DS Major Project September

In this project we have used Iris data set and add a KNN model to it and check the EDA(Exploratory Data Analysis) and applied a s able Classifier,Regressor or Clusterer and calculate the accuracy of the model.

Done by Bhavin Baldota

1.1 Data set - Iris

The iris dataset is a classic and very easy multi-class classification dataset.

Classes ----------------- 3

Samples per class ----------------- 50

Samples total ----------------- 150

Dimensionality ----------------- 4

Features ----------------- real, positive

1.2 Algorithm - KNN Model

K-nearest neighbors (kNN) is a supervised machine learning algorithm that can be used to solve both classification and regression tasks. I see kNN as an algorithm that comes from real life. People tend to be effected by the people around them. Our behaviour is guided by the friends we grew up with. Our parents also shape our personality in some ways. If you grow up with people who love sports, it is higly likely that you will end up loving sports. There are ofcourse exceptions. kNN works similarly.

The value of a data point is determined by the data points around it.

If you have one very close friend and spend most of your time with him/her, you will end up sharing similar interests and enjoying same things. That is kNN with k=1. If you always hang out with a group of 5, each one in the group has an effect on your behavior and you will end up being the average of 5. That is kNN with k=5. kNN classifier determines the class of a data point by majority voting principle. If k is set to 5, the classes of 5 closest points are checked. Prediction is done according to the majority class. Similarly, kNN regression takes the mean value of 5 closest points.

1.3 Algorithm steps

STEP 1: Cgoose the number K of neighbors

STEP 2: Take the K nearest neighbors of the new data point, according to your distance metric

STEP 3: Among these K neighbors, count the number of data points to each category

STEP 4: Assign the new data point to the category where you counted the most neighbors

2. Importing and preperation of data

2.1 Import libraries

In [ ]:
```
import numpy as np
import pandas as pd
```

2.2 Load dataset

NOTE: Iris dataset includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

In [ ]:
```
# Importing the dataset
dataset = pd.read_csv('../input/Iris.csv')
```

2.3 Summarize the Dataset

In [ ]:
```
# We can get a quick idea of how many instances (rows) and how many attributes (column
dataset.shape
```

Out[ ]: (150, 6)

In [ ]:
```
dataset.head(5)
```

Out[ ]:

	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 [ ]:
```
dataset.describe()
```

Out[ ]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [ ]:

```
# Let’s now take a look at the number of instances (rows) that belong to each class. W
dataset.groupby('Species').size()
```

Out[ ]:

```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

2.4 Dividing data into features and labels

NOTE: As we can see dataset contain six columns: Id, SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm and Species. The actual features are described by columns 1-4. Last column contains labels of samples. Firstly we need to split data into two arrays: X (features) and y (labels).

In [ ]:

```
feature_columns = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm','PetalWidthCm']
X = dataset[feature_columns].values
y = dataset['Species'].values

# Alternative way of selecting features and labels arrays:
# X = dataset.iloc[:, 1:5].values
# y = dataset.iloc[:, 5].values
```

2.5 Label encoding

NOTE: As we can see labels are categorical. KNeighborsClassifier does not accept string labels. We need to use LabelEncoder to transform them into numbers. Iris-setosa correspond to 0, Iris-versicolor correspond to 1 and Iris-virginica correspond to 2.

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

2.6 Spliting dataset into training set and test set

Let's split dataset into training set and test set, to check later on whether or not our classifier works correctly.

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_stat
```

Lastly, because features values are in the same order of magnitude, there is no need for feature scaling. Nevertheless in other sercostamses it is extremly important to apply feature scaling before running classification algorythms.

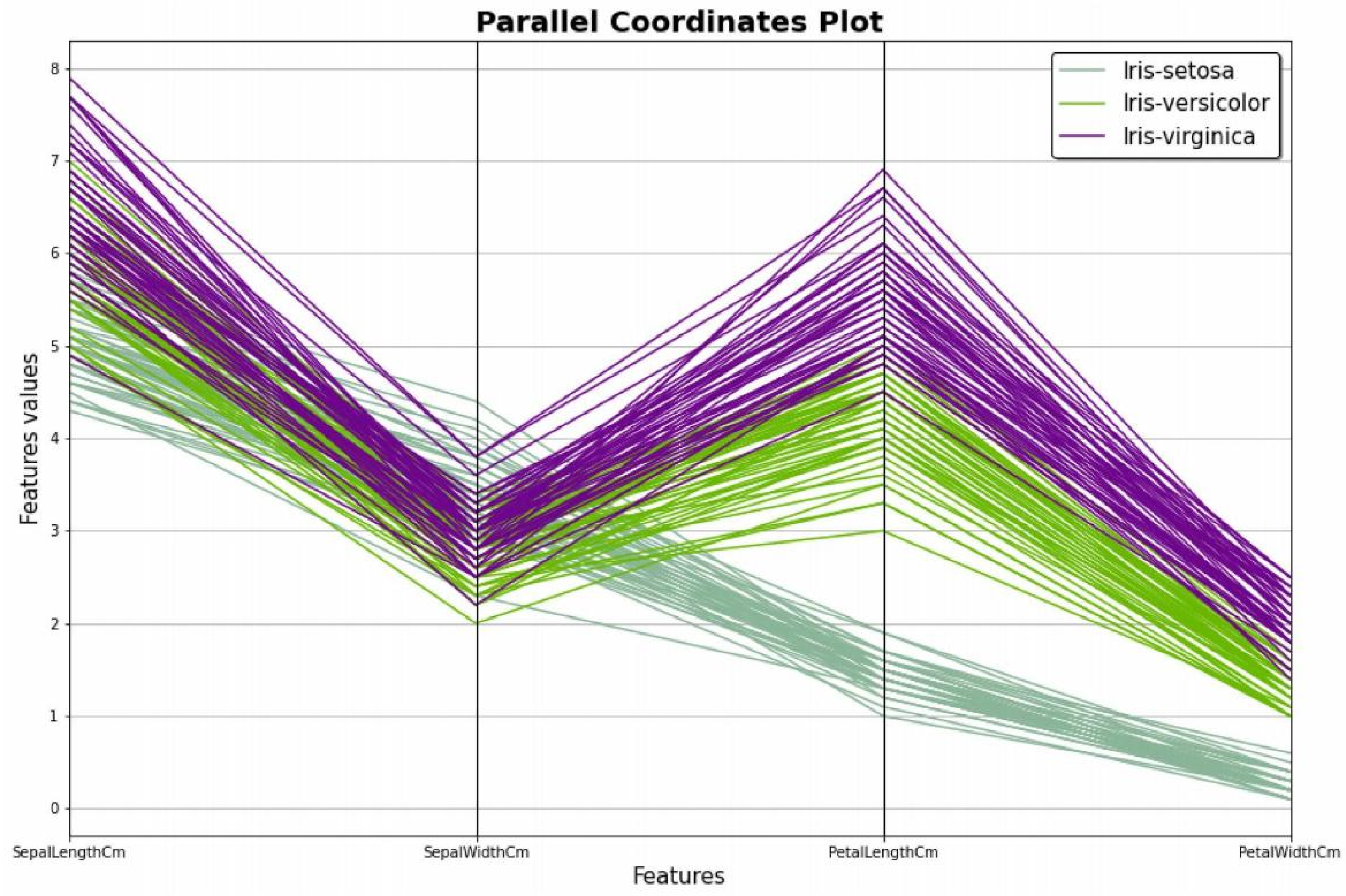
3. Data Visualization

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

3.1. Parallel Coordinates

Parallel coordinates is a plotting technique for plotting multivariate data. It allows one to see clusters in data and to estimate other statistics visually. Using parallel coordinates points are represented as connected line segments. Each vertical line represents one attribute. One set of connected line segments represents one data point. Points that tend to cluster will appear closer together.

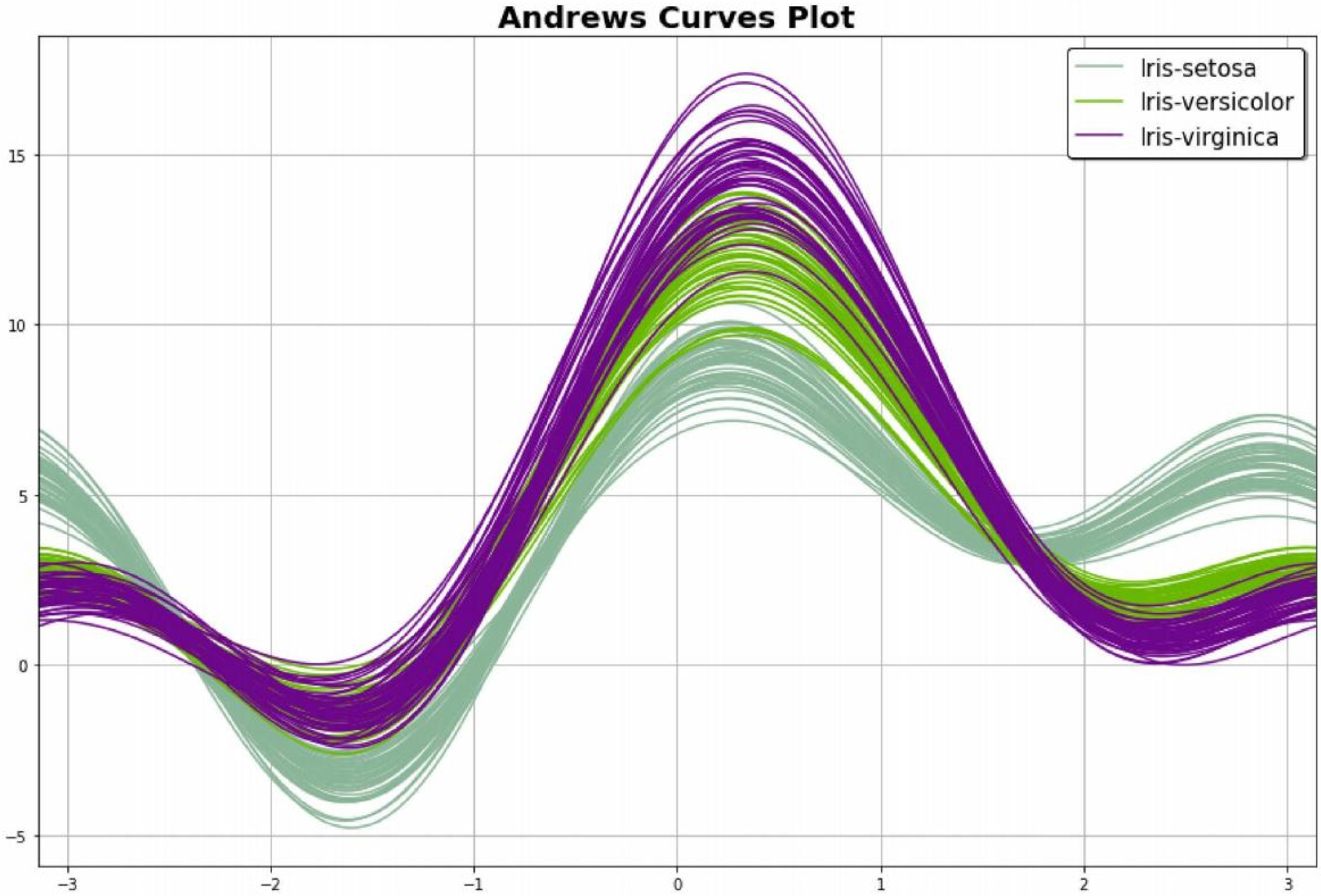
In [ ]:
```
from pandas.plotting import parallel_coordinates
plt.figure(figsize=(15,10))
parallel_coordinates(dataset.drop("Id", axis=1), "Species")
plt.title('Parallel Coordinates Plot', fontsize=20, fontweight='bold')
plt.xlabel('Features', fontsize=15)
plt.ylabel('Features values', fontsize=15)
plt.legend(loc=1, prop={'size': 15}, frameon=True,shadow=True, facecolor="white", edge
plt.show()
```



3.2. Andrews Curves

Andrews curves allow one to plot multivariate data as a large number of curves that are created using the attributes of samples as coefficients for Fourier series. By coloring these curves differently for each class it is possible to visualize data clustering. Curves belonging to samples of the same class will usually be closer together and form larger structures.

In [ ]:
```
from pandas.plotting import andrews_curves
plt.figure(figsize=(15,10))
andrews_curves(dataset.drop("Id", axis=1), "Species")
plt.title('Andrews Curves Plot', fontsize=20, fontweight='bold')
plt.legend(loc=1, prop={'size': 15}, frameon=True,shadow=True, facecolor="white", edge
plt.show()
```

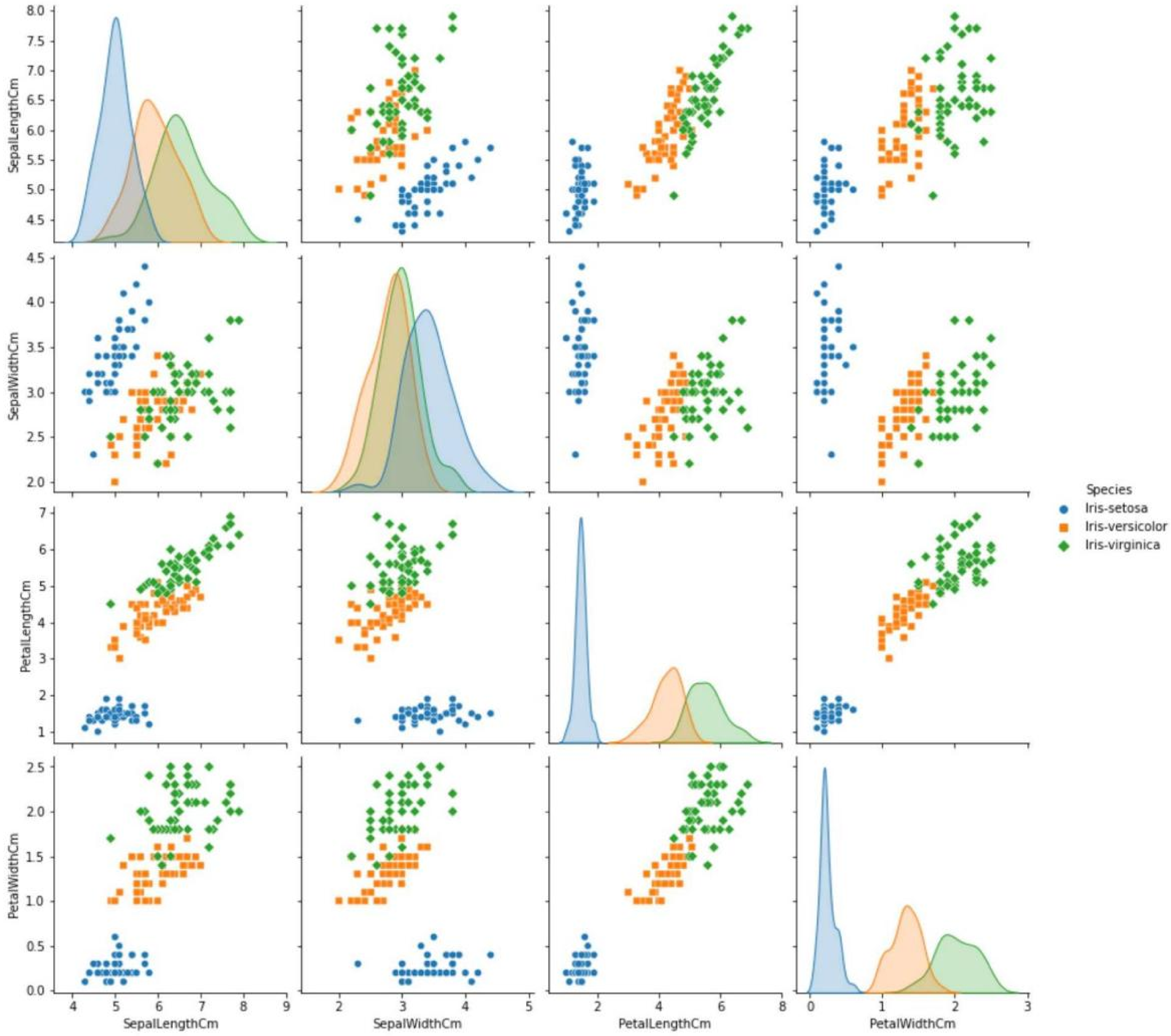


3.3. Pairplot

Pairwise is useful when you want to visualize the distribution of a variable or the relationship between multiple variables separately within subsets of your dataset.

In [ ]:
```
plt.figure()
sns.pairplot(dataset.drop("Id", axis=1), hue = "Species", size=3, markers=["o", "s", "
plt.show()
```

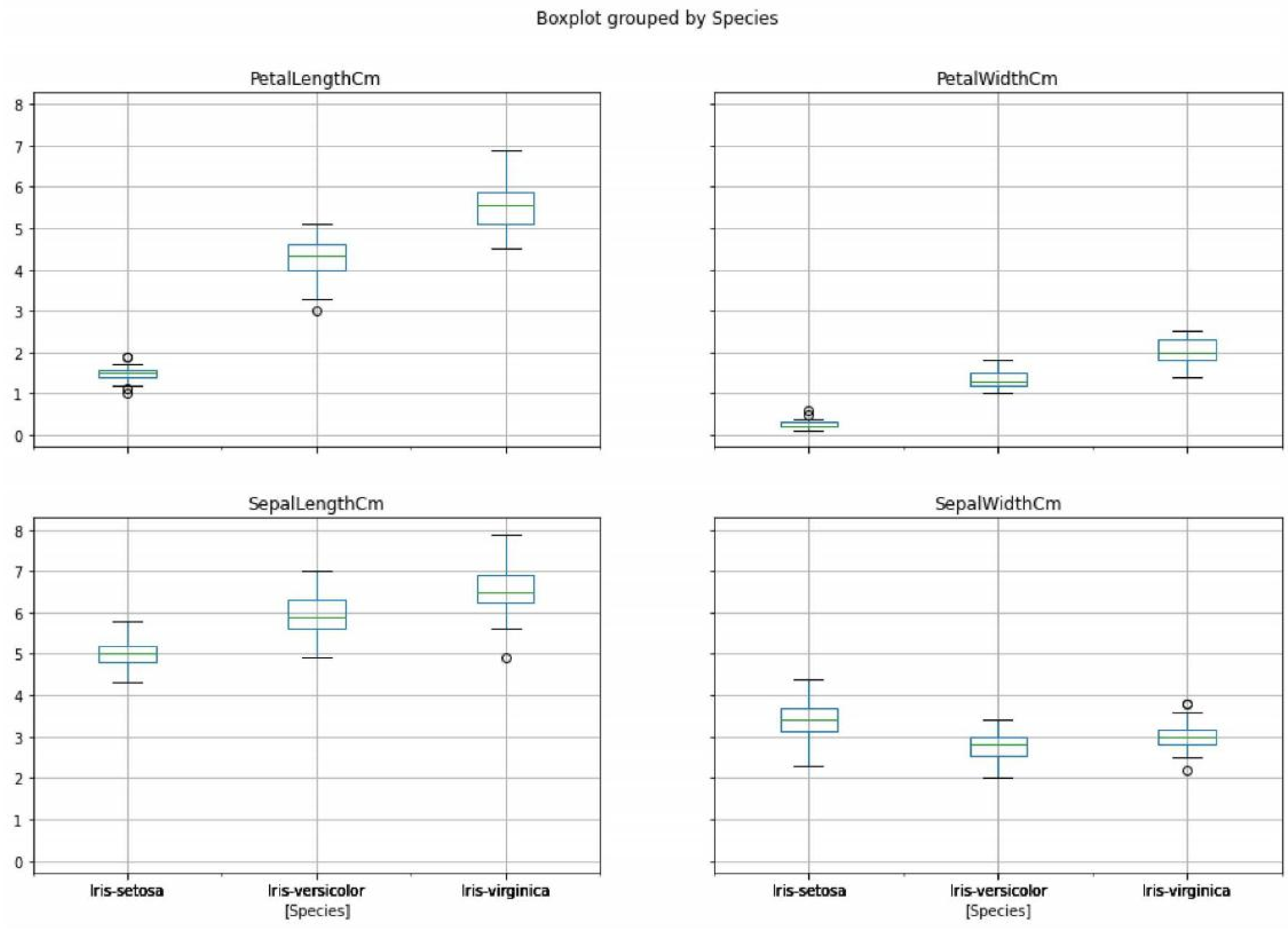
```
/opt/conda/lib/python3.7/site-packages/seaborn/axisgrid.py:2076: UserWarning: The `si
ze` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
<Figure size 432x288 with 0 Axes>
```



3.4. Boxplots

In [ ]:
```
plt.figure()
dataset.drop("Id", axis=1).boxplot(by="Species", figsize=(15, 10))
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```

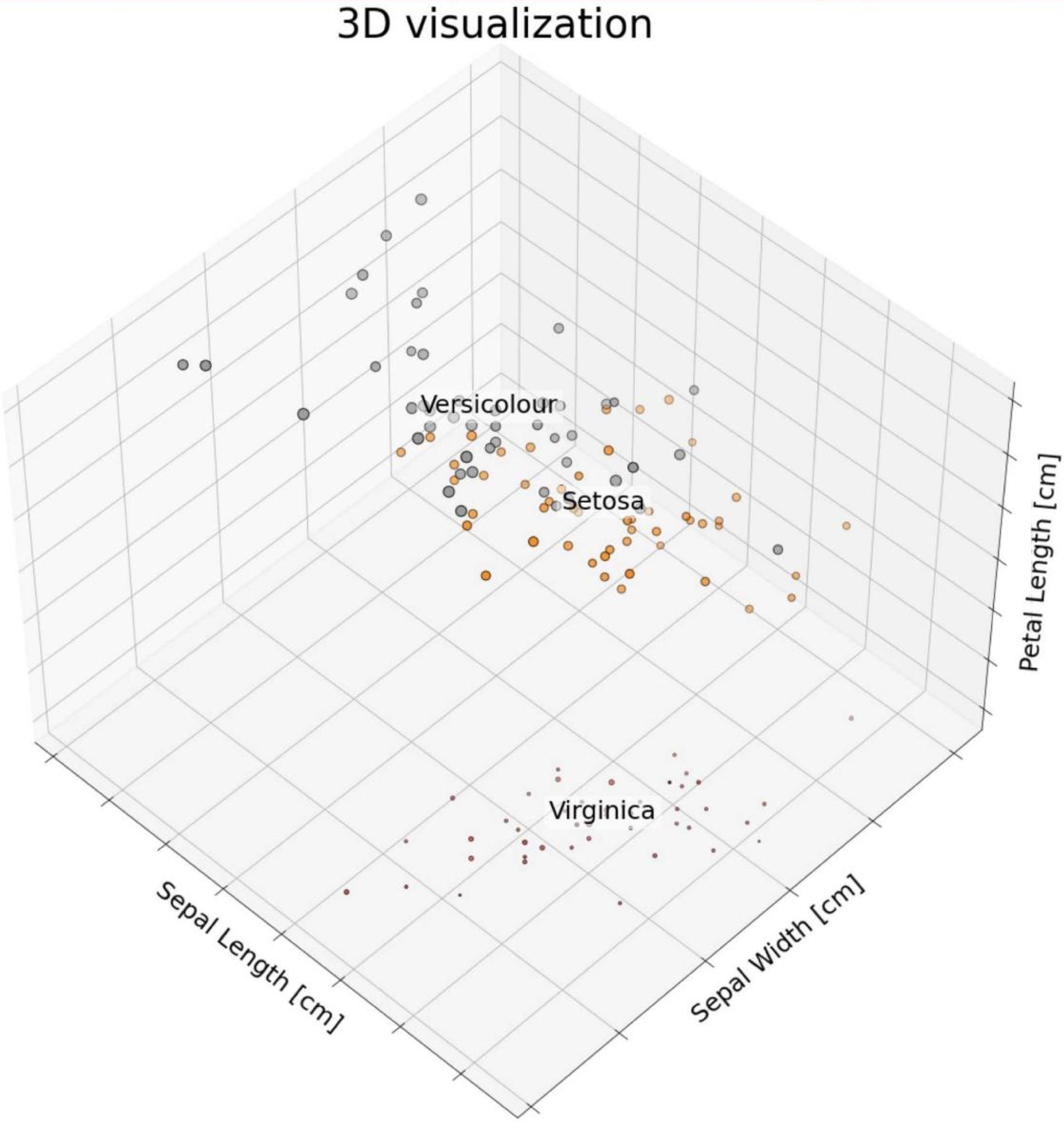


3.5. 3D visualization

You can also try to visualize high-dimensional datasets in 3D using color, shape, size and other properties of 3D and 2D objects. In this plot I used marks sizes to visualize fourth dimenssion which is Petal Width [cm].

In [ ]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(1, figsize=(20, 15))
ax = Axes3D(fig, elev=48, azim=134)
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y,
           cmap=plt.cm.Set1, edgecolor='k', s = X[:, 3]*50)

for name, label in [('Virginica', 0), ('Setosa', 1), ('Versicolour', 2)]:
    ax.text3D(X[y == label, 0].mean(),
              X[y == label, 1].mean(),
              X[y == label, 2].mean(), name,
              horizontalalignment='center',
              bbox=dict(alpha=.5, edgecolor='w', facecolor='w'),size=25)

ax.set_title("3D visualization", fontsize=40)
ax.set_xlabel("Sepal Length [cm]", fontsize=25)
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("Sepal Width [cm]", fontsize=25)
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("Petal Length [cm]", fontsize=25)
ax.w_zaxis.set_ticklabels([])

plt.show()
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:3: MatplotlibDeprecation
Warning: Axes3D(fig) adding itself to the figure is deprecated since 3.4. Pass the ke
yword argument auto_add_to_figure=False and use fig.add_axes(ax) to suppress this war
ning. The default value of auto_add_to_figure will change to False in mpl3.5 and True
values will no longer work in 3.6.  This is consistent with other Axes classes.
  This is separate from the ipykernel package so we can avoid doing imports until
```



4. Using KNN for classification

4.1. Making predictions

In [ ]:
```
# Fitting clasifier to the Training set
# Loading libraries
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score

# Instantiate learning model (k = 3)
classifier = KNeighborsClassifier(n_neighbors=3)
```

```
# Fitting the model
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

4.2. Evaluating predictions

Building confusion matrix:

In [ ]:
```
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[ ]:
```
array([[11,  0,  0],
       [ 0, 12,  1],
       [ 0,  0,  6]])
```

Calculating model accuracy:

In [ ]:
```
accuracy = accuracy_score(y_test, y_pred)*100
print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
```

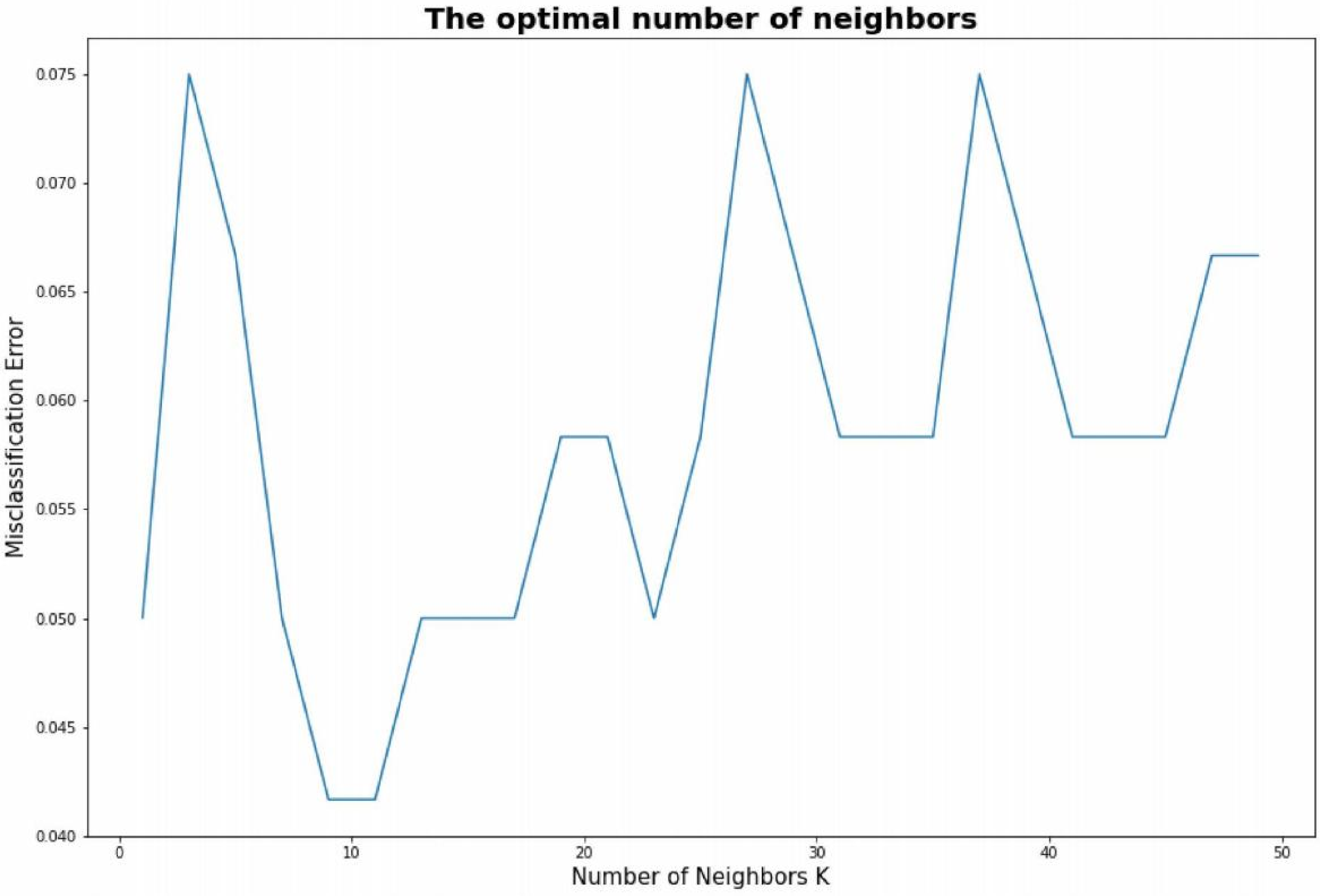
```
Accuracy of our model is equal 96.67 %.
```

4.3. Using cross-validation for parameter tuning:

In [ ]:
```
# creating list of K for KNN
k_list = list(range(1,50,2))
# creating list of cv scores
cv_scores = []

# perform 10-fold cross validation
for k in k_list:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())
```

In [ ]:
```
# changing to misclassification error
MSE = [1 - x for x in cv_scores]

plt.figure()
plt.figure(figsize=(15,10))
plt.title('The optimal number of neighbors', fontsize=20, fontweight='bold')
plt.xlabel('Number of Neighbors K', fontsize=15)
plt.ylabel('Misclassification Error', fontsize=15)
sns.set_style("whitegrid")
plt.plot(k_list, MSE)

plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [ ]:
```
# finding best k
best_k = k_list[MSE.index(min(MSE))]
print("The optimal number of neighbors is %d." % best_k)
```

```
The optimal number of neighbors is 9.
```

5. My own KNN implementation

In [ ]:
```
import numpy as np
import pandas as pd
import scipy as sp

class MyKNeighborsClassifier():
    """
    My implementation of KNN algorithm.
    """

    def __init__(self, n_neighbors=5):
        self.n_neighbors=n_neighbors

    def fit(self, X, y):
        """
        Fit the model using X as array of features and y as array of labels.
        """
        n_samples = X.shape[0]
        # number of neighbors can't be larger then number of samples
        if self.n_neighbors > n_samples:
            raise ValueError("Number of neighbors can't be larger then number of sampl

        # X and y need to have the same number of samples
        if X.shape[0] != y.shape[0]:
```

```
            raise ValueError("Number of samples in X and y need to be equal.")

        # finding and saving all possible class labels
        self.classes_ = np.unique(y)

        self.X = X
        self.y = y

    def predict(self, X_test):

        # number of predictions to make and number of features inside single sample
        n_predictions, n_features = X_test.shape

        # allocationg space for array of predictions
        predictions = np.empty(n_predictions, dtype=int)

        # loop over all observations
        for i in range(n_predictions):
            # calculation of single prediction
            predictions[i] = single_prediction(self.X, self.y, X_test[i, :], self.n_ne

        return(predictions)
```

In [ ]:

```
def single_prediction(X, y, x_train, k):

    # number of samples inside training set
    n_samples = X.shape[0]

    # create array for distances and targets
    distances = np.empty(n_samples, dtype=np.float64)

    # distance calculation
    for i in range(n_samples):
        distances[i] = (x_train - X[i]).dot(x_train - X[i])

    # combining arrays as columns
    distances = sp.c_[distances, y]
    # sorting array by value of first column
    sorted_distances = distances[distances[:,0].argsort()]
    # celecting labels associeted with k smallest distances
    targets = sorted_distances[0:k,1]

    unique, counts = np.unique(targets, return_counts=True)
    return(unique[np.argmax(counts)])
```

In [ ]:

```
# Instantiate learning model (k = 3)
my_classifier = MyKNeighborsClassifier(n_neighbors=3)

# Fitting the model
my_classifier.fit(X_train, y_train)

# Predicting the Test set results
my_y_pred = my_classifier.predict(X_test)
```

In [ ]:

```
accuracy = accuracy_score(y_test, my_y_pred)*100
print('Accuracy of our model is equal ' + str(round(accuracy, 2)) + ' %.')
```

```
Accuracy of our model is equal 96.67 %.
```