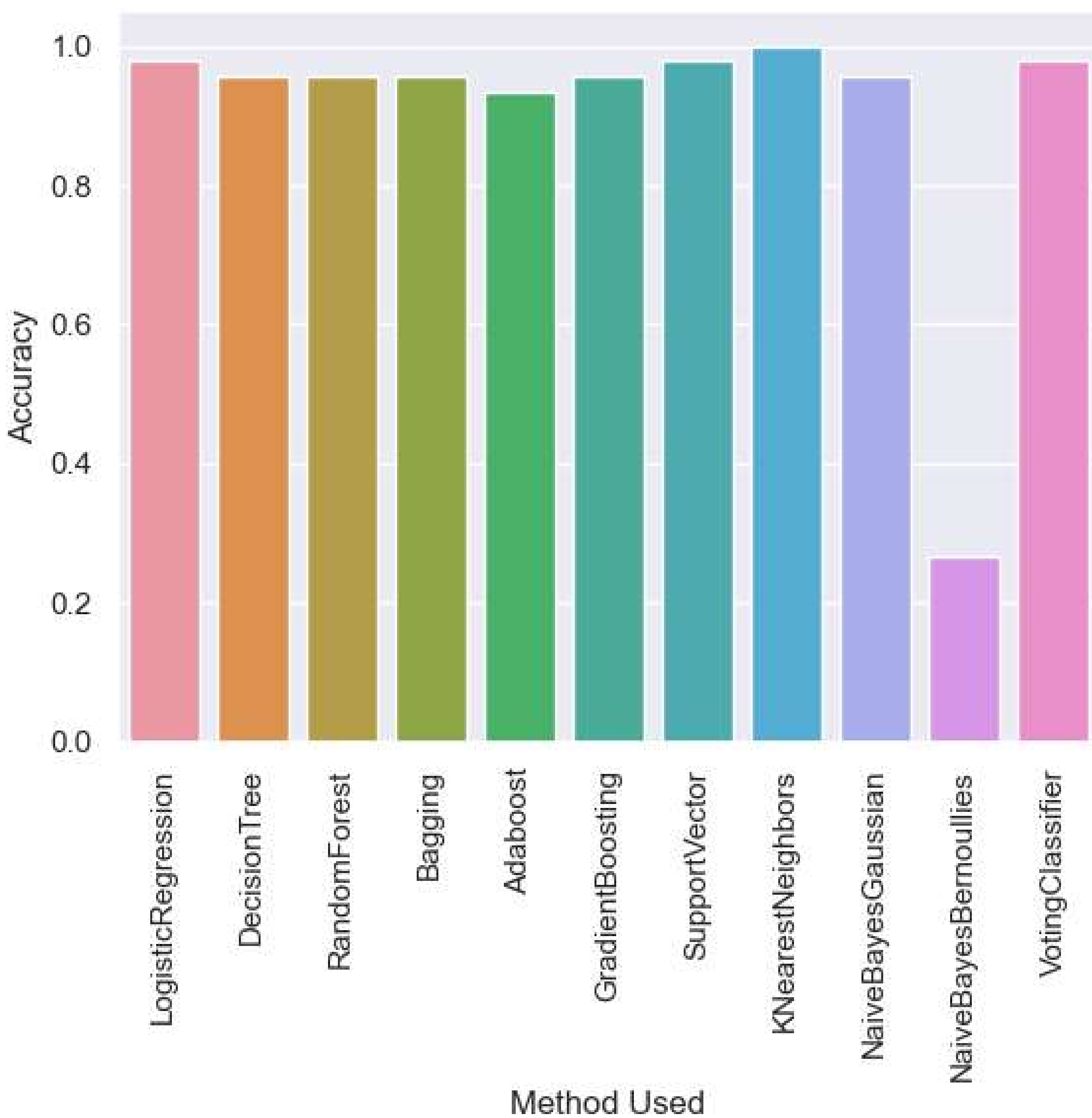
```
           0       1.00      1.00      1.00        13
           1       1.00      0.95      0.97        20
           2       0.92      1.00      0.96        12

    accuracy                           0.98        45
   macro avg       0.97      0.98      0.98        45
weighted avg       0.98      0.98      0.98        45

KNearestNeighborsClassifier:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      1.00      1.00        20
           2       1.00      1.00      1.00        12

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45

NaiveBayesGaussianClassifier:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       0.95      0.95      0.95        20
           2       0.92      0.92      0.92        12

    accuracy                           0.96        45
   macro avg       0.96      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45

NaiveBayesBernoulliesClassifier:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        13
           1       0.00      0.00      0.00        20
           2       0.27      1.00      0.42        12

    accuracy                           0.27        45
   macro avg       0.09      0.33      0.14        45
weighted avg       0.07      0.27      0.11        45
```

In [54]:
```
# plot the decisionTree Classifier
dt_clf = DecisionTreeClassifier(max_depth=2)
dt_clf.fit(x_train, y_train)

y_pred_train = dt_clf.predict(x_train)
y_pred_test = dt_clf.predict(x_test)


from sklearn.tree import plot_tree
from sklearn.tree import export_text
plt.figure(figsize=(15,12))
plot_tree(dt_clf, filled=True, feature_names=None,
          class_names=None)
plt.show()
```



In [55]:
```
final_accuracy = pd.DataFrame({'Method Used': list1, "Accuracy": list2})
print(final_accuracy)
charts = sns.barplot(x="Method Used", y = 'Accuracy', data=final_accuracy)
charts.set_xticklabels(charts.get_xticklabels(), rotation=90)
print(charts)
```

```
               Method Used  Accuracy
0       LogisticRegression  0.977778
1             DecisionTree  0.955556
2             RandomForest  0.955556
3                  Bagging  0.955556
4                 Adaboost  0.933333
5         GradientBoosting  0.955556
6            SupportVector  0.977778
7        KNearestNeighbors  1.000000
8       NaiveBayesGaussian  0.955556
9   NaiveBayesBernoullies  0.266667
10        VotingClassifier  0.977778
Axes(0.125,0.11;0.775x0.77)
```



Conclusion

```
I mainly worked through the Python documentation manuals of the Pandas,
Matplotlib and Seaborn modules as well as the Python 3 documentation.
The pandas library is quite intuitive and in a valuable tool in investigating
and analysing multi-class multi-variates datasets such as the Iris dataset.
There is not such a large difference between the sepal lengths of the three
Iris species, although the Setosa is again showing the smallest average
measurements. The average sepal width of the Setosa however is actually larger
than the averages for the other two species but not by a huge amount.
From the summary statistics of the sepal and petal measurements by class type
it would appear that the differences between the Iris Setosa and the other two
species is more pronounced that any other differences between the three
classes.
Best accuracy Logistic regression,random forest,decision
tree,bagging,adaboost,gradient boosting,support
vector,knearestneighbors,naivebayesgausisian, therom,
```