In [1]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#Loading the data set
df=pd.read_csv('iris.csv')
df.head()
```

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

In [3]:
```
#to display stats about data
df.describe()
```

Out[3]:

	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 [4]:
```
#for basic information about dataset
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 [5]:
```
df['species'].value_counts()
```

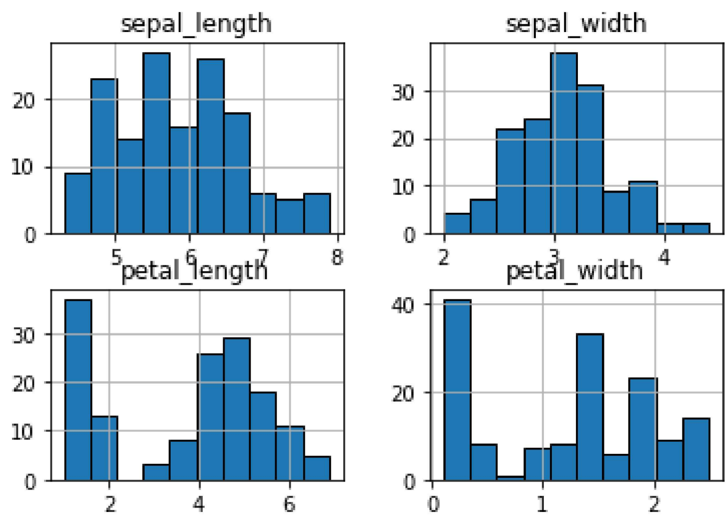
Out[5]:
```
Iris-versicolor    50
Iris-setosa        50
Iris-virginica     50
Name: species, dtype: int64
```

In [6]:
```
#processing the dataset
#check for null values
df.isnull().sum()
```

Out[6]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

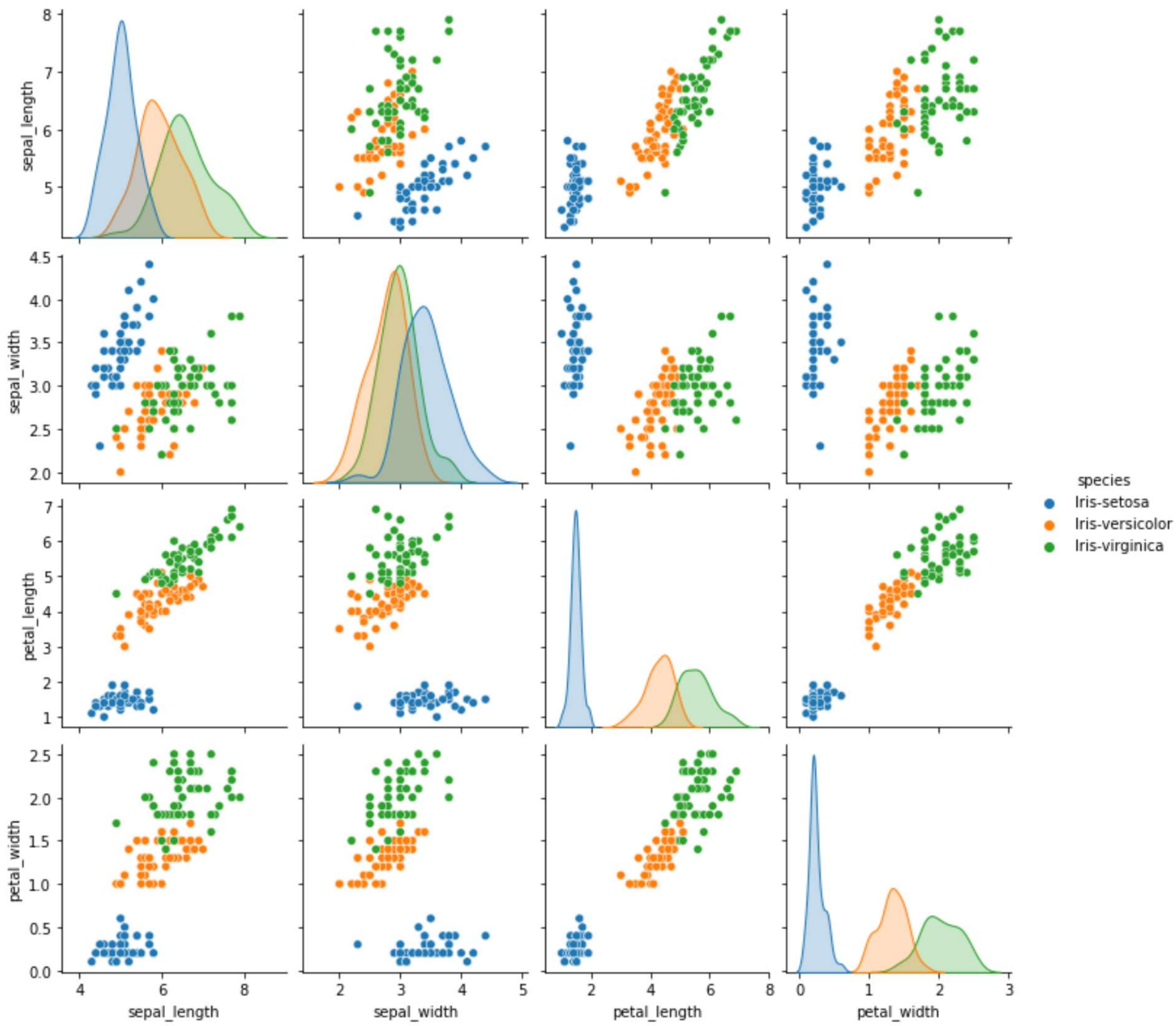
In [7]:
```
#Extraordinary Data Analysis
df.hist(edgecolor='black')
```

Out[7]:
```
array([[<AxesSubplot:title={'center':'sepal_length'}>,
        <AxesSubplot:title={'center':'sepal_width'}>],
       [<AxesSubplot:title={'center':'petal_length'}>,
        <AxesSubplot:title={'center':'petal_width'}>]], dtype=object)
```



In [8]:
```
sns.pairplot(df,hue = 'species')
```

Out[8]: <seaborn.axisgrid.PairGrid at 0x257005a7850>



In [9]:
```
#observation
#All type of flowers are well separable for petallength and petalwidth
#also all type of bit of well seperable for petalwidth and sepalwidth
```

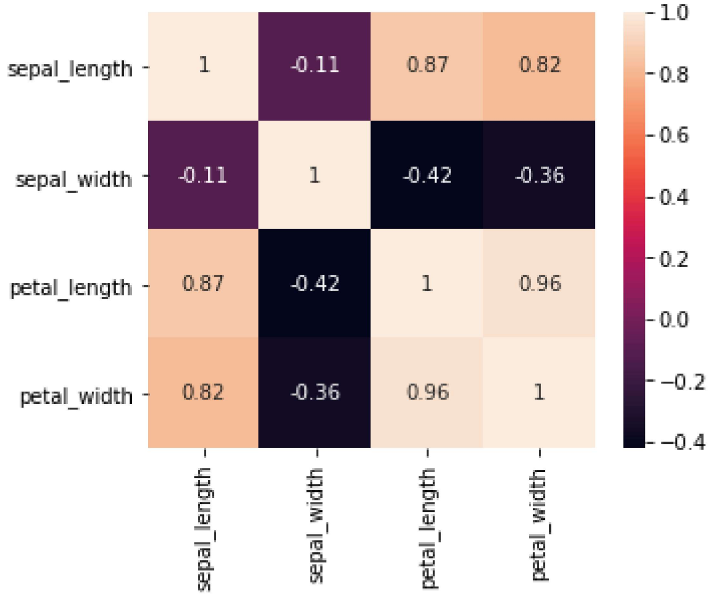
In [10]:
```
#as this is classification problem,we will use classification algorithm for model
```

In [11]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
```

In [12]:
```
df.corr()

corr1=df.corr()
fig,ax=plt.subplots(figsize=(5,4))
sns.heatmap(corr1,annot=True,ax=ax)
```

Out[12]: <AxesSubplot:>



In [13]:
```
df.corr()
```

Out[13]:

	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 [14]:
```
#observations
#petalLength and petalWidth:corr 0.96 :Having highest correlation
#patelLength and sepalLength:corr 0.87:Having second highest correlation
#petalWidth and sepalLength:corr 0.82 :Having fair enough correlation

#in the above figure,we can see that sepalLength and sepalWidth are not correlate
#while the pataLength and petalWidth are highly correlated
```

In [15]:
```
#Iris-setosa :It's usually having smaller features except sepalwidth
#Iris-versicolor:It's having bigger features except sepalwidth
```

In [16]:
```
#observation
```

In [17]:
```
plt.figure(fig size=(10,10))

plt.subplot(2,2,1)
sns.violinplot(data=df, x='Species',y='SepalLength',palette='Set1')
plt.subplot(2,2,2)
sns.violinplot(data=df, x='Species',y='SepalWidth',palette='Set1')

plt.subplot(2,2,3)
sns.violinplot(data=df,x='Species', y='PetalLength',palette='Set1')
plt.subplot(2,2,4)
sns.violinplot(data=df, x='Species', y='PetalWidth',palette='Set1')
```

```
  File "<ipython-input-17-243b53425a80>", line 1
    plt.figure(fig size=(10,10))
                   ^
SyntaxError: invalid syntax
```

In [ ]:
```
x_train, x_test, y_train, y_test=  train X,y, test_size=0.33 ,random_state=4
```

In [ ]:
```
model = LogisticRegression()
model.fit(X_train,y_train)
prediction=model.predict(X_test)
print('Logistic Regression accuracy= ',metrics.accuracy_score(prediction,y_test))
```

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder() #LableEncoder can be normalize Lable
```

In [ ]:
```
df['species']=le.fit_transform(df['species']) #fit_transform:fit label encoder ar
df.head()
```

In [ ]:
```
X=df.drop(columns=['species']) #Drop Down
y=df['species']
X[:5] ## Return list from begining untel index 5
```

In [ ]:
```
y[:5]
```

In [ ]:
```
#splitting Dataset into Training set and Test set
```

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=1)
```

In [ ]:
```
#selecting the model and Metrics
```

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

In [ ]:
```
lr=LogisticRegression()
kmn=KNeighborsClassifier()
svm=SVC()
nb=GaussianNB()
dt=DecisionTreeClassifier()
rf=RandomForestClassifier()
```

models= [lr,kmn,svm,nb,dt,rf] scores=[] for model in models: model.fit(X_train,y_train) y_pred=model.predict(X_test) scores.append(accuracy_score(y_test,y_pred) print("Accuracy of" +type(model.*name*+"is".accuracy_score(y_test,y_pred)

In [ ]:
```
#Decision tree
model = DecisionTreeClassifier()
model.fit(X_train,y_train)

prediction = model.predict(X_test)
print('Decision Tree accuracy = ', metrics.accuracy_score(prediction,y_test))
```

In [ ]:
```
#Support vector Machine (SVM)
model = svm.SVC()
model.fit(X_train,y_train)

prediction = model.predict(X_test)
print('SVM accuracy = ', metrics.accuracy_score(prediction,y_test))
```

In [ ]: