```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


import warnings
warnings.filterwarnings('ignore')


plt.style.use("fivethirtyeight")
%matplotlib inline
```

```
df=pd.read_csv('/content/Iris.csv')
df.head()
```

	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

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.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
```

```
df.describe()

```

	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

```
df.shape

```

```
(150, 6)
```

```
df.drop('Id',axis=1,inplace=True)

```

```
1 df['Species'].value_counts()
2
```

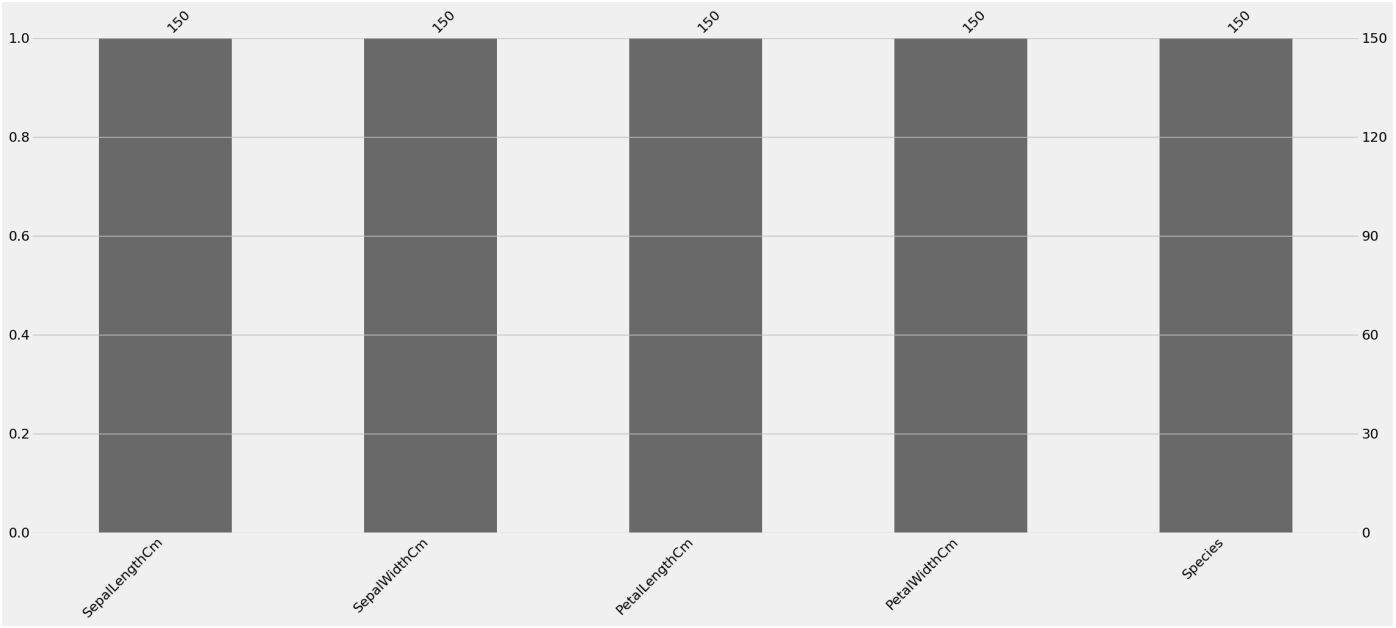
Species	count
Iris-setosa	50
Iris-versicolor	50
Iris-virginica	50

```
1 df.isnull().sum()
2
```

	0
SepalLengthCm	0
SepalWidthCm	0
PetalLengthCm	0
PetalWidthCm	0
Species	0

```
1 import missingno as msno
2 msno.bar(df)
```

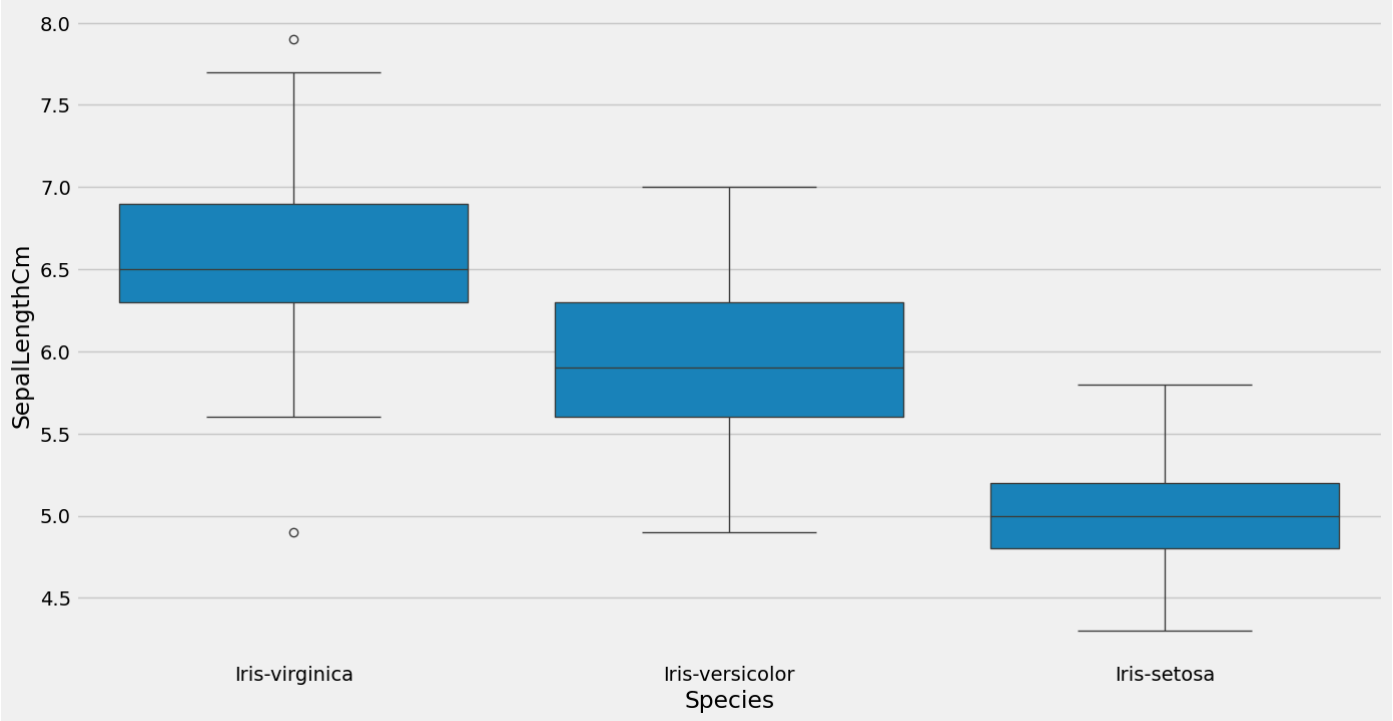
<Axes: >




```
1 df.drop_duplicates(inplace=True)
2
```

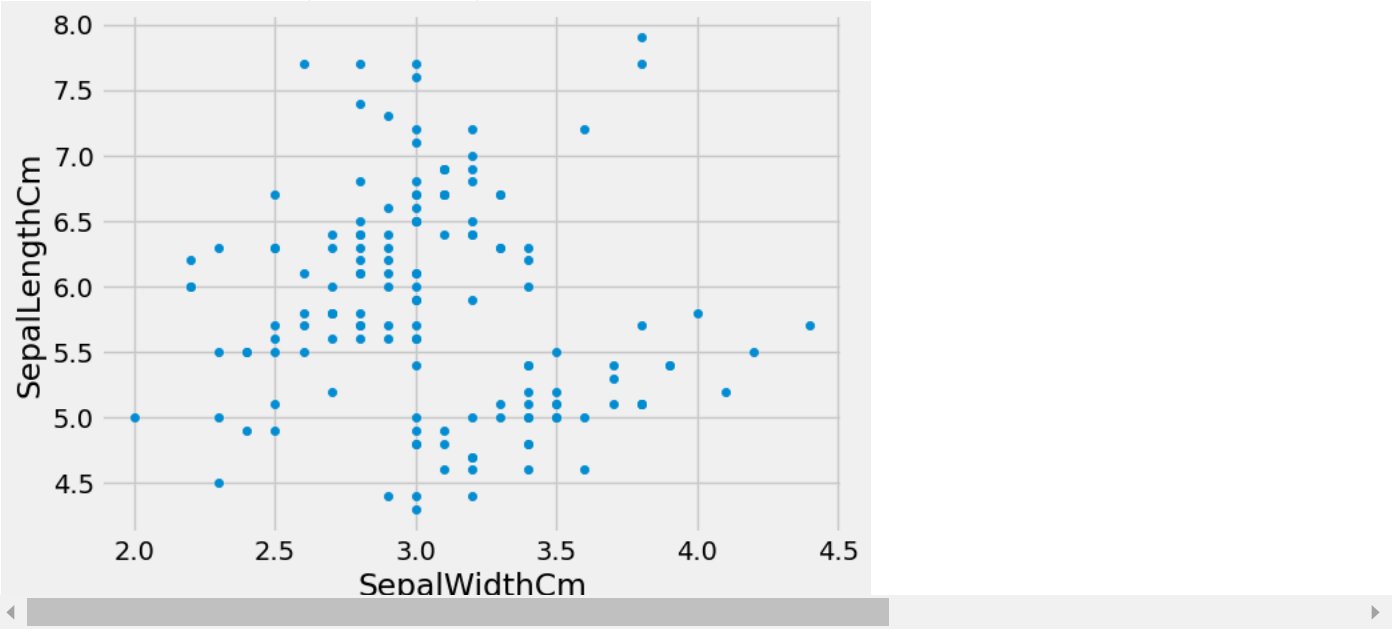
```
1 plt.figure(figsize=(15,8))
2 sns.boxplot(x='Species',y='SepalLengthCm',data=df.sort_values('SepalLengthCm',ascending=False))
```

```
<Axes: xlabel='Species', ylabel='SepalLengthCm'>
```



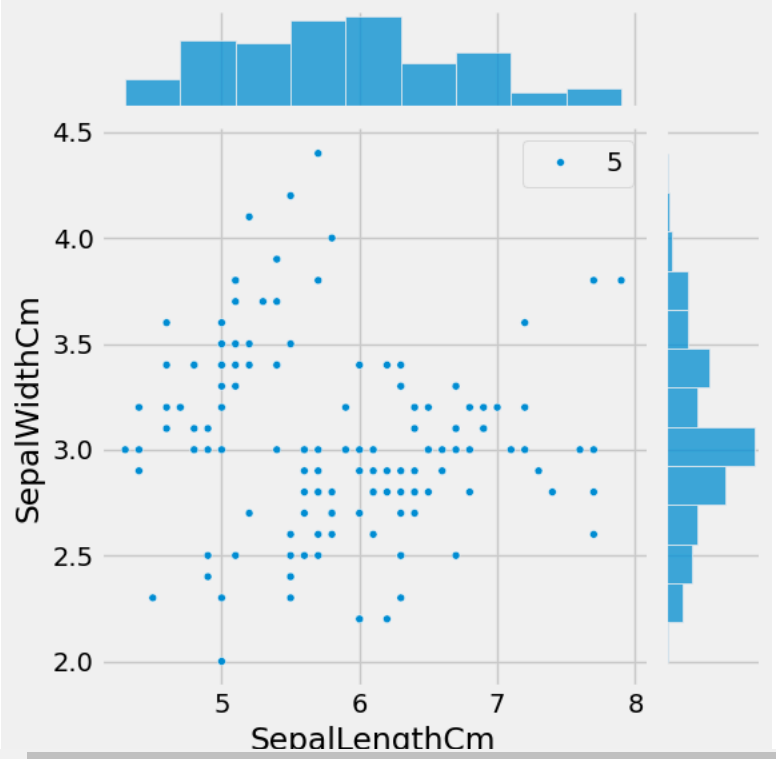

```
1 df.plot(kind='scatter',x='SepalWidthCm',y='SepalLengthCm')
2
```

```
<Axes: xlabel='SepalWidthCm', ylabel='SepalLengthCm'>
```



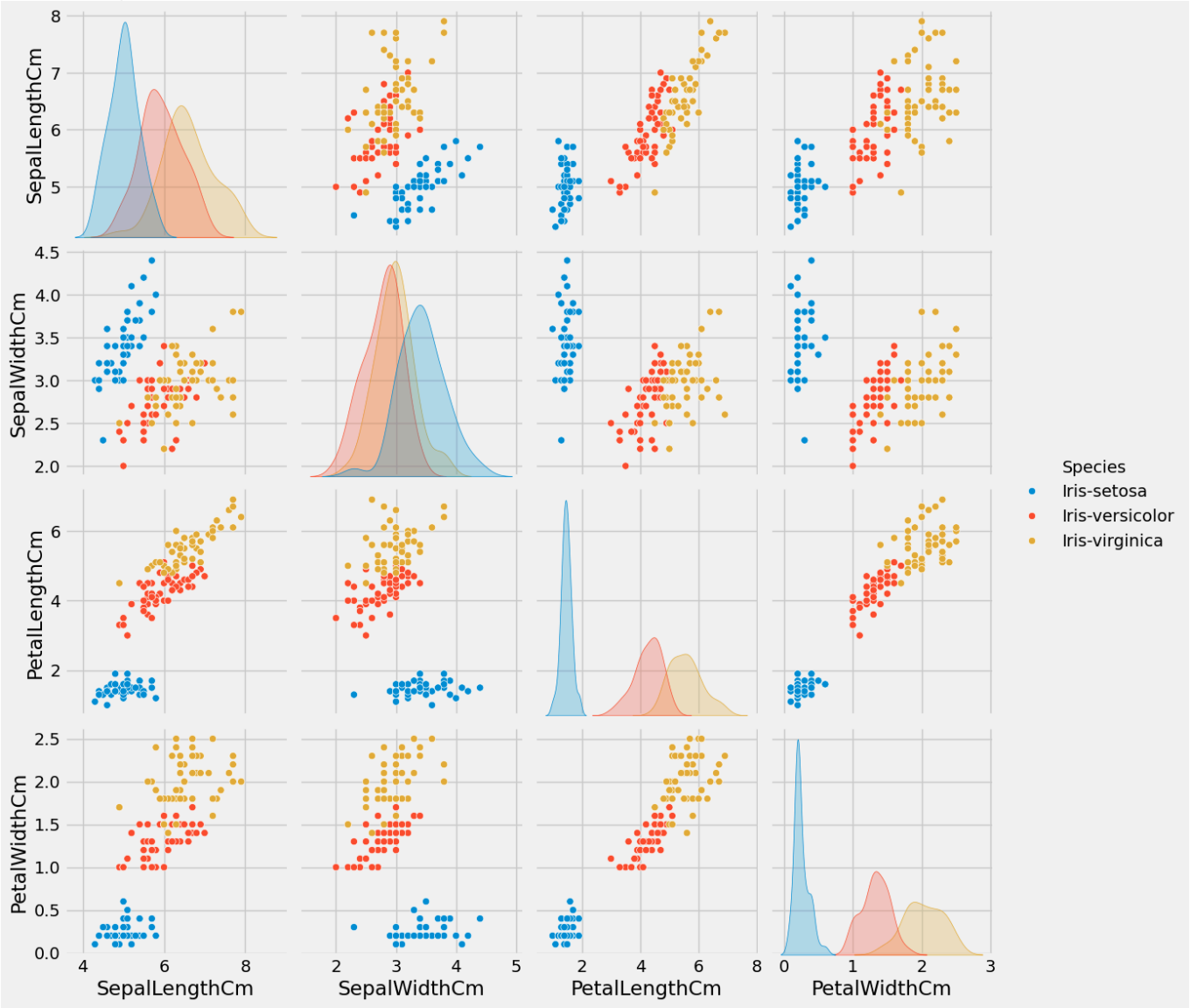

```
1 sns.jointplot(x="SepalLengthCm", y="SepalWidthCm", data=df, size=5)
2
```

```
<seaborn.axisgrid.JointGrid at 0x7e9d2a7cf490>
```



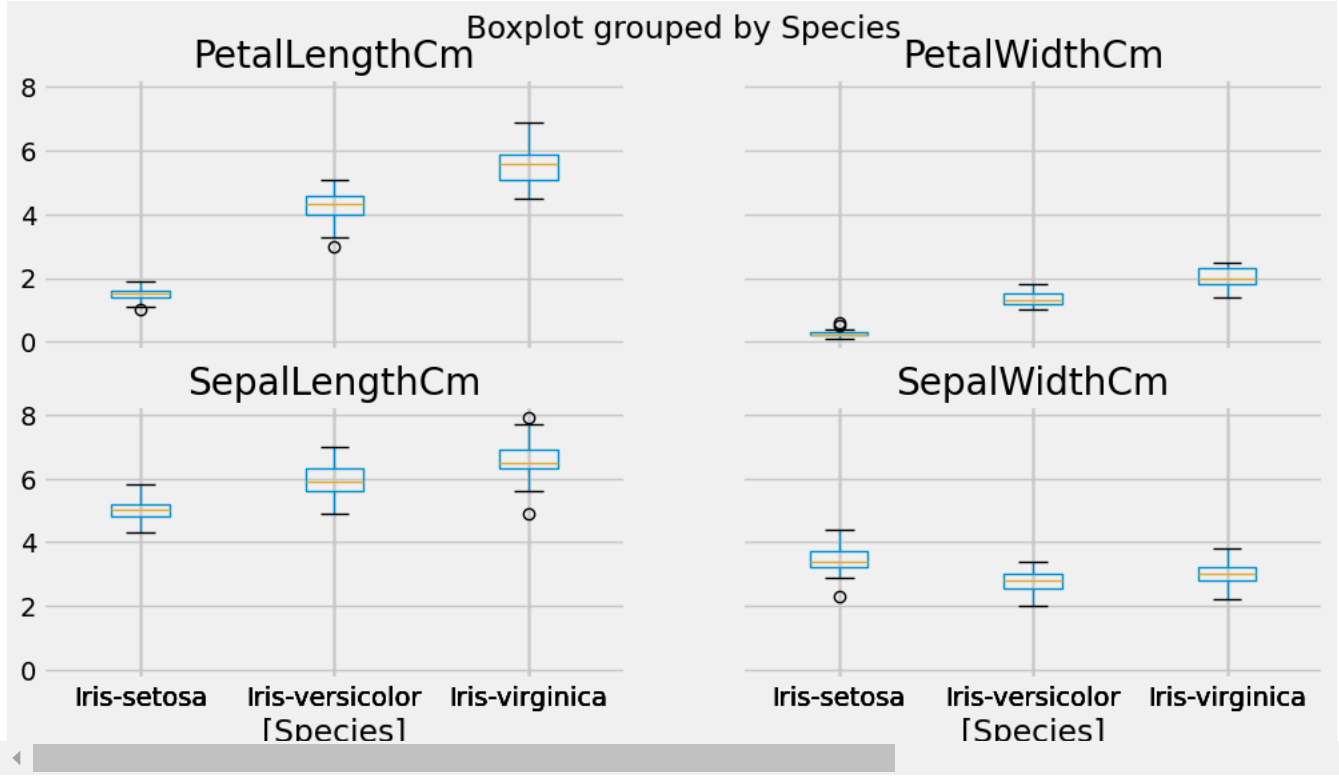

```
1 sns.pairplot(df, hue="Species", size=3)
2
```

```
<seaborn.axisgrid.PairGrid at 0x7e9d28d276a0>
```



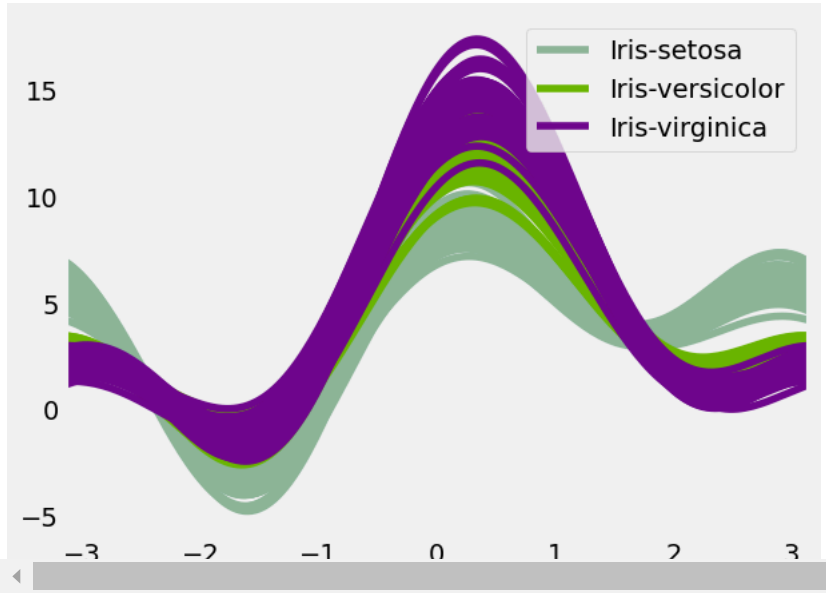

```
1 df.boxplot(by="Species", figsize=(12, 6))
2
```

```
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)
```



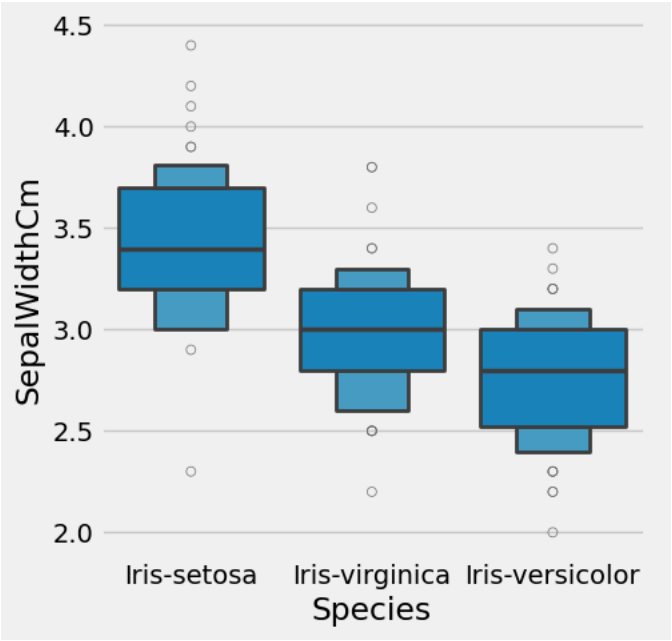

```
1 import pandas.plotting
2 from pandas.plotting import andrews_curves
3 andrews_curves(df, "Species")
```

```
<Axes: >
```



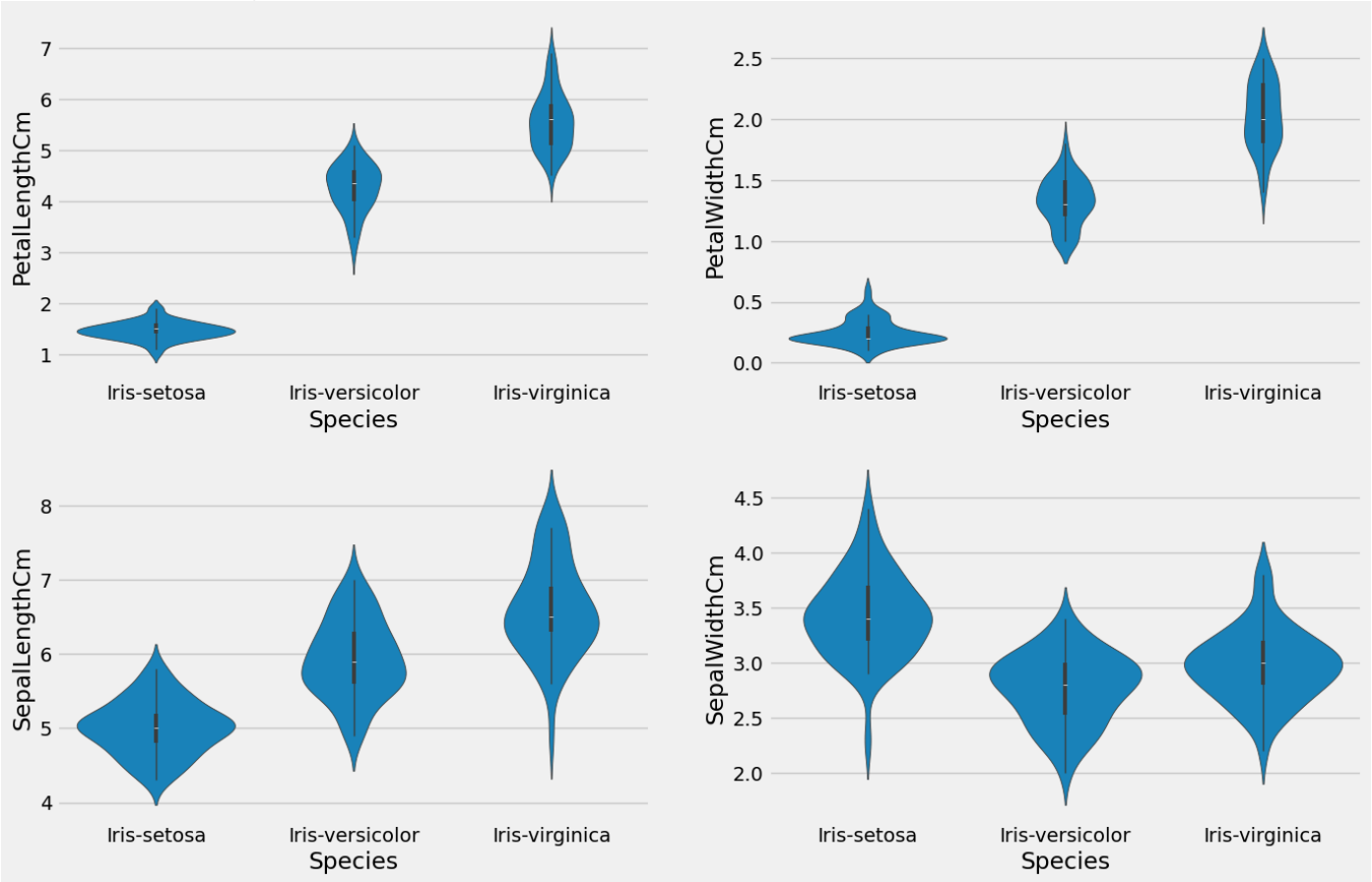

```
1 plt.figure(figsize=(15,15))
2 sns.catplot(x='Species',y='SepalWidthCm',data=df.sort_values('SepalWidthCm',ascending=False),kind='boxen')
```

```
<seaborn.axisgrid.FacetGrid at 0x7e9d2606fb50>
<Figure size 1500x1500 with 0 Axes>
```




```
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.violinplot(x='Species',y='PetalLengthCm',data=df)
plt.subplot(2,2,2)
sns.violinplot(x='Species',y='PetalWidthCm',data=df)
plt.subplot(2,2,3)
sns.violinplot(x='Species',y='SepalLengthCm',data=df)
plt.subplot(2,2,4)
sns.violinplot(x='Species',y='SepalWidthCm',data=df)
```

```
<Axes: xlabel='Species', ylabel='SepalWidthCm'>
```




```
X=df.drop('Species',axis=1)
y=df['Species']
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

```
df['Species'] = pd.Categorical(df.Species)
df['Species'] = df.Species.cat.codes
# Turn response variable into one-hot response vectory = to_categorical(df.response)
y = to_categorical(df.Species)
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.30,stratify=y,random_state=123)
```

```
model=Sequential()
model.add(Dense(100,activation='relu',input_shape=(4,)))

model.add(Dense(3,activation='softmax'))
```

```
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

```
history=model.fit(X_train,y_train,epochs=45,validation_data=(X_test, y_test))
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.8685 - loss: 0.5058 - val_accuracy: 0.8222 - val_loss: 0.5102
Epoch 21/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9262 - loss: 0.4973 - val_accuracy: 0.9111 - val_loss: 0.4990
Epoch 22/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9632 - loss: 0.5093 - val_accuracy: 0.9778 - val_loss: 0.4915
Epoch 23/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9575 - loss: 0.4816 - val_accuracy: 0.9778 - val_loss: 0.4827
Epoch 24/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9512 - loss: 0.4907 - val_accuracy: 1.0000 - val_loss: 0.4712
Epoch 25/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.9789 - loss: 0.4815 - val_accuracy: 0.9111 - val_loss: 0.4612
Epoch 26/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9311 - loss: 0.4578 - val_accuracy: 0.8667 - val_loss: 0.4544
Epoch 27/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.8900 - loss: 0.4486 - val_accuracy: 0.8444 - val_loss: 0.4475
Epoch 28/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - accuracy: 0.8991 - loss: 0.4462 - val_accuracy: 0.8667 - val_loss: 0.4395
Epoch 29/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9145 - loss: 0.4281 - val_accuracy: 0.9111 - val_loss: 0.4322
Epoch 30/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9539 - loss: 0.4452 - val_accuracy: 0.9333 - val_loss: 0.4258
Epoch 31/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9789 - loss: 0.4102 - val_accuracy: 0.9111 - val_loss: 0.4195
Epoch 32/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9747 - loss: 0.4141 - val_accuracy: 0.9111 - val_loss: 0.4134
Epoch 33/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9601 - loss: 0.4387 - val_accuracy: 0.9778 - val_loss: 0.4081
Epoch 34/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9653 - loss: 0.4023 - val_accuracy: 0.9778 - val_loss: 0.4028
Epoch 35/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.9604 - loss: 0.3708 - val_accuracy: 0.9778 - val_loss: 0.3987
Epoch 36/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9632 - loss: 0.3889 - val_accuracy: 0.9778 - val_loss: 0.3938
Epoch 37/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - accuracy: 0.9718 - loss: 0.3829 - val_accuracy: 0.9333 - val_loss: 0.3955
Epoch 38/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9298 - loss: 0.3786 - val_accuracy: 0.9333 - val_loss: 0.3929
Epoch 39/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - accuracy: 0.9330 - loss: 0.3752 - val_accuracy: 0.9778 - val_loss: 0.3790
Epoch 40/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9757 - loss: 0.3561 - val_accuracy: 0.9778 - val_loss: 0.3725
Epoch 41/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9632 - loss: 0.3710 - val_accuracy: 0.9778 - val_loss: 0.3677
Epoch 42/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9716 - loss: 0.3535 - val_accuracy: 0.9333 - val_loss: 0.3629
Epoch 43/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.9695 - loss: 0.3660 - val_accuracy: 0.9778 - val_loss: 0.3585
Epoch 44/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - accuracy: 0.9664 - loss: 0.3532 - val_accuracy: 0.9778 - val_loss: 0.3543
Epoch 45/45
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9664 - loss: 0.3474 - val_accuracy: 0.9333 - val_loss: 0.3501
```

```
1 model.evaluate(X_test,y_test)
2
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9347 - loss: 0.3464
[0.3501420021057129, 0.9333333373069763]
```

```
1 pred = model.predict(X_test[:10])
2 print(pred)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 69ms/step
[[0.00409993 0.2921109  0.70378923]
 [0.00334097 0.29247147 0.7041875 ]
 [0.05933839 0.60630274 0.33435893]
 [0.03337201 0.5517102  0.41491777]
 [0.9470508  0.04667047 0.00627864]
 [0.03037986 0.60066277 0.36895737]
 [0.00333107 0.2219175  0.77475154]
 [0.00393734 0.24097247 0.7550902 ]
 [0.94911987 0.04532756 0.00555256]
 [0.0109275  0.36407247 0.62500006]]
```

```
1 p=np.argmax(pred,axis=1)
2 print(p)
3 print(y_test[:10])
```

```
[2 2 1 1 0 1 2 2 0 2]
[[0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 1. 0.]
```

```
       [1. 0. 0.]
       [0. 1. 0.]
       [0. 0. 1.]
       [0. 0. 1.]
       [1. 0. 0.]
       [0. 0. 1.]]
```

```
1 history.history['accuracy']
2
```

```
[0.3333333432674408,
 0.36274510622024536,
 0.3333333432674408,
 0.5098039507865906,
 0.656862735748291,
 0.8725489974021912,
 0.9313725233078003,
 0.813725471496582,
 0.9313725233078003,
 0.9313725233078003,
 0.8529411554336548,
 0.7941176295280457,
 0.7941176295280457,
 0.8725489974021912,
 0.9509803652763367,
 0.970588207244873,
 0.970588207244873,
 0.9509803652763367,
 0.9215686321258545,
 0.843137264251709,
 0.9117646813392639,
 0.970588207244873,
 0.9509803652763367,
 0.9509803652763367.
```