In [1]:

**import** numpy **as** np **import** pandas **as** pd **import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt sns**.**set(style**=**"white",color\_codes**=True**) **import** warnings warnings**.**filterwarnings("ignore") **import** os

# Understanding the Data

In [2]:

iris**=**pd**.**read\_csv("Iris.csv.2.csv") iris**.**head()

## The basic description of the data, in terms of basic mathematics.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Out[2]: |  | 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 [3]:

iris**.**describe()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Out[3]: | 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 |

# Analysing the data visually

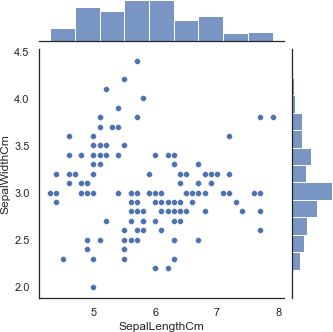
## At the outset , let us look at a simple scatter plot, to get a visual feel of the data.

In [19]:

sns**.**jointplot(x**=**"SepalLengthCm",y**=**"SepalWidthCm",data**=**iris,size**=**5)

Out[19]:

<seaborn.axisgrid.JointGrid at 0x18271621fd0>



In [20]:

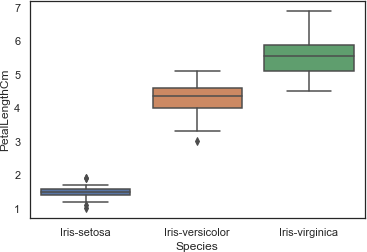
sns**.**boxplot(x**=**"Species",y**=**"PetalLengthCm",data**=**iris)

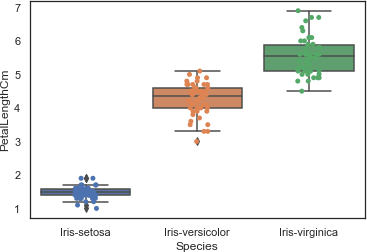
Out[20]:

In [21]:

ax**=**sns**.**boxplot(x**=**"Species",y**=**"PetalLengthCm",data**=**iris) ax**=**sns**.**stripplot(x**=**"Species",y**=**"PetalLengthCm",data**=**iris, jitter**=True**,edgecol

<AxesSubplot:xlabel='Species', ylabel='PetalLengthCm'>





In [22]:

sns**.**violinplot(x**=**"Species",y**=**"PetalLengthCm",data**=**iris,size**=**6)

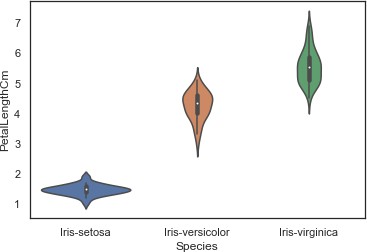
# This is a special plot called violin plot

A Violin plot combines the benefits of the previous two plots and simplifies them

# Denser regions of the data are fatter, and sparser thiner in a violin plot

Out[22]:

<AxesSubplot:xlabel='Species', ylabel='PetalLengthCm'>



# Pairplot

## Another useful seaborn plot is hybrid plot called pairplot,which shows the bivariate relation between each pair of features.

In [23]:

# From the pairplot, we'll see that Iris-setosa species is separated from the other

two across all feature combinations

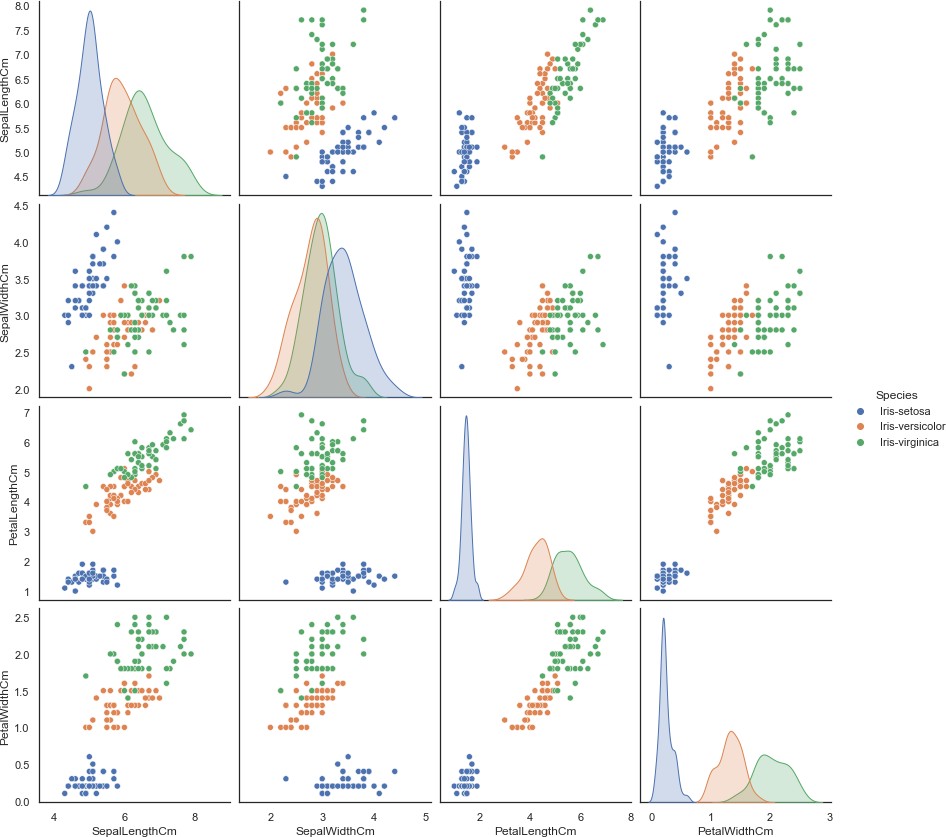
sns**.**pairplot(iris**.**drop("Id",axis**=**1),hue**=**"Species",size**=**3)

Out[23]:

In [24]:

iris**.**drop("Id",axis**=**1)**.**boxplot(by**=**"Species",figsize**=**(13,5))

<seaborn.axisgrid.PairGrid at 0x182742dc790>



# Box plot grid

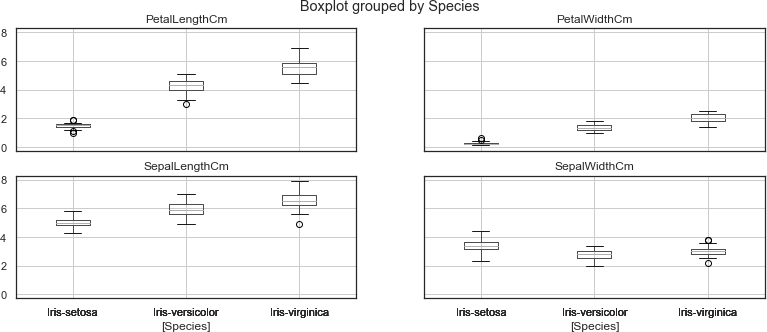
Out[24]:

array([[<AxesSubplot:title={'center':'PetalLengthCm'}, xlabel='[Species]'>,

<AxesSubplot:title={'center':'PetalWidthCm'}, xlabel='[Species]'>], [<AxesSubplot:title={'center':'SepalLengthCm'}, xlabel='[Species]'>,

<AxesSubplot:title={'center':'SepalWidthCm'}, xlabel='[Species]'>]], dtype=object)

14



## one cool more sophisticated technique pandas has available is called Andrews Curves

In [25]:

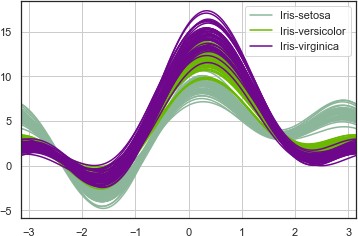
**from** pandas.plotting **import** andrews\_curves andrews\_curves(iris**.**drop("Id",axis**=**1),"Species")

# Andrews Curves involves using attributes of samples as coefficients for Fourier series

and then plotting these

Out[25]:

<AxesSubplot:>



# Another multivariate visualization technique pandas has is parallel\_coordinates

Parallel coordinates plots each feature on a separate column & then draws lines

In [26]:

**from** pandas.plotting **import** parallel\_coordinates parallel\_coordinates(iris**.**drop("Id",axis**=**1),"Species")

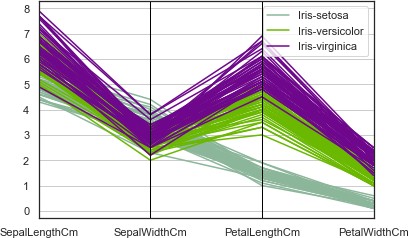
# connecting the features for each data sample

Out[26]:

In [27]:

**from** pandas.plotting **import** radviz radviz(iris**.**drop("Id",axis**=**1),"Species")

<AxesSubplot:>



A final multivariate visualization technique

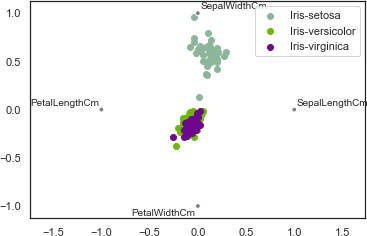
# Which puts each feature as a point on a 2D plane, and then simulates

having each sample attached to those points through a spring weighted

# by the relative value for that feature

Out[27]:

<AxesSubplot:>



In [28]:

**from** sklearn.metrics **import** confusion\_matrix

**from** sklearn.metrics **import** classification\_report

**from** sklearn.model\_selection **import** train\_test\_split

In [29]:

*#Separating the data into dependent and independent variables*

x**=**iris**.**iloc[:,:**-**1]**.**values y**=**iris**.**iloc[:,**-**1]**.**values

*#Splitting the dataset into the Training set and Test set*

**from** sklearn.model\_selection **import** train\_test\_split x\_train,x\_test,y\_test,y\_test**=**train\_test\_split(x,y,test\_size**=**0.2,random\_state**=**

# Training the model

## Using some of the commonly used algorithms, we will be training our model to check how accurate every algorithm is. we will be implementing these algorithms to compare:

1. Logistic Regression

## k-Nearest Neighbours(KNN)

1. Support Vector Machine(SVM)
2. Decision Treesaive Bayes classifier

## The first algorithm Logical Regression we can build our model like :

In [ ]:

17

**In [28]:**

#LogisticRegression

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression()

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

# Summary of the predictions made by the classifier

print(classification\_report(y\_test, y\_pred))

print(confusion\_matrix(y\_test, y\_pred))

# Accuracy score

from sklearn.metrics import accuracy\_score

print('accuracy is',accuracy\_score(y\_pred,y\_test))

Out[28]:

precision recall f1-score support

Iris-setosa 1.00 1.00 1.00 11

Iris-versicolor 1.00 0.85 0.92 13

Iris-virginica 0.75 1.00 0.86 6

accuracy 0.93 30

macro avg 0.92 0.95 0.92 30

weighted avg 0.95 0.93 0.94 30

[ [11 0 0]

[0 11 2]

[0 0 6]]

accuracy is 0.9333333333333333

Now , let us see the scores with **K-Nearest Neighbors** technique.

In [31]:

#K-Nearest Neighbours

from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n\_neighbors=8)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

# Summary of the predictions made by the classifier

print(classification\_report(y\_test, y\_pred))

print(confusion\_matrix(y\_test, y\_pred))

18

#Accuracy score

from sklearn.metrics import accuracy\_score

print('accuracy is',accuracy\_score(y\_pred,y\_test))

Out [31]:

precision recall f1-score support

Iris-setosa 1.00 1.00 1.00 11

Iris-versicolor 1.00 1.00 1.00 13

Iris-virginica 1.00 1.00 1.00 6

accuracy 1.00 30

macro avg 1.00 1.00 1.00 30

weighted avg 1.00 1.00 1.00 30

[[11 0 0]

[ 0 13 0]

[ 0 0 6]]

accuracy is 1.0

Thirdly , with **SVM (Support Vector Machines)**.

In [34]:

#Support Vector Machine's

from sklearn.svm import SVC

classifier = SVC()

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

# Summary of the predictions made by the classifier

print(classification\_report(y\_test, y\_pred))

print(confusion\_matrix(y\_test, y\_pred))

# Accuracy score

from sklearn.metrics import accuracy\_score

print('accuracy is',accuracy\_score(y\_pred,y\_test))

19

Out [34]:

precision recall f1-score support

Iris-setosa 1.00 1.00 1.00 11

Iris-versicolor 1.00 1.00 1.00 13

Iris-virginica 1.00 1.00 1.00 6

accuracy 1.00 30

macro avg 1.00 1.00 1.00 30

weighted avg 1.00 1.00 1.00 30

[[11 0 0]

[ 0 13 0]

[ 0 0 6]]

accuracy is 1.0

In [35]:

# Decision Tree's

from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier()

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

# Summary of the predictions made by the classifier

print(classification\_report(y\_test, y\_pred))

print(confusion\_matrix(y\_test, y\_pred))

# Accuracy score

from sklearn.metrics import accuracy\_score

print('accuracy is',accuracy\_score(y\_pred,y\_test))

Out[35]:

precision recall f1-score support

Iris-setosa 1.00 1.00 1.00 11

Iris-versicolor 0.93 1.00 0.96 13

Iris-virginica 1.00 0.83 0.91 6

accuracy 0.97 30

macro avg 0.98 0.94 0.96 30

weighted avg 0.97 0.97 0.97 30

[[11 0 0]

[ 0 13 0]

[ 0 1 5]]

accuracy is 0.9666666666666667

20

In[36]:

# Gaussian Naive Bayes

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

# Summary of the predictions made by the classifier

print(classification\_report(y\_test, y\_pred))

print(confusion\_matrix(y\_test, y\_pred))

# Accuracy score

from sklearn.metrics import accuracy\_score

print('accuracy is',accuracy\_score(y\_pred,y\_test))

Out[36]:

precision recall f1-score support

Iris-setosa 1.00 1.00 1.00 11

Iris-versicolor 1.00 1.00 1.00 13

Iris-virginica 1.00 1.00 1.00 6

accuracy 1.00 30

macro avg 1.00 1.00 1.00 30

weighted avg 1.00 1.00 1.00 30

[[11 0 0]

[ 0 13 0]

[ 0 0 6]]

accuracy is 1.0

In[37]:

# Multinomial Naive Bayes

from sklearn.naive\_bayes import MultinomialNB

classifier = MultinomialNB()

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

# Summary of the predictions made by the classifier

print(classification\_report(y\_test, y\_pred))

print(confusion\_matrix(y\_test, y\_pred))

# Accuracy score

from sklearn.metrics import accuracy\_score

print('accuracy is',accuracy\_score(y\_pred,y\_test))

21

Out[37]:

precision recall f1-score support

Iris-setosa 1.00 0.91 0.95 11

Iris-versicolor 0.83 0.77 0.80 13

Iris-virginica 0.62 0.83 0.71 6

accuracy 0.83 30

macro avg 0.82 0.84 0.82 30

weighted avg 0.85 0.83 0.84 30

[[10 1 0]

[ 0 10 3]

[ 0 1 5]]

accuracy is 0.8333333333333334

In[38]:

# Bernoulli Naive Bayes

from sklearn.naive\_bayes import BernoulliNB

classifier = BernoulliNB()

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

# Summary of the predictions made by the classifier

print(classification\_report(y\_test, y\_pred))

print(confusion\_matrix(y\_test, y\_pred))

# Accuracy score

from sklearn.metrics import accuracy\_score

print('accuracy is',accuracy\_score(y\_pred,y\_test))

Out[38]:

precision recall f1-score support

Iris-setosa 0.00 0.00 0.00 11

Iris-versicolor 0.00 0.00 0.00 13

Iris-virginica 0.20 1.00 0.33 6

accuracy 0.20 30

macro avg 0.07 0.33 0.11 30

weighted avg 0.04 0.20 0.07 30

[[ 0 0 11]

[ 0 0 13]

[ 0 0 6]]

accuracy is 0.2

22

In[39]:

# Complement Naive Bayes

from sklearn.naive\_bayes import ComplementNB

classifier = ComplementNB()

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

# Summary of the predictions made by the classifier

print(classification\_report(y\_test, y\_pred))

print(confusion\_matrix(y\_test, y\_pred))

# Accuracy score

from sklearn.metrics import accuracy\_score

print('accuracy is',accuracy\_score(y\_pred,y\_test))

Out[39]:

precision recall f1-score support

Iris-setosa 0.69 1.00 0.81 11

Iris-versicolor 0.00 0.00 0.00 13

Iris-virginica 0.43 1.00 0.60 6

accuracy 0.57 30

macro avg 0.37 0.67 0.47 30

weighted avg 0.34 0.57 0.42 30

[[11 0 0]

[ 5 0 8]

[ 0 0 6]]

accuracy is 0.5666666666666667

In[40]:

from sklearn.metrics import accuracy\_score, log\_loss

classifiers = [

GaussianNB(),

MultinomialNB(),

BernoulliNB(),

ComplementNB(),

]

# Logging for Visual Comparison

log\_cols=["Classifier", "Accuracy", "Log Loss"]

log = pd.DataFrame(columns=log\_cols)

for clf **in** classifiers:

clf.fit(X\_train, y\_train)

name = clf.\_\_class\_\_.\_\_name\_\_

print("="\*30)

22

print(name)

print('\*\*\*\*Results\*\*\*\*')

train\_predictions = clf.predict(X\_test)

acc = accuracy\_score(y\_test, train\_predictions)

print("Accuracy: **{:.4%}**".format(acc))

log\_entry = pd.DataFrame([[name, acc\*100, 11]], columns=log\_cols)

log = log.append(log\_entry)

print("="\*30)

Out[40]:

==============================

GaussianNB

\*\*\*\*Results\*\*\*\*

Accuracy: 100.0000%

==============================

==============================

MultinomialNB

\*\*\*\*Results\*\*\*\*

Accuracy: 83.3333%

==============================

==============================

BernoulliNB

\*\*\*\*Results\*\*\*\*

Accuracy: 20.0000%

==============================

==============================

ComplementNB

\*\*\*\*Results\*\*\*\*

Accuracy: 56.6667%

==============================

23