Appendix - Python Code

Classification Algorithms on Iris Dataset

Importing required libraries

In [271]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
```

Data Preprocessing

In [272]:
```
data = pd.read_csv("Iris.csv")
```

In [273]:
```
data.head()
```

Out[273]:

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

In [274]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

Dropping 'Id' column

In [275]:
```
data = data.drop('Id', axis = 1)
```

Checking for NA values

In [276]:
```
data.isna().sum()
```

Out[276]:
```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

Checking for duplicate values

In [277]:
```
data[data.duplicated() == True]
```

Out[277]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
34	4.9	3.1	1.5	0.1	Iris-setosa
37	4.9	3.1	1.5	0.1	Iris-setosa
142	5.8	2.7	5.1	1.9	Iris-virginica

Descriptive statistics

In [278]:
```
data.describe()
```

Out[278]:

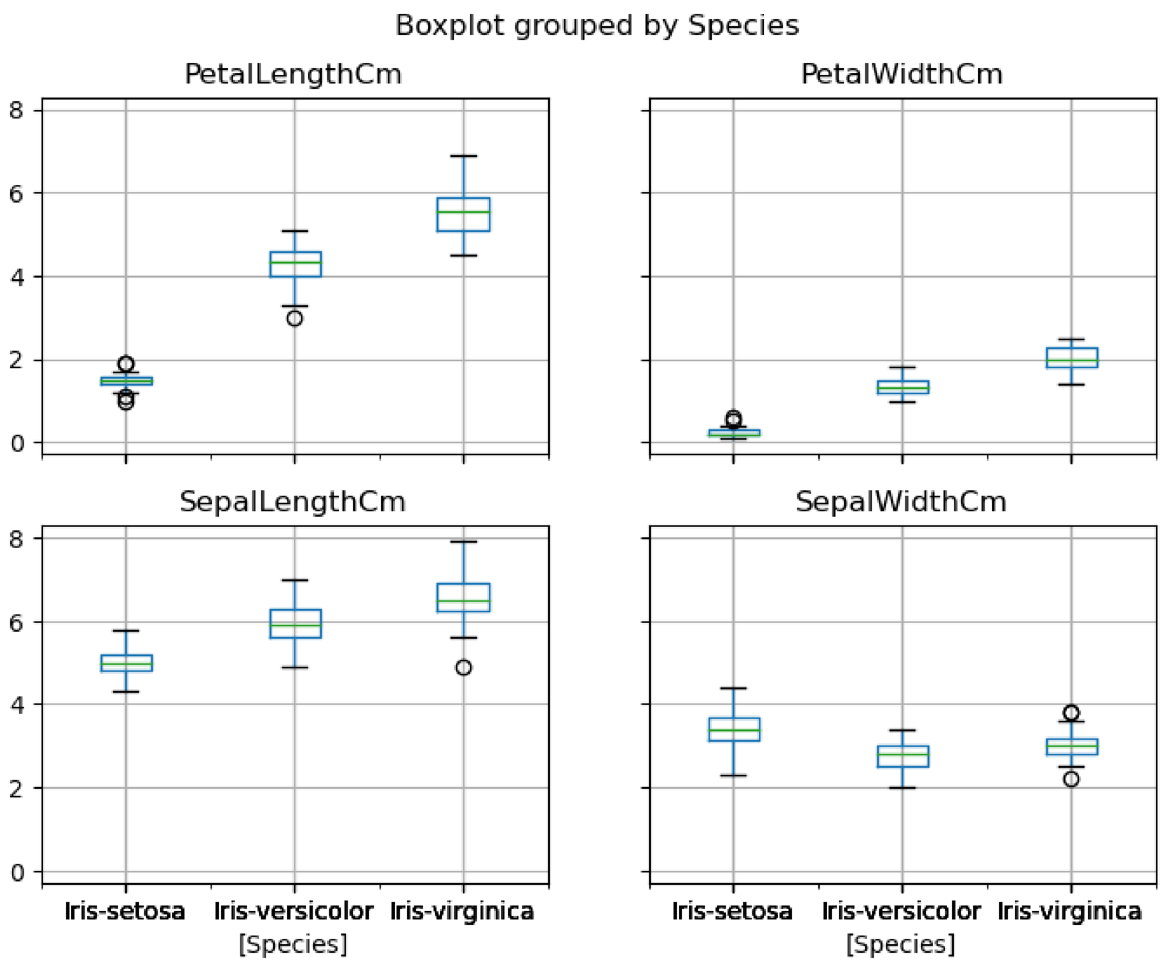
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000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

Data Exploration

Box plot grouped by species

In [279]:
```
data.boxplot(by="Species", figsize=(8, 6))
```

Out[279]:
```
array([[<Axes: title={'center': 'PetalLengthCm'}, xlabel='[Species]'>,
        <Axes: title={'center': 'PetalWidthCm'}, xlabel='[Species]'>],
       [<Axes: title={'center': 'SepalLengthCm'}, xlabel='[Species]'>,
        <Axes: title={'center': 'SepalWidthCm'}, xlabel='[Species]'>]],
      dtype=object)
```



Violin Chart

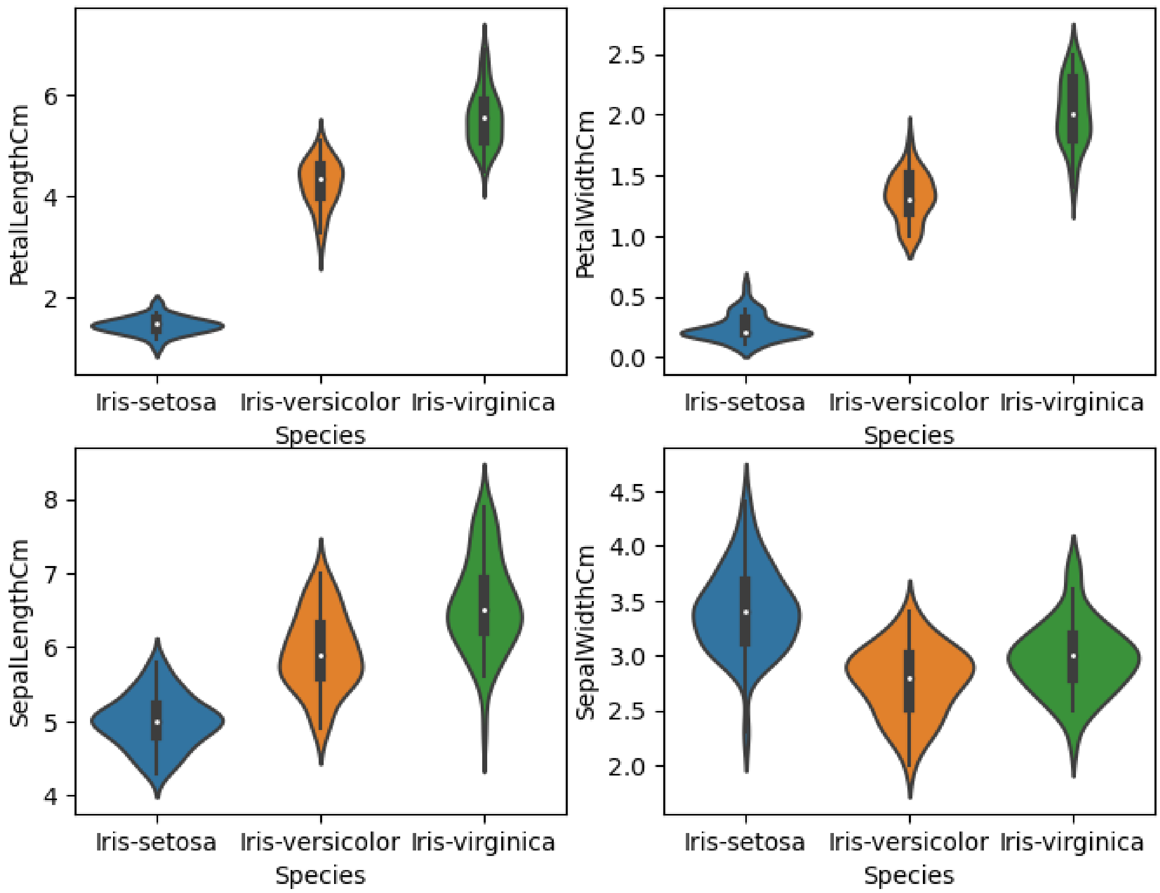
In [280]:
```
plt.figure(figsize=(8,6))
plt.subplot(2,2,1)
sns.violinplot(x='Species', y='PetalLengthCm', data=data)

plt.subplot(2,2,2)
sns.violinplot(x='Species', y='PetalWidthCm', data=data)

plt.subplot(2,2,3)
sns.violinplot(x='Species', y='SepalLengthCm', data=data)

plt.subplot(2,2,4)
sns.violinplot(x='Species', y='SepalWidthCm', data=data)
```

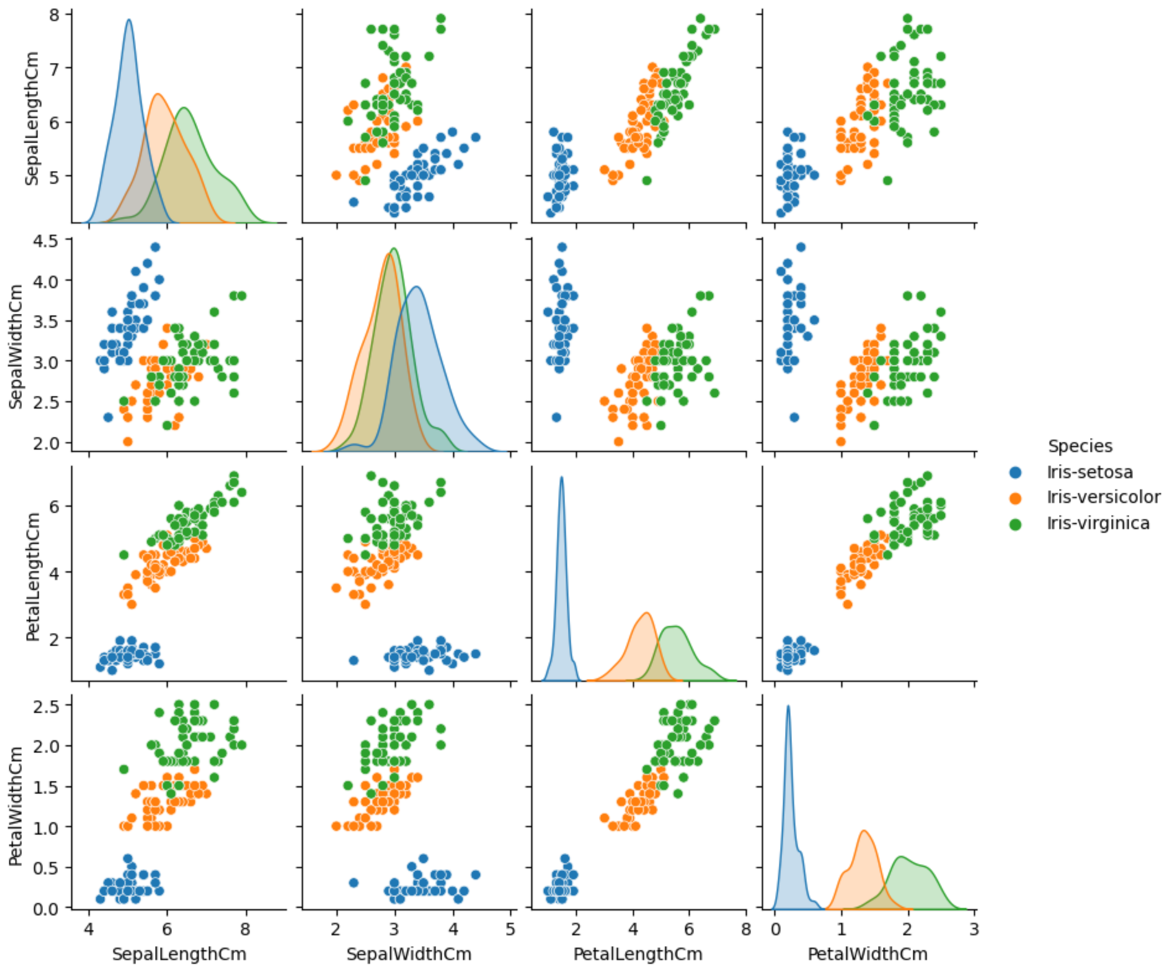
Out[280]: <Axes: xlabel='Species', ylabel='SepalWidthCm'>



Pair plot

In [281]:
```
sns.pairplot(data, hue='Species', height=2)
```

Out[281]: <seaborn.axisgrid.PairGrid at 0x1adf71853f0>



Classification Algorithms

70 - 30 Train test split

In [309]:
```
#Splitting the data into test and train sets

train, test = train_test_split(data, test_size = 0.3)

print(train.shape)
print(test.shape)

#storing the training data attributes as x
train_x = train[['SepalLengthCm','SepalWidthCm','PetalLengthCm', 'PetalWidthCr

#output of our training data
train_y = train.Species

#storing test data attributes
test_x= test[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']]

#output of our test data
test_y =test.Species
```

```
(105, 5)
(45, 5)
```

In [283]:
```
y_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 45 entries, 114 to 10
Data columns (total 1 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   Species  45 non-null     object
dtypes: object(1)
memory usage: 720.0+ bytes
```

We will use following algorithms and compare the accuracy

1. Decision Tree
2. Support Vector Machine
3. Random Forests

1. Decision Tree

In [312]:

```
# Decision tree classifier object
dtree = DecisionTreeClassifier()

# Training decision tree classifier
dtree = dtree.fit(train_x, train_y)

# Predicting the response for test data
dtree_pred = dtree.predict(test_x)
```

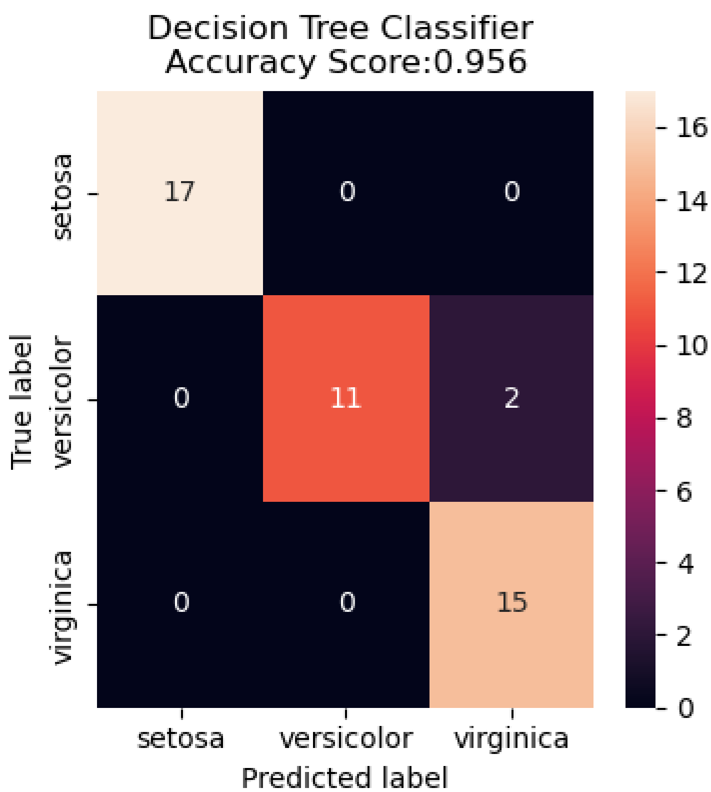
In [345]:
```
# Evaluating the model

# Creating confusion matrix
dtree_cm = metrics.confusion_matrix(test_y, dtree_pred)

# Converting confusion matrix into data frame for better plotting
dtree_cm_df = pd.DataFrame(dtree_cm,
                           index = ['setosa', 'versicolor', 'virginica'],
                           columns = ['setosa', 'versicolor', 'virginica'])

# Plotting confusion matrix
plt.figure(figsize=(4,4))
sns.heatmap(dtree_cm_df, annot=True)
plt.title('Decision Tree Classifier \nAccuracy Score:{0:.3f}'.format(metrics.a
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# Decision Tree Classification Report
print(metrics.classification_report(test_y, dtree_pred))
```



```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        17
Iris-versicolor       1.00      0.85      0.92        13
 Iris-virginica       0.88      1.00      0.94        15

       accuracy                           0.96        45
      macro avg       0.96      0.95      0.95        45
   weighted avg       0.96      0.96      0.96        45
```

2. Support Vector Machine

In [317]:

```python
# Creating SVM Classifier
svm_clf = svm.SVC(kernel='linear')

# Training the classifier
svm_clf.fit(train_x, train_y.values.ravel())

# Predicting the response for test set
svm_pred = svm_clf.predict(test_x)
```

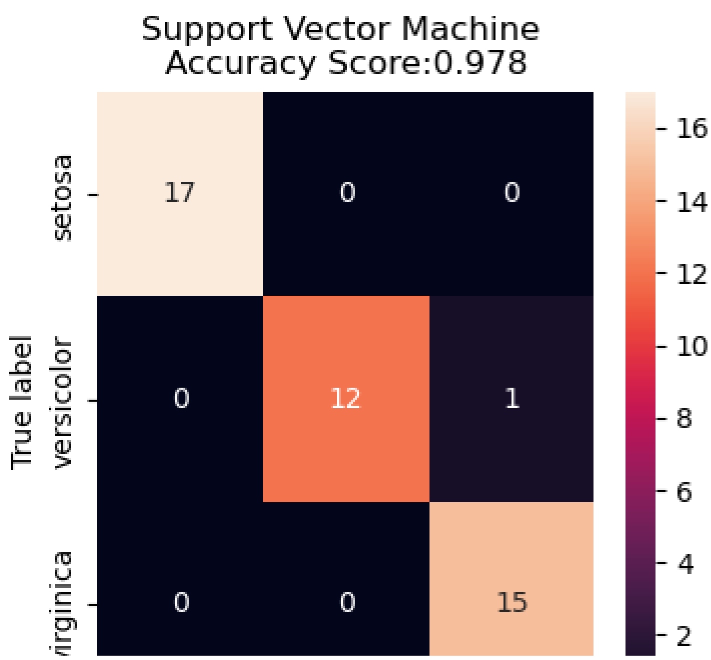
In [344]:
```
# Evaluating the model

# Creating confusion matrix
svm_cm = metrics.confusion_matrix(test_y, svm_pred)

# Converting confusion matrix into data frame for better plotting
svm_cm_df = pd.DataFrame(svm_cm,
                         index = ['setosa', 'versicolor', 'virginica'],
                         columns = ['setosa', 'versicolor', 'virginica'])

# Plotting confusion matrix
plt.figure(figsize=(4,4))
sns.heatmap(svm_cm_df, annot=True)
plt.title('Support Vector Machine \nAccuracy Score:{0:.3f}'.format(metrics.acc
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# Support Vector Machine Classification Report
print(metrics.classification_report(test_y, svm_pred))
```



3. Random Forest

In [334]:

```
# Creating Random Forest Classifier Instance
rf = RandomForestClassifier()

# Training the classifier
rf.fit(train_x, train_y)

# Predicting the response for test set
rf_pred = rf.predict(test_x)
```

In [336]:

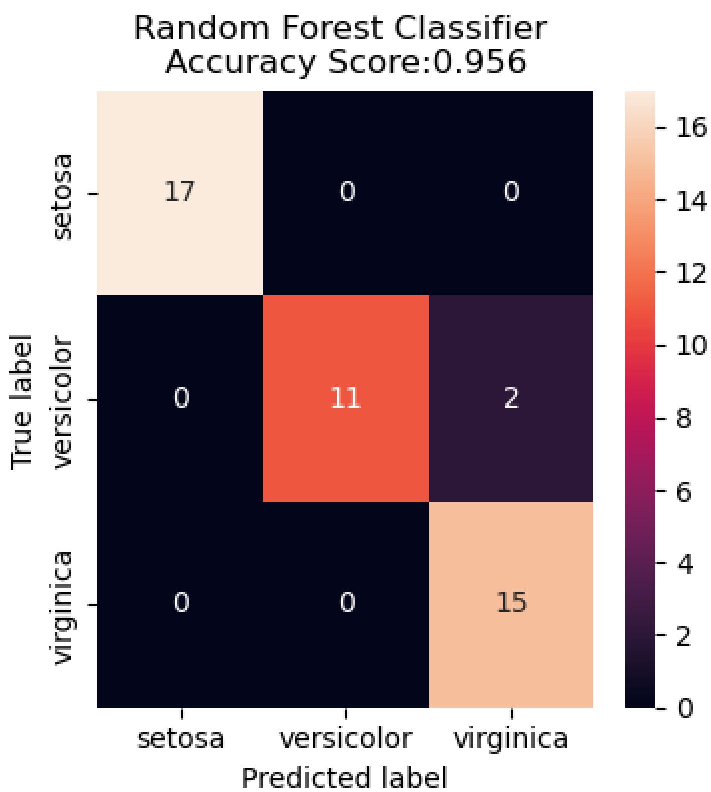
```python
# Evaluating the model

# Creating confusion matrix
rf_cm = metrics.confusion_matrix(test_y, rf_pred)

# Converting confusion matrix into data frame for better plotting
rf_cm_df = pd.DataFrame(rf_cm,
                        index = ['setosa', 'versicolor', 'virginica'],
                        columns = ['setosa', 'versicolor', 'virginica'])

# Plotting confusion matrix
plt.figure(figsize=(4,4))
sns.heatmap(rf_cm_df, annot=True)
plt.title('Random Forest Classifier \nAccuracy Score:{0:.3f}'.format(metrics.a
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()

# Random Forest Classification Report
print(metrics.classification_report(test_y, rf_pred))
```



	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	17
Iris-versicolor	1.00	0.85	0.92	13
Iris-virginica	0.88	1.00	0.94	15
accuracy			0.96	45
macro avg	0.96	0.95	0.95	45
weighted avg	0.96	0.96	0.96	45