Classification

- Covers classification based model
- Built various classification models based on different algorithm
- All models are based on `iris` dataset

Imports and Settings

In [54]:

```
import numpy as np
import pandas as pd

# Visualization Libraries
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import datasets # for using built-in datasets
from sklearn import metrics # for checking the model accuracy

#To plot the graph embedded in the notebook
%matplotlib inline
```

Load Dataset: `iris`

- we will load the `iris` dataset from the scikit-learn library.
- The `iris` dataset contains different attributes of iris flowers, and the correspinding species (class) of the iris.
- There are 150 samples (50 in each of three classes) and 5 columns (4 feature and one target varriables) at this dataset.
- The objective is to predict the class (species) of iris flower using the given features.

In [55]:
```
#loading the dataset direclty from sklearn
iris = datasets.load_iris()
```

Explore the Dataset

- sklearn returns Dictionary-like object.

In [56]:
```
# Let's print the keys of 'iris' dictionary object
iris.keys()
```

Out[56]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename'])
```

- data: contains the information for various iris flowers (feature matrix)
- target: species of the iris flowers (target array/vector)
- target_names: names of the species (classes) of iris flowers
- DESCR: describes the dataset
- feature_names: names of the features
- filename: the physical location of boston csv dataset

In [57]:
```
# Let's print the description of iris dataset
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
```

        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min Max   Mean   SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length: 4.3 7.9   5.84   0.83   0.7826
    sepal width: 2.0 4.4   3.05   0.43   -0.4194
    petal length: 1.0 6.9   3.76   1.76   0.9490 (high!)
    petal width: 0.1 2.5   1.20   0.76   0.9565 (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature. Fisher's paper is a classic in the field and
is referenced frequently to this day. (See Duda & Hart, for example.) The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant. One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.

```
    (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
  - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
    Structure and Classification Rule for Recognition in Partially Exposed
    Environments".  IEEE Transactions on Pattern Analysis and Machine
    Intelligence, Vol. PAMI-2, No. 1, 67-71.
  - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
    on Information Theory, May 1972, 431-433.
  - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
    conceptual clustering system finds 3 classes in the data.
  - Many, many more ...
```

- As we can see fron the dataset description that there is no missing values.
- And, it contains equal number of samples for each of the class of iris.

In [58]:

```
# Let's print the data (features matrix) of iris dataset
print(iris.data)
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
```

```
[5.1 3.7 1.5 0.4]
[4.6 3.6 1.  0.2]
[5.1 3.3 1.7 0.5]
[4.8 3.4 1.9 0.2]
[5.  3.  1.6 0.2]
[5.  3.4 1.6 0.4]
[5.2 3.5 1.5 0.2]
[5.2 3.4 1.4 0.2]
[4.7 3.2 1.6 0.2]
[4.8 3.1 1.6 0.2]
[5.4 3.4 1.5 0.4]
[5.2 4.1 1.5 0.1]
[5.5 4.2 1.4 0.2]
[4.9 3.1 1.5 0.2]
[5.  3.2 1.2 0.2]
[5.5 3.5 1.3 0.2]
[4.9 3.6 1.4 0.1]
[4.4 3.  1.3 0.2]
[5.1 3.4 1.5 0.2]
[5.  3.5 1.3 0.3]
[4.5 2.3 1.3 0.3]
[4.4 3.2 1.3 0.2]
[5.  3.5 1.6 0.6]
[5.1 3.8 1.9 0.4]
[4.8 3.  1.4 0.3]
[5.1 3.8 1.6 0.2]
[4.6 3.2 1.4 0.2]
[5.3 3.7 1.5 0.2]
[5.  3.3 1.4 0.2]
[7.  3.2 4.7 1.4]
[6.4 3.2 4.5 1.5]
[6.9 3.1 4.9 1.5]
[5.5 2.3 4.  1.3]
[6.5 2.8 4.6 1.5]
[5.7 2.8 4.5 1.3]
[6.3 3.3 4.7 1.6]
[4.9 2.4 3.3 1. ]
[6.6 2.9 4.6 1.3]
[5.2 2.7 3.9 1.4]
[5.  2.  3.5 1. ]
[5.9 3.  4.2 1.5]
```

```
[6.  2.2 4.  1. ]
[6.1 2.9 4.7 1.4]
[5.6 2.9 3.6 1.3]
[6.7 3.1 4.4 1.4]
[5.6 3.  4.5 1.5]
[5.8 2.7 4.1 1. ]
[6.2 2.2 4.5 1.5]
[5.6 2.5 3.9 1.1]
[5.9 3.2 4.8 1.8]
[6.1 2.8 4.  1.3]
[6.3 2.5 4.9 1.5]
[6.1 2.8 4.7 1.2]
[6.4 2.9 4.3 1.3]
[6.6 3.  4.4 1.4]
[6.8 2.8 4.8 1.4]
[6.7 3.  5.  1.7]
[6.  2.9 4.5 1.5]
[5.7 2.6 3.5 1. ]
[5.5 2.4 3.8 1.1]
[5.5 2.4 3.7 1. ]
[5.8 2.7 3.9 1.2]
[6.  2.7 5.1 1.6]
[5.4 3.  4.5 1.5]
[6.  3.4 4.5 1.6]
[6.7 3.1 4.7 1.5]
[6.3 2.3 4.4 1.3]
[5.6 3.  4.1 1.3]
[5.5 2.5 4.  1.3]
[5.5 2.6 4.4 1.2]
[6.1 3.  4.6 1.4]
[5.8 2.6 4.  1.2]
[5.  2.3 3.3 1. ]
[5.6 2.7 4.2 1.3]
[5.7 3.  4.2 1.2]
[5.7 2.9 4.2 1.3]
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
```

```
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
[6.5 3.2 5.1 2. ]
[6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.  2. ]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
[6.9 3.2 5.7 2.3]
[5.6 2.8 4.9 2. ]
[7.7 2.8 6.7 2. ]
[6.3 2.7 4.9 1.8]
[6.7 3.3 5.7 2.1]
[7.2 3.2 6.  1.8]
[6.2 2.8 4.8 1.8]
[6.1 3.  4.9 1.8]
[6.4 2.8 5.6 2.1]
[7.2 3.  5.8 1.6]
[7.4 2.8 6.1 1.9]
[7.9 3.8 6.4 2. ]
[6.4 2.8 5.6 2.2]
[6.3 2.8 5.1 1.5]
[6.1 2.6 5.6 1.4]
[7.7 3.  6.1 2.3]
[6.3 3.4 5.6 2.4]
[6.4 3.1 5.5 1.8]
[6.  3.  4.8 1.8]
[6.9 3.1 5.4 2.1]
[6.7 3.1 5.6 2.4]
[6.9 3.1 5.1 2.3]
[5.8 2.7 5.1 1.9]
[6.8 3.2 5.9 2.3]
```

```
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
```

In [59]:

```
# Let's check the shape of features matrix
print(iris.data.shape)
```

```
(150, 4)
```

In [60]:

```
# Let's print the feature names
print(iris.feature_names)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

In [61]:

```
# Let's print the target vector
print(iris.target)
```

```
[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 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 2]
```

In [62]:

```
# Let's check the shape of target
print(iris.target.shape)
```

```
(150,)
```

In [63]:

```
# Let's print the target class/species names
print(iris.target_names)
```

```
['setosa' 'versicolor' 'virginica']
```

In [64]:

```
# Let's load the data (features matrix) into pandas DataFrame

iris_df = pd.DataFrame(iris.data, columns=iris.feature_names)

# Print the DataFrame
iris_df
```

Out[64]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
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

- We can see that the target label `species` is missing in the DataFrame (that is alright).
- Now, create a new column for target label and add it to the dataframe.

In [65]:

```
# Let's add target label into pandas DataFrame

iris_df['species'] = iris.target

# Print the DataFrame
iris_df
```

Out[65]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	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

150 rows × 5 columns

- Target species names:
  - 0 = 'setosa'
  - 1 = 'versicolor'
  - 2 = 'virginica'

In [66]:

```
# replace the target values with species names
iris_df['species'] = iris_df['species'].replace([0, 1, 2], ['setosa', 'versicolor', 'virginica'])

# Print the DataFrame
iris_df
```

Out[66]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

Exploratory Data Analysis

- After loading the data, it's a good practice to see if there are any missing values in the data.
- We count the number of missing values for each feature using `isnull()`.

In [67]:
```
# Return numerical summary of each attribute of iris
iris_df.describe()
```

Out[67]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
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

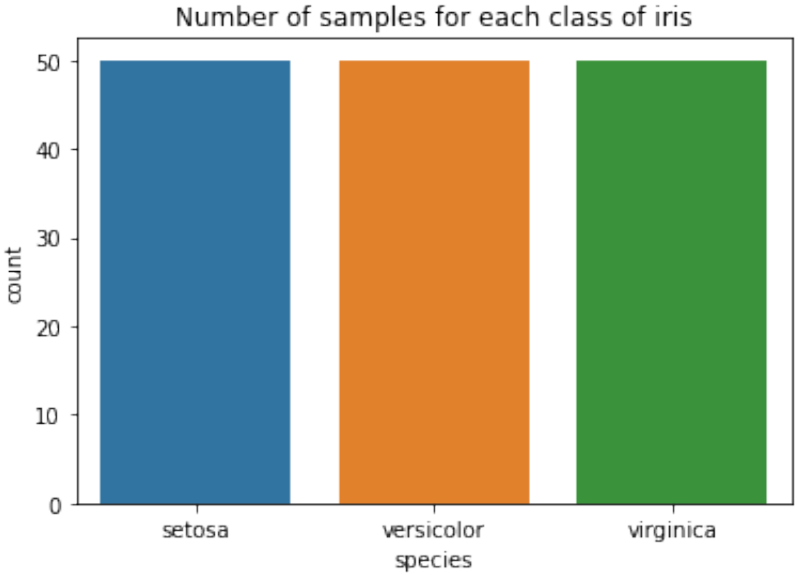
In [68]:
```
# let's check number of samples for each class of iris

iris_df.groupby('species').size()
```

Out[68]:
```
species
setosa        50
versicolor    50
virginica     50
dtype: int64
```

- As we can see that each class has the same number of instances.

In [69]:
```
# let's visualise the number of samples for each class with count plot
sns.countplot(x='species', data=iris_df)
plt.title("Number of samples for each class of iris");
```

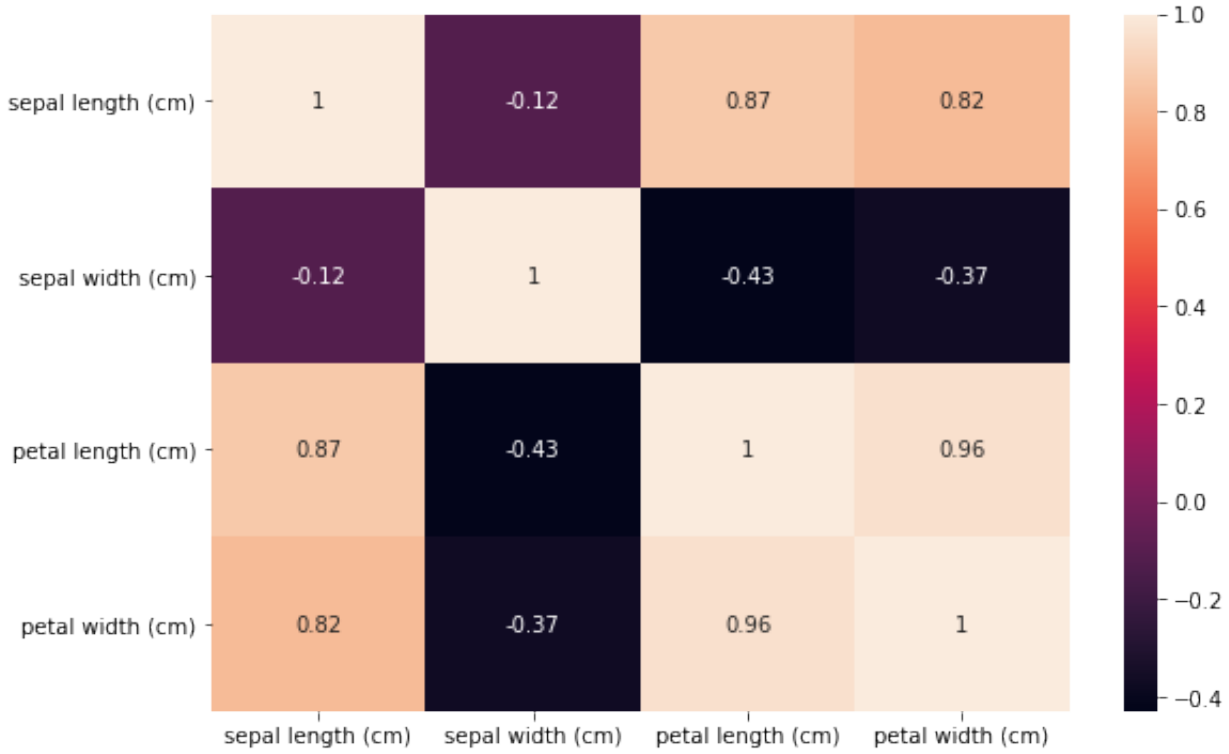


- Next, let's make a correlation matrix to quantitatively examine the relationship between variables.
- If there are features and many of the features are highly correlated, then training an algorithm with all the featues will reduce the accuracy. Thus features selection should be done carefully. This dataset has less featues but still we will see the correlation.
- The correlation matrix can be formed by using the `corr` function from the pandas library.
- The correlation coefficient ranges from `-1` to `1`. If the value is close to `1`, it means that there is a *strong positive correlation* between the two variables. When it is close to `-1`, the variables have a *strong negative correlation*.
- Then, we will use the `heatmap` function from the seaborn library to plot the correlation matrix.

In [70]:

```
# corr() to calculate the correlation between variables
correlation_matrix = iris_df.corr().round(2)

# changing the figure size
plt.figure(figsize = (9, 6))
# "annot = True" to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True);
```

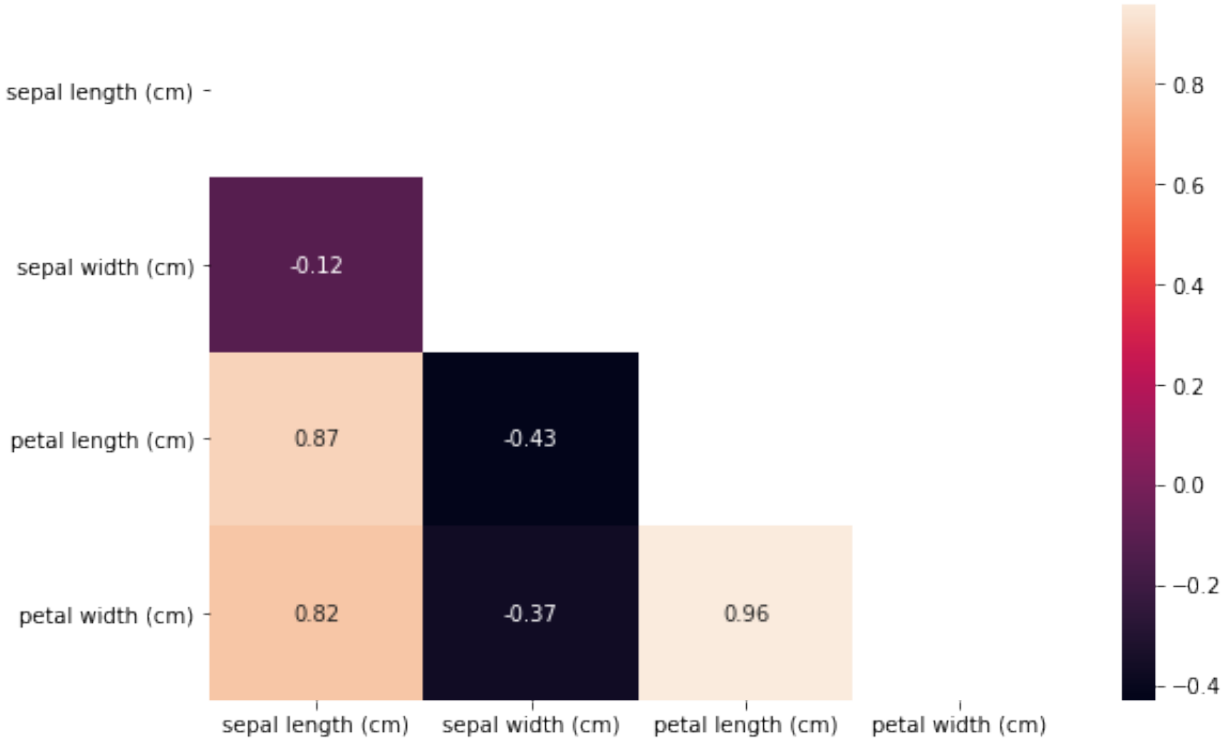


In [71]:

```
# Steps to remove redundant values

# Return a array filled with zeros
mask = np.zeros_like(correlation_matrix)
# Return the indices for the upper-triangle of array
mask[np.triu_indices_from(mask)] = True

# changing the figure size
plt.figure(figsize = (9, 6))
# "annot = True" to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True, mask=mask);
```

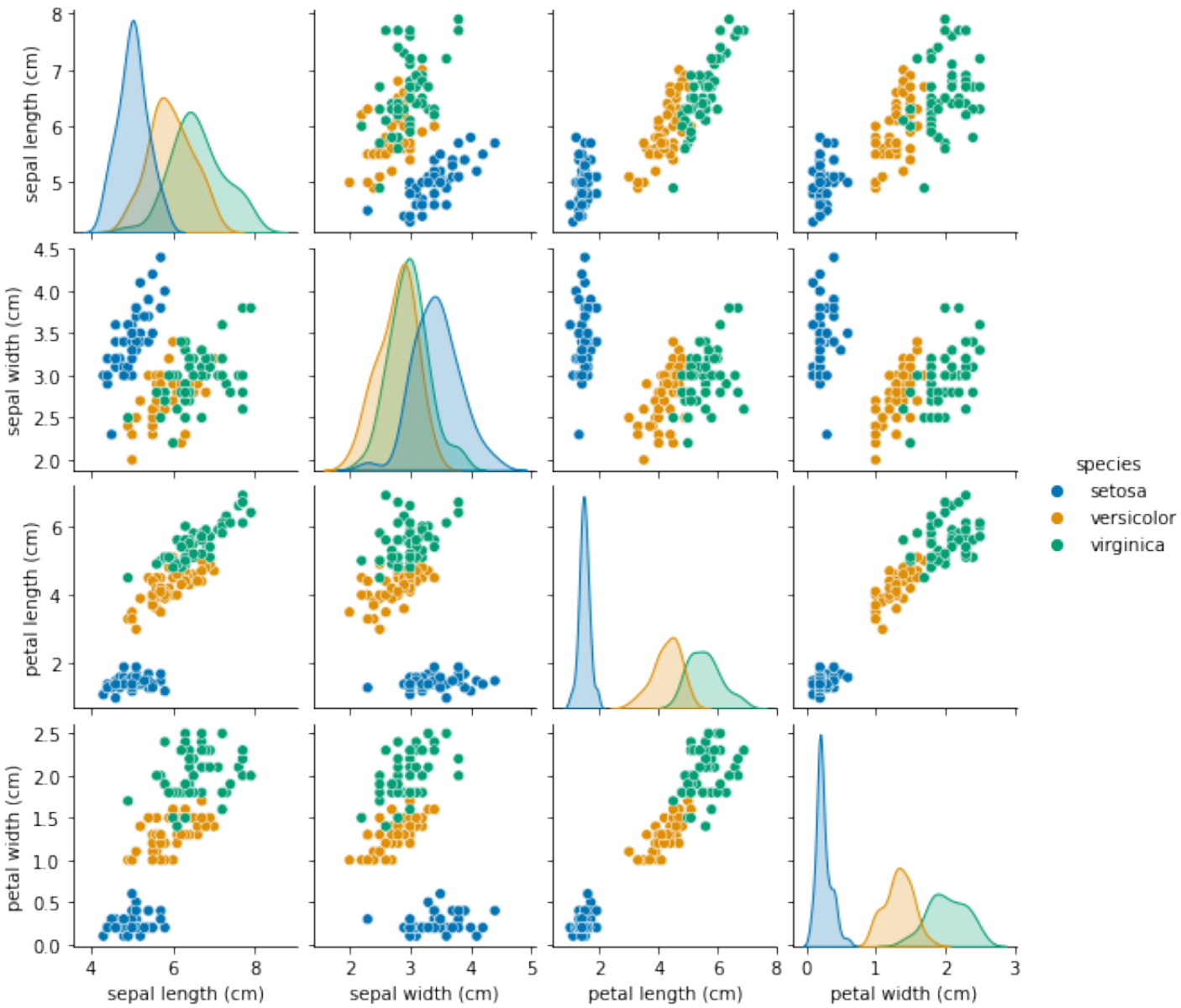


- As we can see, the Sepal Width and Length are not correlated but the Petal Width and Length are highly correlated.
- Also, note that the petal features have relatively high correlation with sepal_length, but not with sepal_width.

In [72]:

```
# let's create pairplot to visualise the data for each pair of attributes

sns.pairplot(iris_df, hue="species", height = 2, palette = 'colorblind');
```

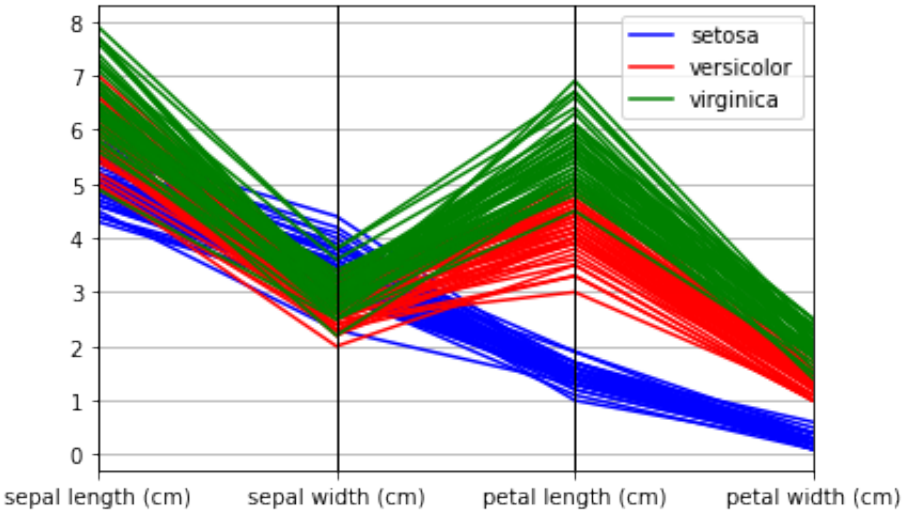


- As we can see from the above figure, iris `setosa` is separated from both other species in all the features.
- Besides, the Petal Features are giving a better cluster division compared to the Sepal features (that means, the petal measurements separate the different species better than the sepal ones).
- This is an indication that the Petals can help in better and accurate Predictions over the Sepal.

- For this dataset, another useful visualization plot is **parallel coordinate**, which represents each row as a line.
- A parallel plot plot allows to compare the feature of several individual observations (series) on a set of numeric variables. Interestingly, Pandas is probably the best way to plot a parallel coordinate plot with python.

In [73]:
```
from pandas.plotting import parallel_coordinates

parallel_coordinates(iris_df, "species", color = ['blue', 'red', 'green']);
```



- As we have seen before, petal measurements can separate species better than the sepal ones.

Create Features Matrix & Target Variable

In [74]:
```
# Feature matrix
X = iris_df[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']]
X
```

Out[74]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
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 [75]:
```
# Target variable
y = iris_df['species']
y
```

Out[75]:
```
0         setosa
1         setosa
2         setosa
3         setosa
4         setosa
         ...
145    virginica
146    virginica
147    virginica
148    virginica
149    virginica
Name: species, Length: 150, dtype: object
```

Split the dataset

In [116…

```
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 = 16)

print("X_train shape: ", X_train.shape)
print("X_test shape: ", X_test.shape)
print("y_train shape: ", y_train.shape)
print("y_test shape: ", y_test.shape)
```

```
X_train shape:  (105, 4)
X_test shape:  (45, 4)
y_train shape:  (105,)
y_test shape:  (45,)
```

Create Model: Support Vector Machine (SVM)

In [151…

```
# importing the necessary package to use the classification algorithm
from sklearn import svm  #for Support Vector Machine (SVM) Algorithm

model_svm = svm.SVC() #select the algorithm

model_svm.fit(X_train, y_train) #train the model with the training dataset

y_prediction_svm = model_svm.predict(X_test) # pass the testing data to the trained model

# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_svm = metrics.accuracy_score(y_prediction_svm, y_test).round(4)

print("----------------------------------")
print('The accuracy of the SVM is: {}'.format(score_svm))
print("----------------------------------")

# save the accuracy score
score = set()
score.add(('SVM', score_svm))
```

```
----------------------------------
The accuracy of the SVM is: 0.9556
----------------------------------
```

- SVM is giving very good accuracy

Create Model: Decision Tree

In [152…

```
# importing the necessary package to use the classification algorithm
from sklearn.tree import DecisionTreeClassifier #for using Decision Tree Algoithm

model_dt = DecisionTreeClassifier(random_state=4)

model_dt.fit(X_train, y_train) #train the model with the training dataset

y_prediction_dt = model_dt.predict(X_test) #pass the testing data to the trained model

# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_dt = metrics.accuracy_score(y_prediction_dt, y_test).round(4)

print("---------------------------------")
print('The accuracy of the DT is: {}'.format(score_dt))
print("---------------------------------")

# save the accuracy score
score.add(('DT', score_dt))
```

```
---------------------------------
The accuracy of the DT is: 0.9333
---------------------------------
```

Create Model: K Nearest Neighbours (KNN)

In [153…

```
# importing the necessary package to use the classification algorithm
from sklearn.neighbors import KNeighborsClassifier  # for K nearest neighbours

#from sklearn.linear_model import LogisticRegression  # for Logistic Regression algorithm

model_knn = KNeighborsClassifier(n_neighbors=3) # 3 neighbours for putting the new data into a class

model_knn.fit(X_train, y_train) #train the model with the training dataset

y_prediction_knn = model_knn.predict(X_test) #pass the testing data to the trained model

# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output

score_knn = metrics.accuracy_score(y_prediction_knn, y_test).round(4)

print("----------------------------------")
print('The accuracy of the KNN is: {}'.format(score_knn))
print("----------------------------------")

# save the accuracy score
score.add(('KNN', score_knn))
```

```
----------------------------------
The accuracy of the KNN is: 0.9556
----------------------------------
```

Create Model: Logistic Regression

In [154…

```
# importing the necessary package to use the classification algorithm
from sklearn.linear_model import LogisticRegression  # for Logistic Regression algorithm

model_lr = LogisticRegression()

model_lr.fit(X_train, y_train) #train the model with the training dataset

y_prediction_lr = model_lr.predict(X_test) #pass the testing data to the trained model

# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_lr = metrics.accuracy_score(y_prediction_lr, y_test).round(4)

print("---------------------------------")
print('The accuracy of the LR is: {}'.format(score_lr))
print("---------------------------------")

# save the accuracy score
score.add(('LR', score_lr))
```

```
---------------------------------
The accuracy of the LR is: 0.9333
---------------------------------
```

Create Model: Naive Bayes

In [155…

```
# importing the necessary package to use the classification algorithm
from sklearn.naive_bayes import GaussianNB

model_nb = GaussianNB()

model_nb.fit(X_train, y_train) #train the model with the training dataset

y_prediction_nb = model_nb.predict(X_test) #pass the testing data to the trained model

# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_nb = metrics.accuracy_score(y_prediction_nb, y_test).round(4)

print("---------------------------------")
print('The accuracy of the NB is: {}'.format(score_nb))
print("---------------------------------")

# save the accuracy score
score.add(('NB', score_nb))
```

```
---------------------------------
The accuracy of the NB is: 0.9111
---------------------------------
```

Compare Accuracy Score of Different Models

In [156…

```
print("The accuracy scores of different Models:")
print("----------------------------------------")

for s in score:
    print(s)
```

```
The accuracy scores of different Models:
----------------------------------------
('NB', 0.9111)
('LR', 0.9333)
('SVM', 0.9556)
('KNN', 0.9556)
('DT', 0.9333)
```

- As we can see that all the models have accuracy score more than `91%`.

This lecture belongs to **Programming in Python** course, given by **Dr Akinul Islam Jony.**