Data Science Project Report

Predicting the species of the Iris dataset

Under the guidance
of
Mr. Sumit Kumar Shukla

Submitted by: Sharath C Nair
Rajadhani Institute of Engineering and Technology,
Trivandrum Kerala

EITHICAL EDUFABRICA (2020)

Contents

Acknowledgement

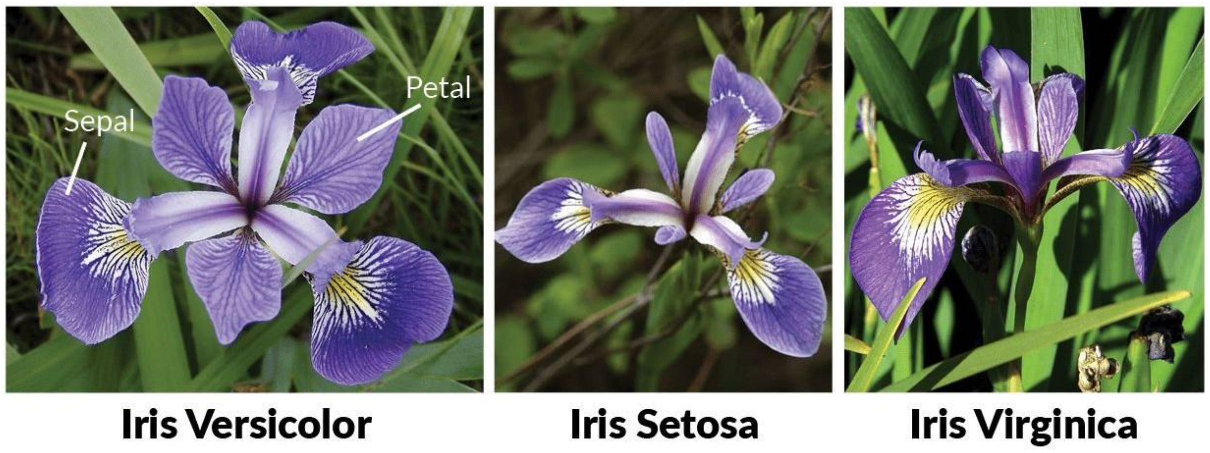
I would like to express my sincere gratitude to my supervisor Mr. Sumit Kumar Shukla for providing their invaluable guidance, comments and suggestions throughout the course of the project. I would specially thank Mr. Sumit Kumar Sir for getting me the iris dataset for my exploration and successful completion of my project.

About Iris Dataset

This data sets consists of 3 different types of irises' (Setosa, Versicolor, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray

The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.

The Iris flower data set was introduced by the British statistician and biologist Ronald Fisher in his 1936 paper "The use of multiple measurements in taxonomic problems".

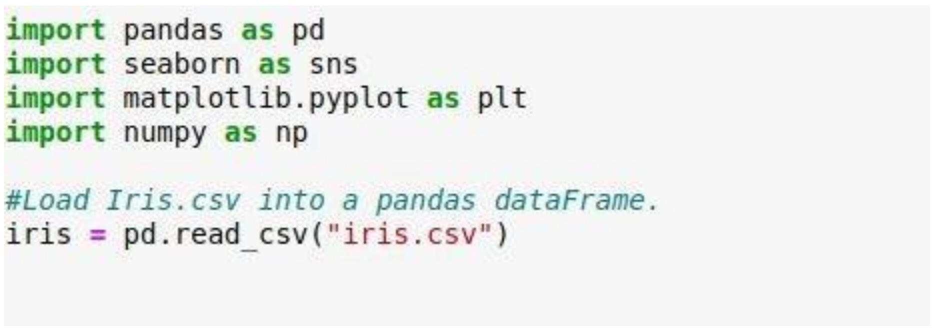


Pictures of the three flowers species FIG-1.0

Objective

Given the sepal length, sepal width, petal length and petal width, classify the Iris flower into one of the three species — Setosa, Virginica and Versicolor.

Importing libraries and loading the file.



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

#Load Iris.csv into a pandas dataFrame.
iris = pd.read_csv("iris.csv")
```

Understanding Data

Number of Datapoints and Features:

->print (iris.shape)
(150, 5) / This is also the number of rows and column .

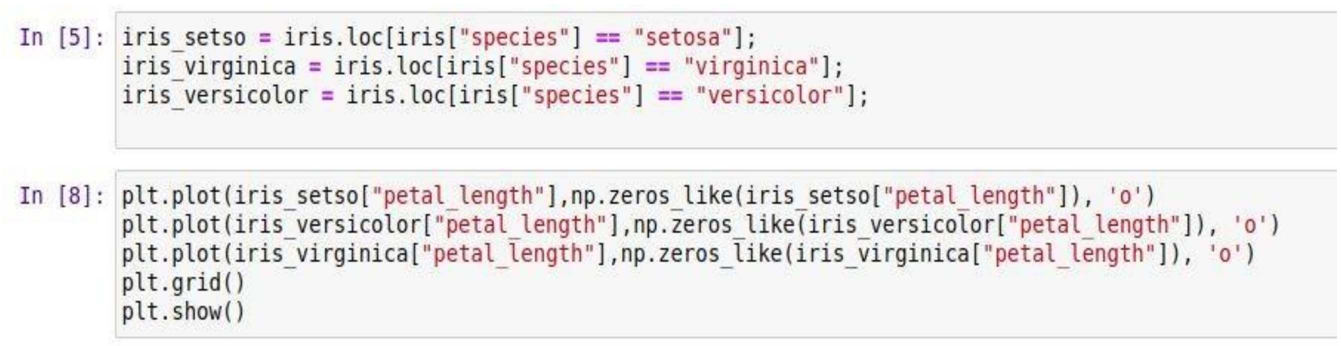
Names of columns in our dataset.
->print (iris.columns)

```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
       'species'],
      dtype='object')
```

```
->iris["species"].value_counts()
versicolor    50
virginica     50
setosa        50
Name: species, dtype: int64
```

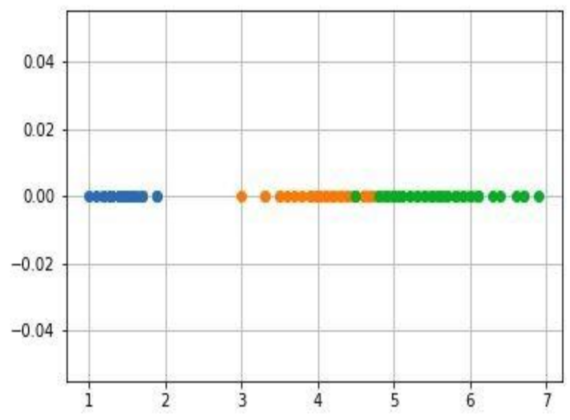
As you can see after execution of this “iris[“species”].value_counts()” ,the data distribution among setosa, virginica, versicolor are equal so iris dataset is a **Balanced dataset** (as the number of data points for every class is 50).

1-D Scatter plot



```
In [5]: iris_setso = iris.loc[iris["species"] == "setosa"];
        iris_virginica = iris.loc[iris["species"] == "virginica"];
        iris_versicolor = iris.loc[iris["species"] == "versicolor"];
```

```
In [8]: plt.plot(iris_setso["petal_length"],np.zeros_like(iris_setso["petal_length"]), 'o')
        plt.plot(iris_versicolor["petal_length"],np.zeros_like(iris_versicolor["petal_length"]), 'o')
        plt.plot(iris_virginica["petal_length"],np.zeros_like(iris_virginica["petal_length"]), 'o')
        plt.grid()
        plt.show()
```



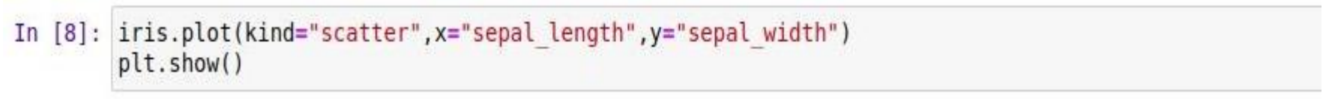
1-D scatter plot

Observation

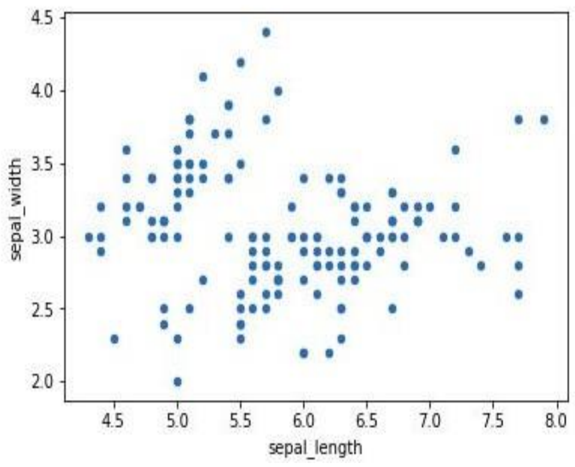
- Green points are Virginica, orange points are Versicolor and blue points are Setosa
- Virginica and Versicolor are overlapping
- 1-D Scatter are very hard to read and understand
    -

2-D scatter plot

Always understand the axis: labels and scale.
2-D Scatter plot without color-coding for each flower type/class.



```
In [8]: iris.plot(kind="scatter",x="sepal_length",y="sepal_width")
        plt.show()
```

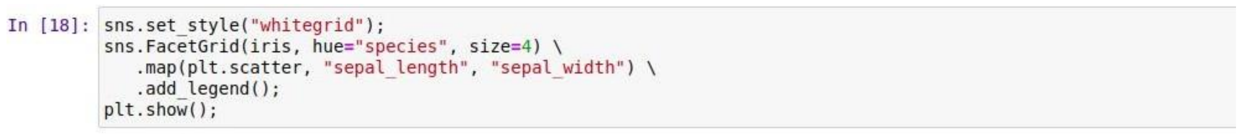


2D Scatter plot without colour

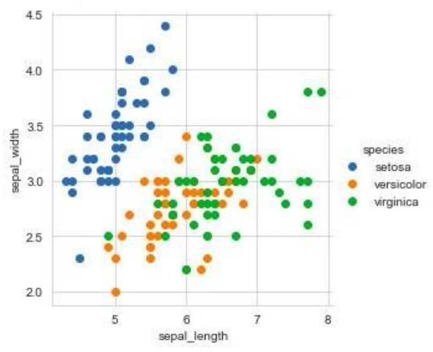
In the above figure, we are plotting sepal length on x-axis and sepal width on y-axis.we are scattering all the points that we have and putting it on the plot.. and it is called a 2D plot because we are using 2 features i.e on x-axis and y-axis.

In the above figure, we are't able to understand which is setosa or versicolor or virginica flower because all points are in same colour. It cannot make much sense out it.

2-D Scatter plot with color-coding for each flower type/class.



```
In [18]: sns.set_style("whitegrid");
         sns.FacetGrid(iris, hue="species", size=4) \
            .map(plt.scatter, "sepal_length", "sepal_width") \
            .add_legend();
         plt.show();
```



2D Scatter plot using colour-code

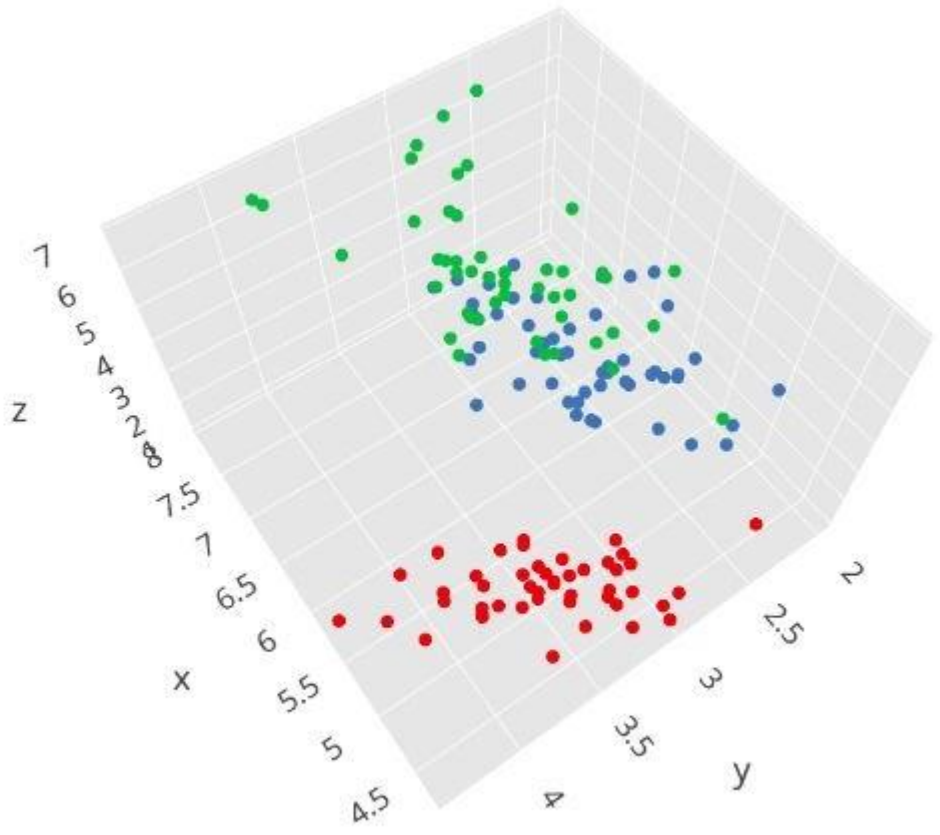
- Here 'sns' corresponds to seaborn.
- Notice that the blue points can be easily separated
- From red and green by drawing a line.
- But red and green data points cannot be easily separated.

Observation(s):

- Using sepal_length and sepal_width features, we can distinguish Setosa flowers from others.
- Separating Versicolor from Virginica is much harder as they have considerable overlap.

3-D Scatter Plot

```
import plotly.express as px
df = px.data.iris()
fig = px.scatter_3d(df, x='sepal_length', y='sepal_width',
z='petal_width',
              color='species')

fig.show()
```



3d Scatter Plot

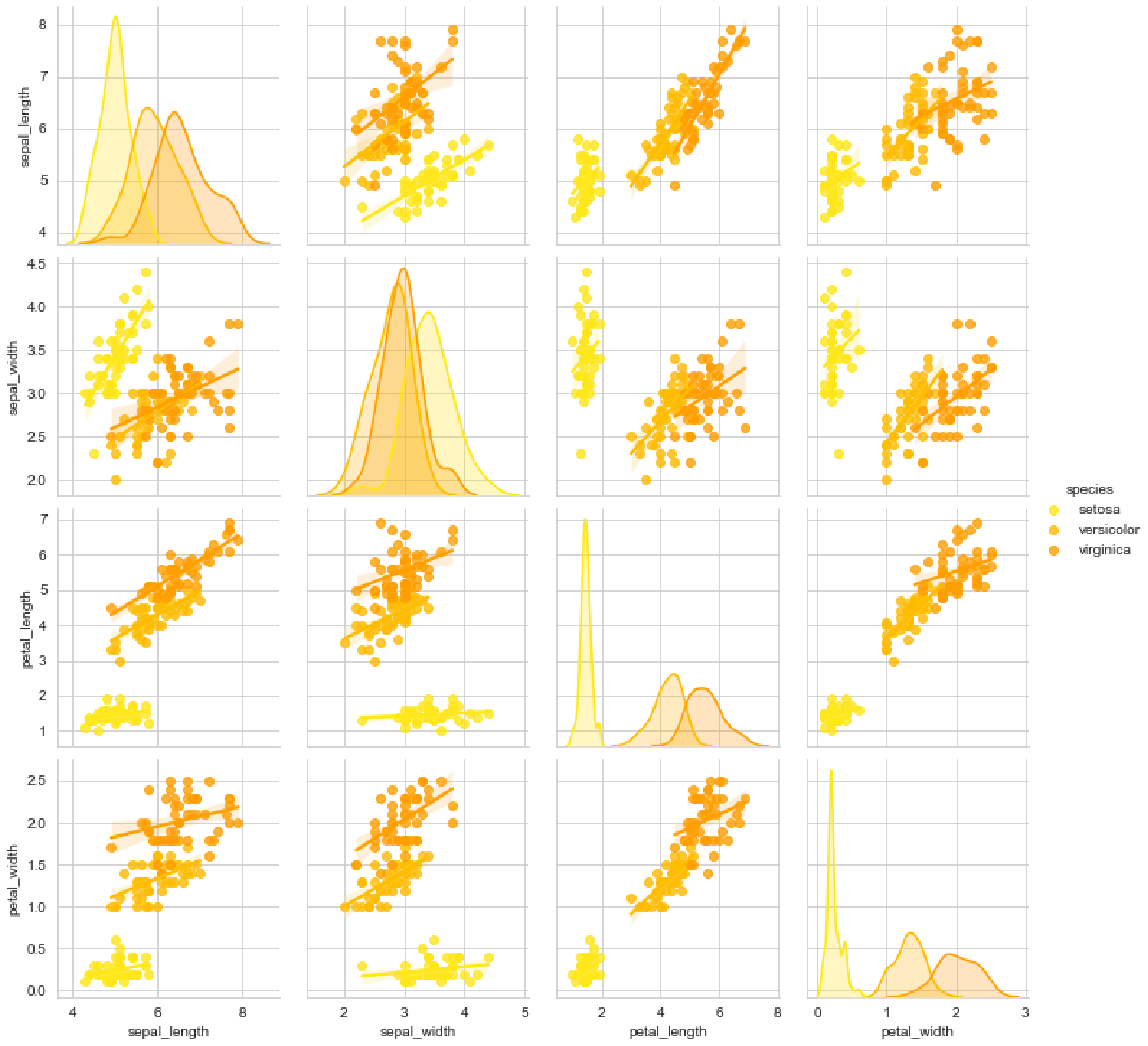
Here we are using **plotly** library for plotting as you can see we have used sepal length on the x-axis, sepal width on the y-axis and petal length on the z-axis.A 3D plot will be used for three variables or dimensions. However, what would do if we have more than 3 dimensions or features in our dataset as we humans do have the capability to visualize more than 3 dimensions?

One solution to this problem is **pair plots**.

Pair plots

Pairwise Scatter plot: Pair plot
Disadvantages:-Cannot visualize higher dimensional patterns in 3-D and 4-D, Only possible to visualize 2-D patterns

```
sns.pairplot(df_iris, hue = 'species', diag_kind = 'kde' , kind = 'scatter' , palette = 'Wistia')
plt.savefig("iris_regkde.png",transparent=True)
```



Pair- plot (changed graph)

Observations

1. petal_length and petal_width are the most useful features to identify various flower types.

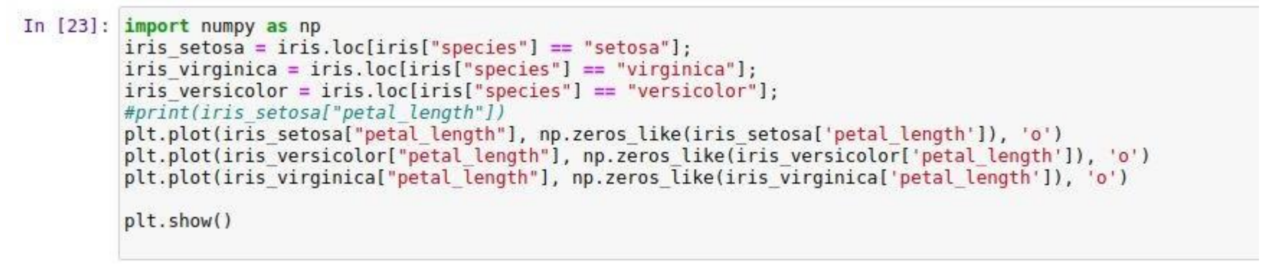
2. While Setosa can be easily identified (linearly separable), Virnica and Versicolor have some overlap (almost linearly separable).
3. We can find "lines" and "if-else" conditions to build a simple model to classify the flower types.

Histogram and Introduction of PDF

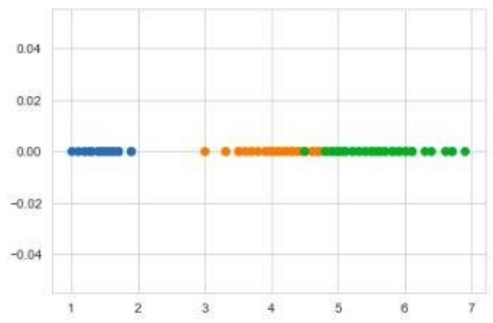
A histogram is an accurate graphical representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable (quantitative variable).To construct a histogram, the first step is to “bin” the range of values — that is, divide the entire range of values into a series of intervals — and then count how many values fall into each interval. The bins are usually specified as consecutive, non-overlapping intervals of a variable.

(source-wikipediA)

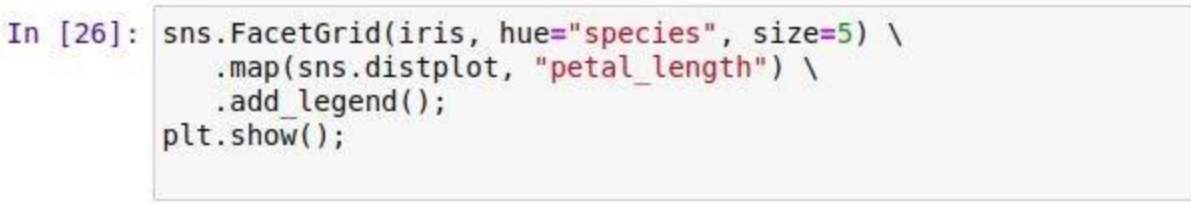
1-D scatter plot of petal-length



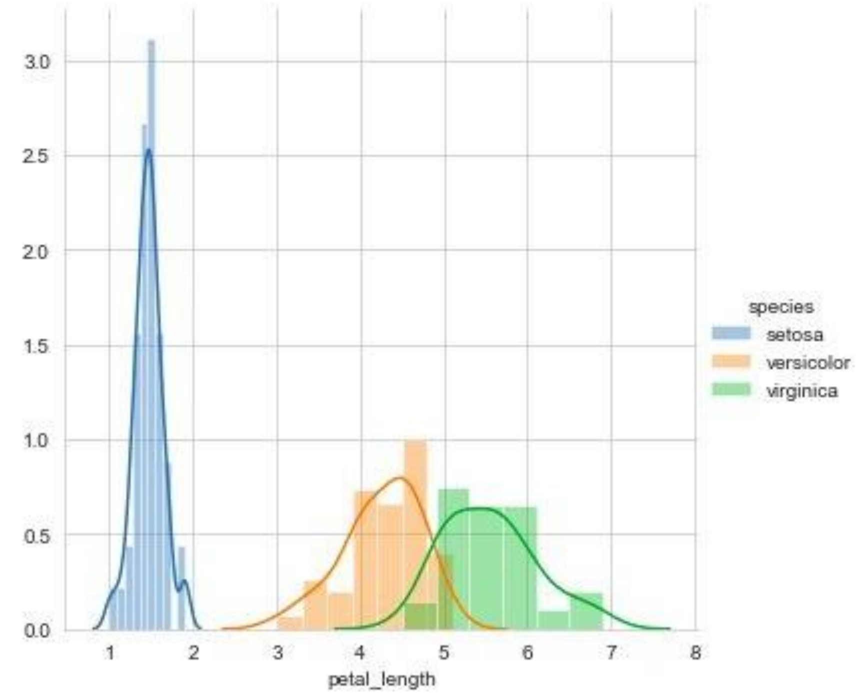
```
In [23]: import numpy as np
         iris_setosa = iris.loc[iris["species"] == "setosa"];
         iris_virginica = iris.loc[iris["species"] == "virginica"];
         iris_versicolor = iris.loc[iris["species"] == "versicolor"];
         #print(iris_setosa["petal_length"])
         plt.plot(iris_setosa["petal_length"], np.zeros_like(iris_setosa['petal_length']), 'o')
         plt.plot(iris_versicolor["petal_length"], np.zeros_like(iris_versicolor['petal_length']), 'o')
         plt.plot(iris_virginica["petal_length"], np.zeros_like(iris_virginica['petal_length']), 'o')

         plt.show()
```



Disadvantages of 1-D scatter plot: Very hard to make sense as points are overlapping.



```
In [26]: sns.FacetGrid(iris, hue="species", size=5) \
            .map(sns.distplot, "petal_length") \
            .add_legend();
         plt.show();
```

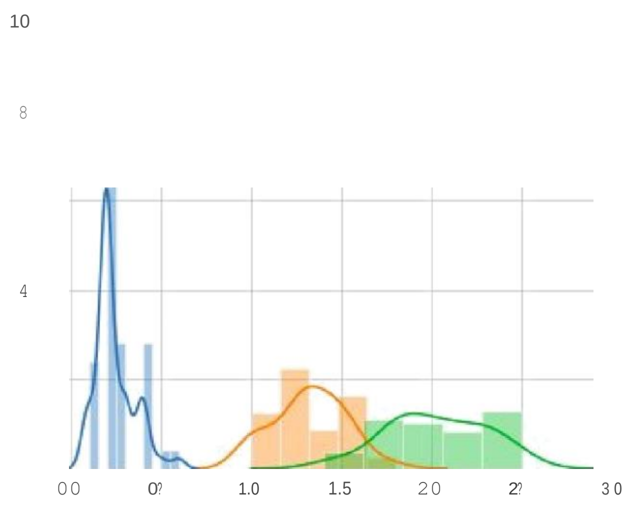


Histogram and PDF

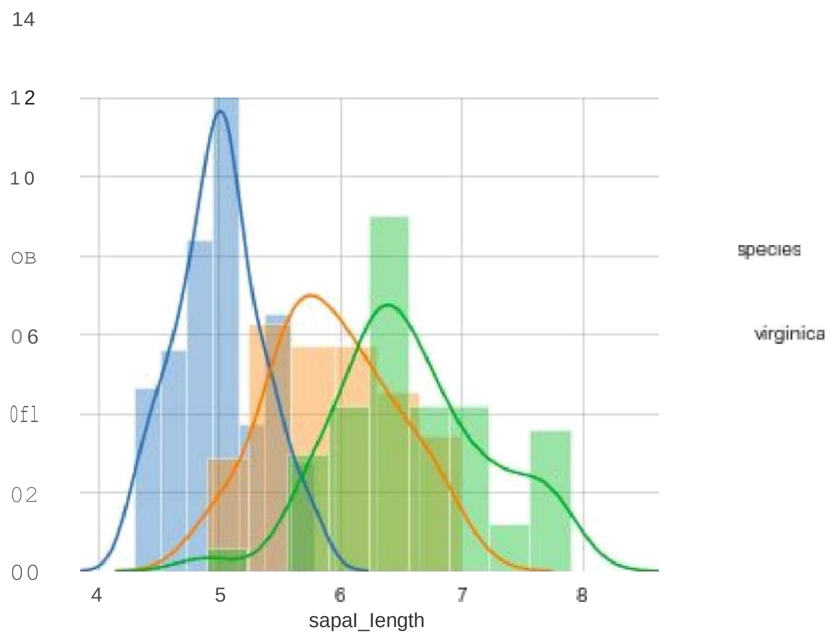
Here in the figure, x-axis is the petal length and the y axis is a count of no of points that exist in the given range. And using this plot we are able to observe how many points are there in particular regions.Histogram basically represents how many points exist for each value on the x-axis.

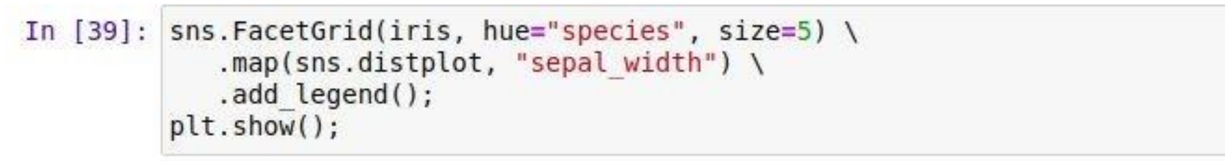
PDF (Probability Density Functions)is smoothness of histogram.

```
In [83}: sns.Facet6rid(iris, hue=”speries”, size=S) \
            .map(sns.distplot, ”petal_width“) \
            . add legend l ) ;
         ptt . s how ( ) ;
```

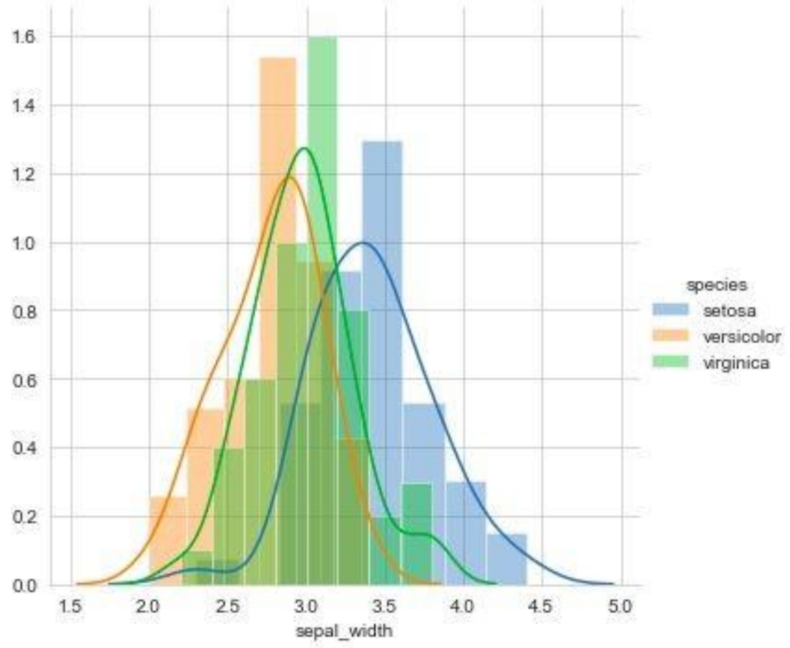


```
In  [36]     sns . FacetGr1d ( 1r1s , hue=" spec1es " , szze=5 ) \
                 . map(sns. dzstpTot , " sepat length " l \
                 . add 1eqend ( ) ;
             ptt . show ( ) ;
```





```
In [39]: sns.FacetGrid(iris, hue="species", size=5) \
            .map(sns.distplot, "sepal_width") \
            .add_legend();
         plt.show();
```



CDF(Cumulative distribution function)

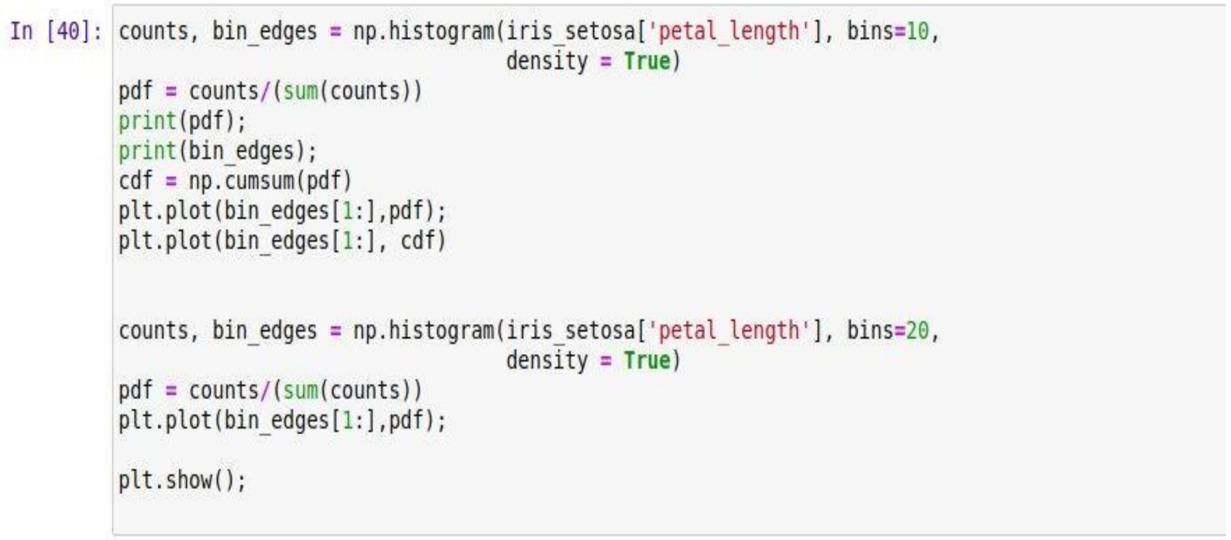
Cumulative Distribution Function

The *cumulative distribution function* (CDF) *F*X(*x*) describes the probability that a random variable *X* with a given probability distribution will be found at a value less than or equal to *x*. This function is given as:

$$F_X(x) = P\left[X \le x\right] = \int_{-\infty}^{x} f_X(u)du$$

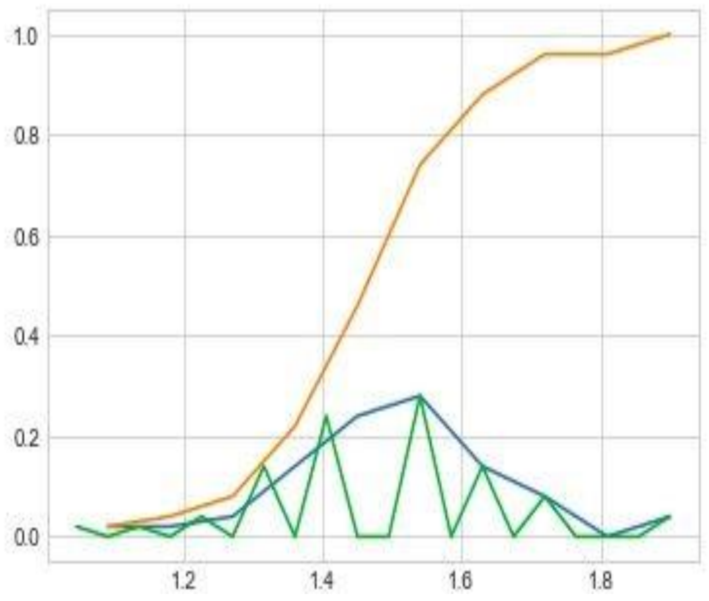
We can visually use cdf that what percentage of versicolor flowers have a petal_length of less than 5?

Plot CDF of petal_length

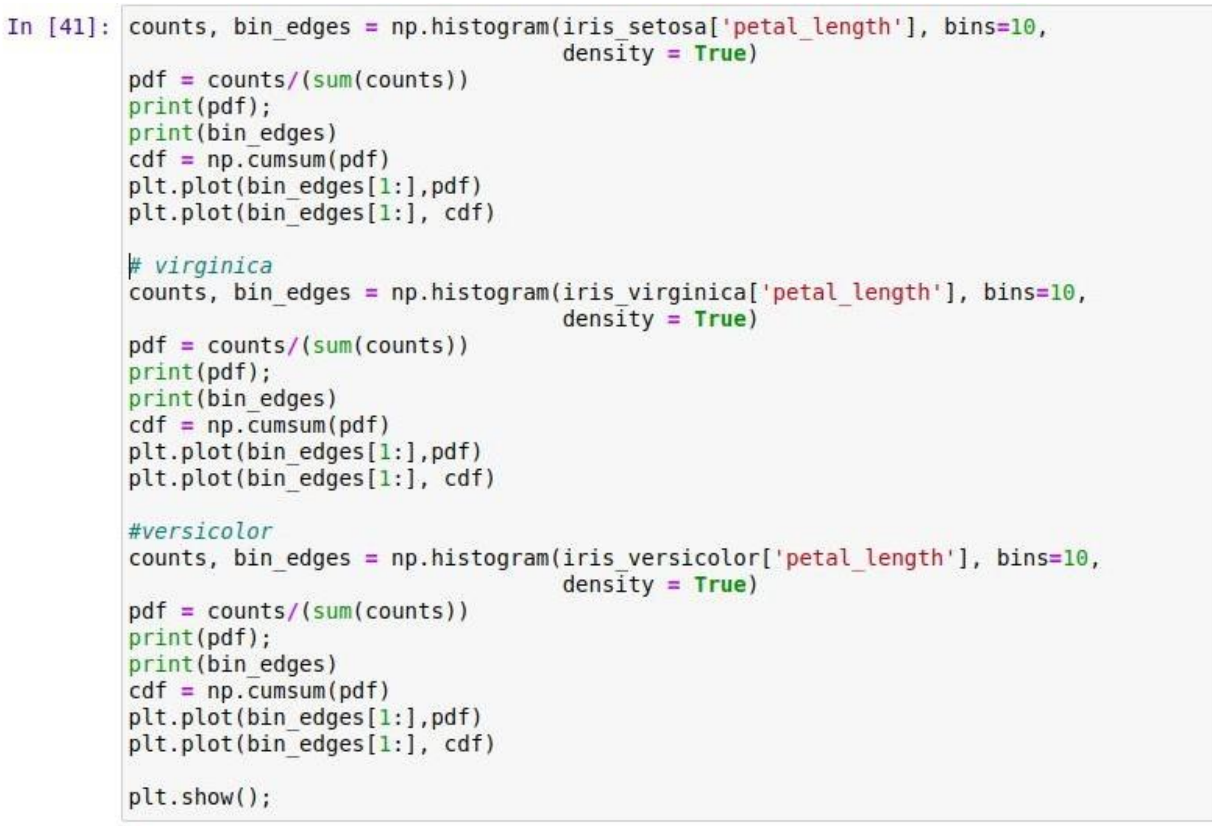


```
In [40]: counts, bin_edges = np.histogram(iris_setosa['petal_length'], bins=10,
                                          density = True)
         pdf = counts/(sum(counts))
         print(pdf);
         print(bin_edges);
         cdf = np.cumsum(pdf)
         plt.plot(bin_edges[1:],pdf);
         plt.plot(bin_edges[1:], cdf)


         counts, bin_edges = np.histogram(iris_setosa['petal_length'], bins=20,
                                          density = True)
         pdf = counts/(sum(counts))
         plt.plot(bin_edges[1:],pdf);

         plt.show();
```

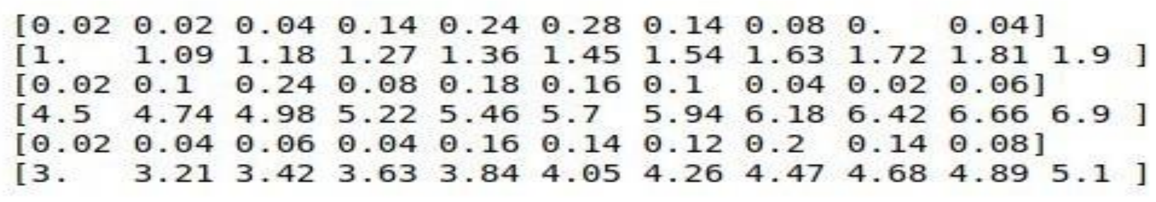
```
[ 0.02  0.02  0.04  0.14  0.24  0.28  0.14  0.08  0.    0.04]
[ 1.    1.09  1.18  1.27  1.36  1.45  1.54  1.63  1.72  1.81  1.9 ]
```



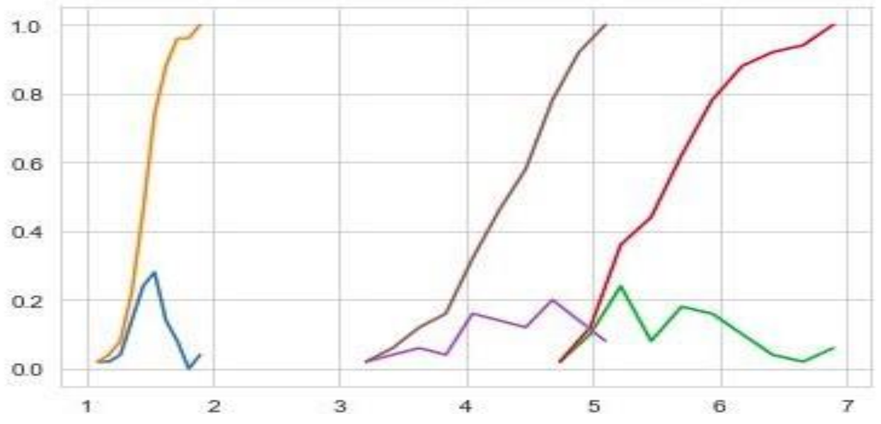
Plots of CDF of petal_length for various types of flowers.



```
In [41]: counts, bin_edges = np.histogram(iris_setosa['petal_length'], bins=10,
                                          density = True)
         pdf = counts/(sum(counts))
         print(pdf);
         print(bin_edges)
         cdf = np.cumsum(pdf)
         plt.plot(bin_edges[1:],pdf)
         plt.plot(bin_edges[1:], cdf)

         # virginica
         counts, bin_edges = np.histogram(iris_virginica['petal_length'], bins=10,
                                          density = True)
         pdf = counts/(sum(counts))
         print(pdf);
         print(bin_edges)
         cdf = np.cumsum(pdf)
         plt.plot(bin_edges[1:],pdf)
         plt.plot(bin_edges[1:], cdf)

         #versicolor
         counts, bin_edges = np.histogram(iris_versicolor['petal_length'], bins=10,
                                          density = True)
         pdf = counts/(sum(counts))
         print(pdf);
         print(bin_edges)
         cdf = np.cumsum(pdf)
         plt.plot(bin_edges[1:],pdf)
         plt.plot(bin_edges[1:], cdf)

         plt.show();
```



```
[0.02 0.02 0.04 0.14 0.24 0.28 0.14 0.08 0.   0.04]
[1.   1.09 1.18 1.27 1.36 1.45 1.54 1.63 1.72 1.81 1.9 ]
[0.02 0.1  0.24 0.08 0.18 0.16 0.1  0.04 0.02 0.06]
[4.5  4.74 4.98 5.22 5.46 5.7  5.94 6.18 6.42 6.66 6.9 ]
[0.02 0.04 0.06 0.04 0.16 0.14 0.12 0.2  0.14 0.08]
[3.   3.21 3.42 3.63 3.84 4.05 4.26 4.47 4.68 4.89 5.1 ]
```



Differentiation of CDF = PDF

Integration of PDF = CDF

Mean, Variance and Standard Deviation

Mean is average of a given set of data

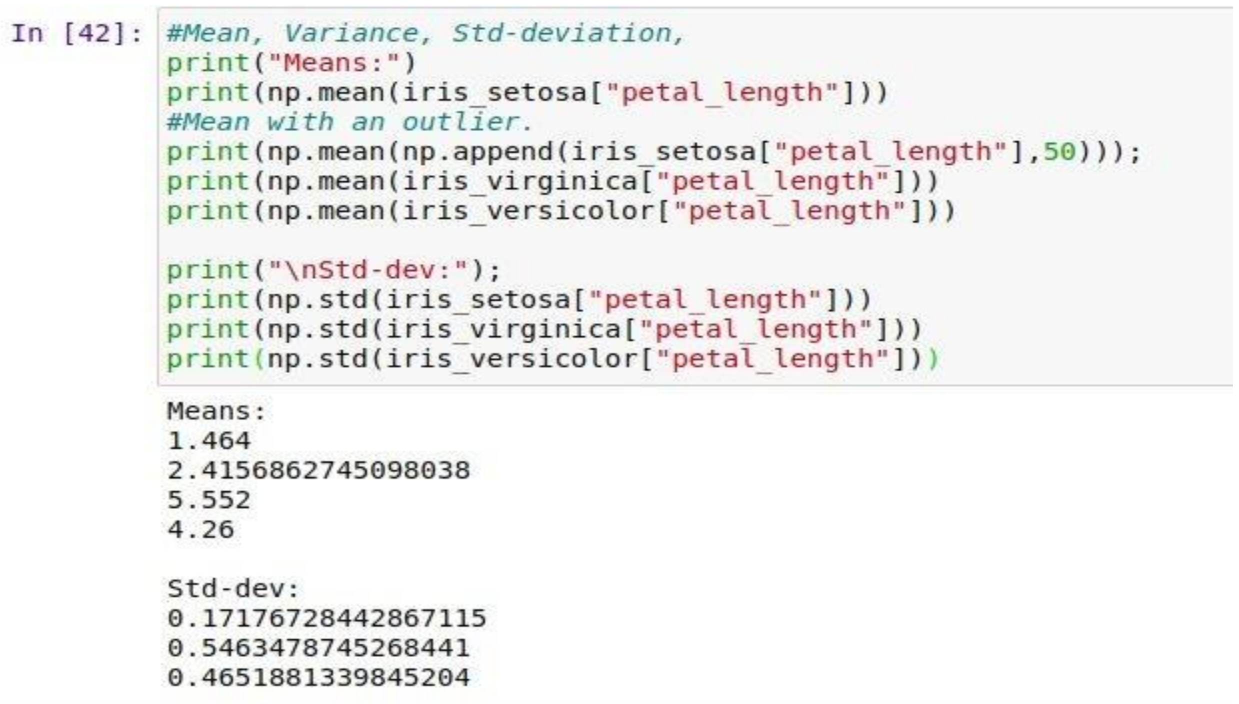
```
Ex - 1,2,3,4,5
These five data points have the mean (average) of 3:
(1+2+3+4+5)/ 5 =3
```

Variance is the sum of squares of differences between all numbers and means.

```
(1-3)^2 = 4                  (4-3)^2 = 1
(2-3)^2 = 1                  (5-3)^2 = 4
(3-3)^2 = 0
Variance =(4+1+0+1+4)/5 = 2
```

Standard Deviation is square root of variance. It is a measure of the extent to which data varies from the mean.
Standard Deviation = Square root of 2 is $\sqrt{2}$

Means and Std-dev



```
In [42]: #Mean, Variance, Std-deviation,
         print("Means:")
         print(np.mean(iris_setosa["petal_length"]))
         #Mean with an outlier.
         print(np.mean(np.append(iris_setosa["petal_length"],50)));
         print(np.mean(iris_virginica["petal_length"]))
         print(np.mean(iris_versicolor["petal_length"]))

         print("\nStd-dev:");
         print(np.std(iris_setosa["petal_length"]))
         print(np.std(iris_virginica["petal_length"]))
         print(np.std(iris_versicolor["petal_length"]))

         Means:
         1.464
         2.4156862745098038
         5.552
         4.26

         Std-dev:
         0.17176728442867115
         0.5463478745268441
         0.4651881339845204
```

Observation(s) | Conclusion

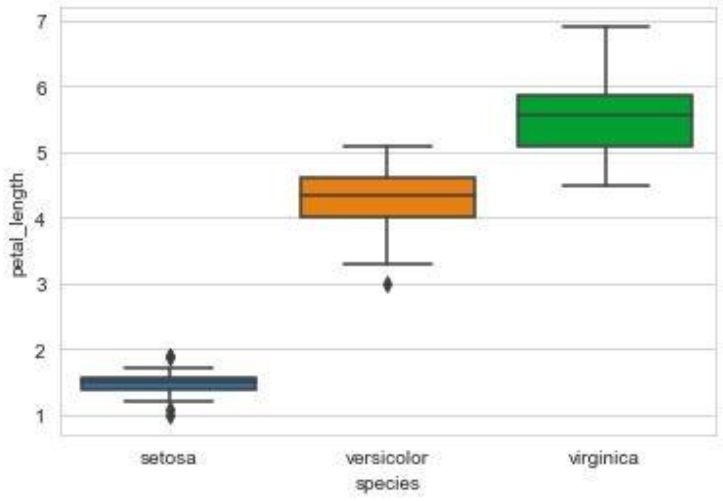
1. Now we can say that Satosa has less petal length
2. Virginica and Versicolor both have slightly closer patel length.

Box plot and Whiskers

A box and whisker plot (sometimes called a boxplot) is a graph that presents information from a five-number summary.
It does not show a distribution in as much detail as a stem and leaf plot or histogram does, but is especially useful for indicating whether a distribution is skewed and whether there are potential unusual observations (outliers) in the data set.
Box-plot with whiskers: another method of visualising the 1-D scatter plot more intuitive

Box-plot can be visualized as a PDF on the side-ways.

```
In [43]: sns.boxplot(x='species',y='petal_length', data=iris)
         plt.show()
```

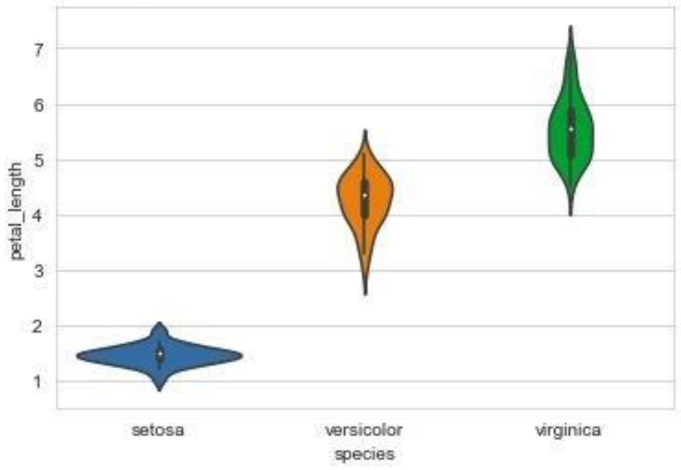


Box-plot

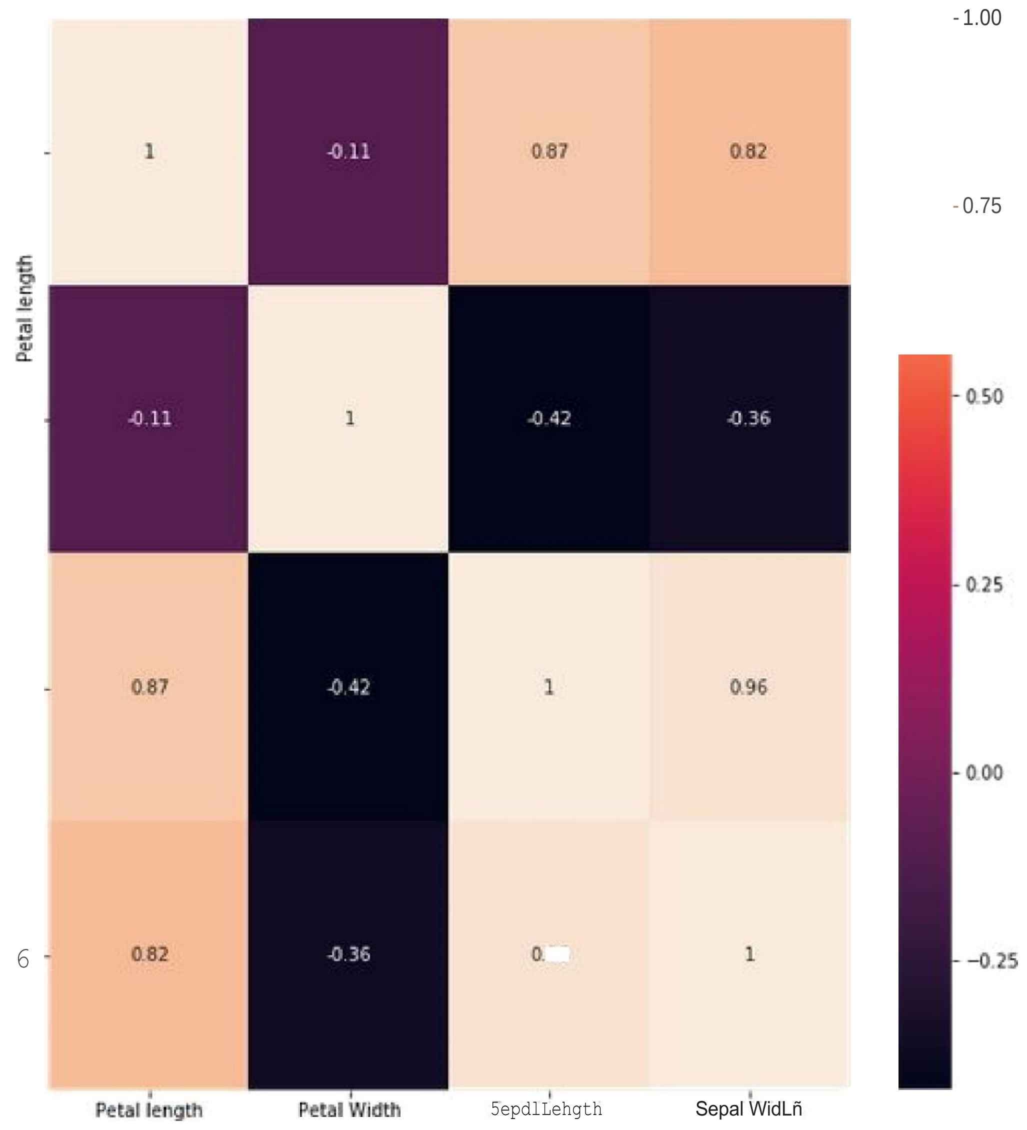
Violin plots

A violin plot combines the benefits of the previous two plots and simplifies them
Denser regions of the data are fatter, and sparser ones thinner in a violin plot

```
In [44]: sns.violinplot(x="species", y="petal_length", data=iris, size=8)
         plt.show()
```



Now we will see how these features are correlated to each other using a heatmap in seaborn library. We can see that Sepal Length and Sepal Width features are slightly correlated with each other.

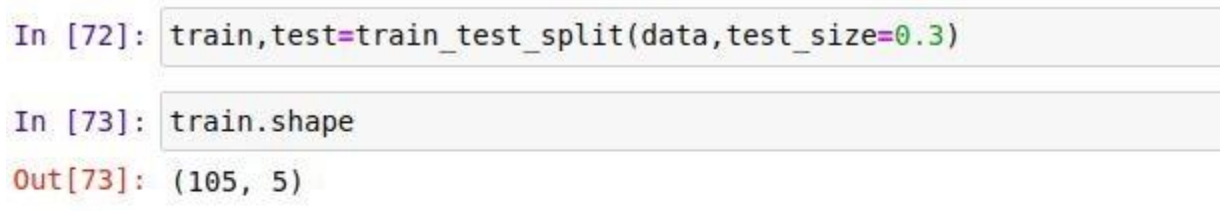


Petal length
6
1
-0.11
0.87
0.82
-0.11
1
-0.42
-0.36
0.87
-0.42
1
0.96
0.82
-0.36
0.
1
Petal length
Petal Width
5epdlLehgth
Sepal WidLñ
1.00
0.75
0.50
0.25
0.00
−0.25

Building Model

Splitting Dataset:

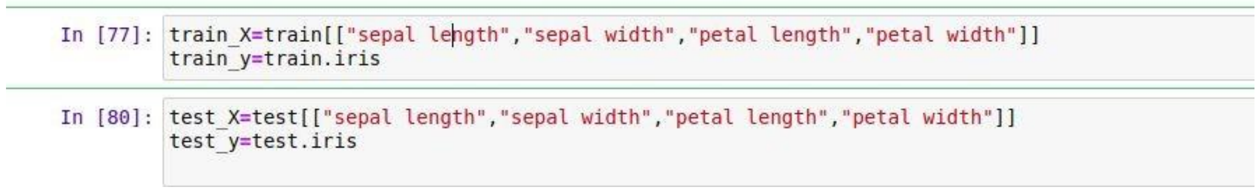
Before implementing any model we need to split the dataset to *train* and *test* sets. We use *train_test_split* class from *sklearn.model_selection* library to split our dataset.



```
In [72]: train,test=train_test_split(data,test_size=0.3)

In [73]: train.shape
Out[73]: (105, 5)
```

The above code will split the dataset to 70% as train and 30% as test datasets.

Now let’s split the train and test sets further as input and output sets.



```
In [77]: train_X=train[["sepal length","sepal width","petal length","petal width"]]
         train_y=train.iris

In [80]: test_X=test[["sepal length","sepal width","petal length","petal width"]]
         test_y=test.iris
```

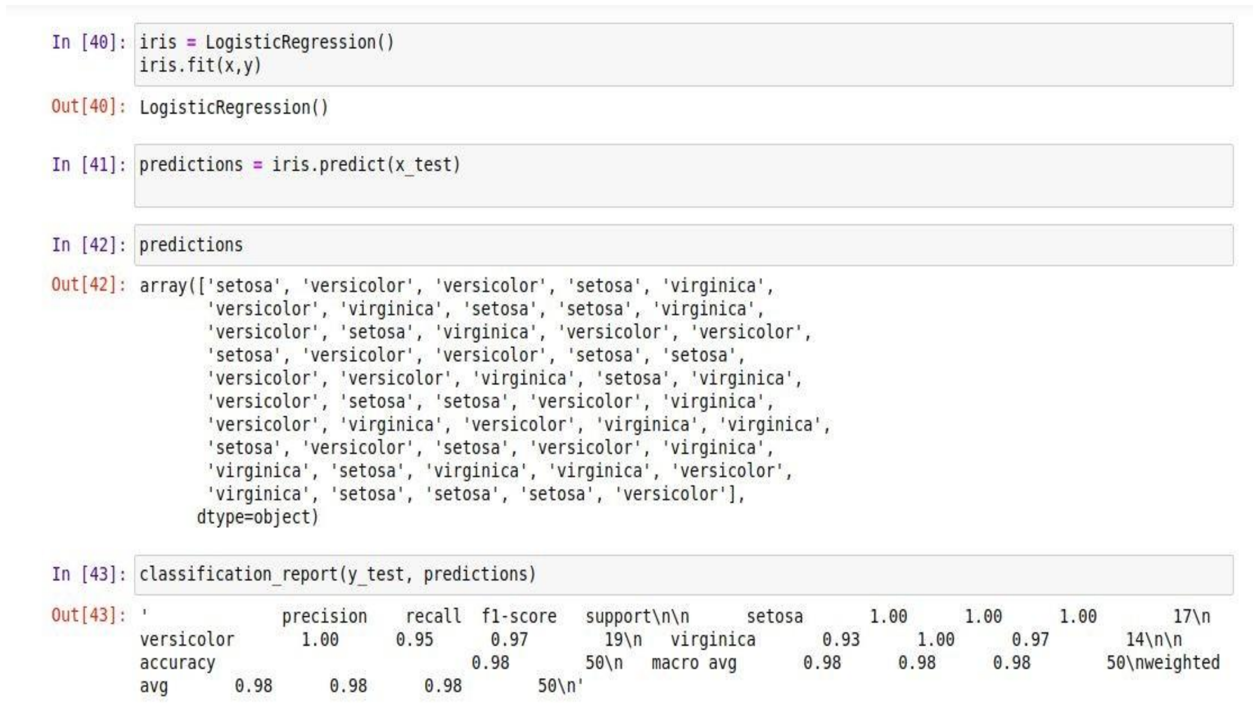
Decision Trees Model:

Decision trees build classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. It uses *Entropy* and *Information Gain* to construct a decision tree.

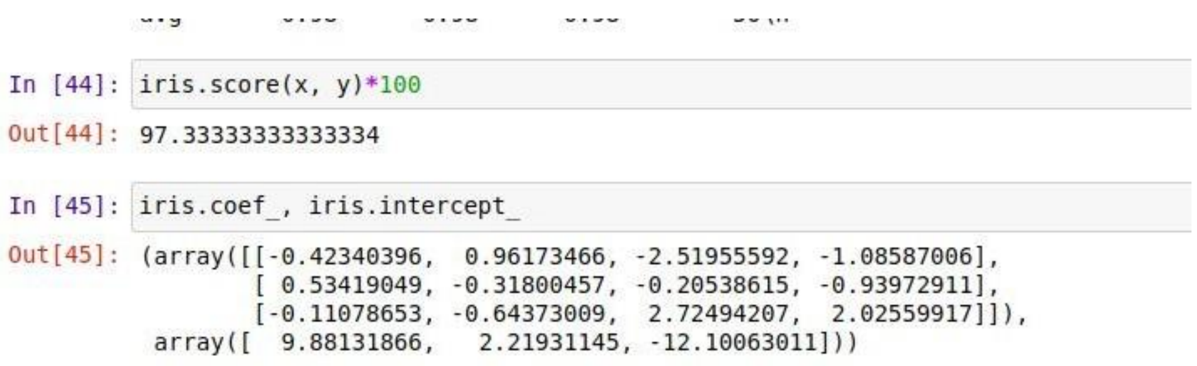
Entropy

Entropy controls how a Decision Tree decides to split the data. It actually affects how a Decision Tree draws its boundaries.

Let’s build the Decision tree model on the train set.We can predict the output for the test dataset using predict() function. Let’s do that.And calculate the accuracy score of the model.

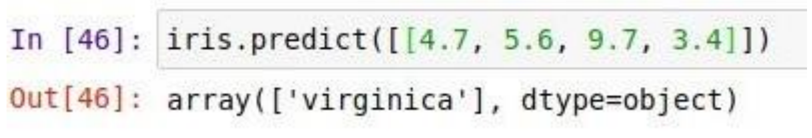


```
In [40]: iris = LogisticRegression()
         iris.fit(x,y)

Out[40]: LogisticRegression()

In [41]: predictions = iris.predict(x_test)

In [42]: predictions

Out[42]: array(['setosa', 'versicolor', 'versicolor', 'setosa', 'virginica',
                'versicolor', 'virginica', 'setosa', 'setosa', 'virginica',
                'versicolor', 'setosa', 'virginica', 'versicolor', 'versicolor',
                'setosa', 'versicolor', 'versicolor', 'setosa', 'setosa',
                'versicolor', 'versicolor', 'virginica', 'setosa', 'virginica',
                'versicolor', 'setosa', 'setosa', 'versicolor', 'virginica',
                'versicolor', 'virginica', 'versicolor', 'virginica', 'virginica',
                'setosa', 'versicolor', 'setosa', 'versicolor', 'virginica',
                'virginica', 'setosa', 'virginica', 'virginica', 'versicolor',
                'virginica', 'setosa', 'setosa', 'setosa', 'versicolor'],
               dtype=object)

In [43]: classification_report(y_test, predictions)

Out[43]: '              precision    recall  f1-score   support\n\n      setosa       1.00      1.00      1.00        17\n
         versicolor       1.00      0.95      0.97        19\n   virginica       0.93      1.00      0.97        14\n\n
         accuracy                           0.98        50\n   macro avg       0.98      0.98      0.98        50\nweighted
         avg       0.98      0.98      0.98        50\n'
```



```
In [44]: iris.score(x, y)*100

Out[44]: 97.33333333333334

In [45]: iris.coef_, iris.intercept_

Out[45]: (array([[-0.42340396,  0.96173466, -2.51955592, -1.08587006],
                 [ 0.53419049, -0.31800457, -0.20538615, -0.93972911],
                 [-0.11078653, -0.64373009,  2.72494207,  2.02559917]]),
          array([  9.88131866,   2.21931145, -12.10063011]))
```

Now we can predict the species using the sepal length, sepal width, petal length and petal width,

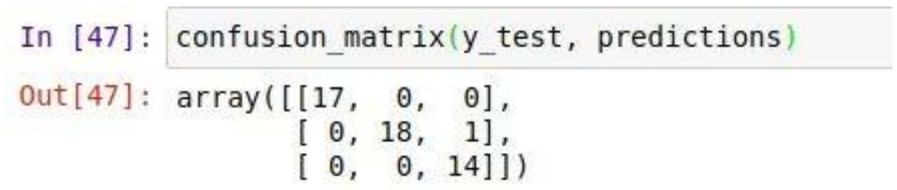


```
In [46]: iris.predict([[4.7, 5.6, 9.7, 3.4]])

Out[46]: array(['virginica'], dtype=object)
```

Here, we have values like sepal_length=4.7, sepal_width=5.6, petal_length=9.7, petal_width=3.4 and using these values we have predicted that the species is ‘Verginica’.

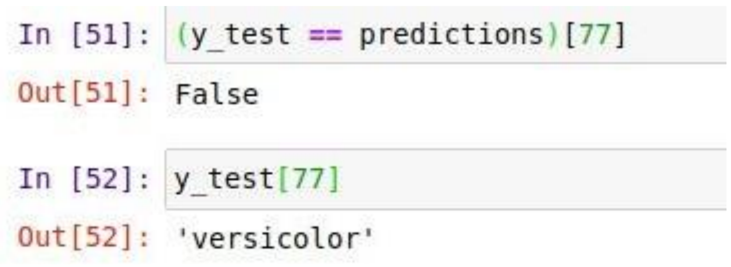
Evaluating the model:

Confusion matrix :-A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.



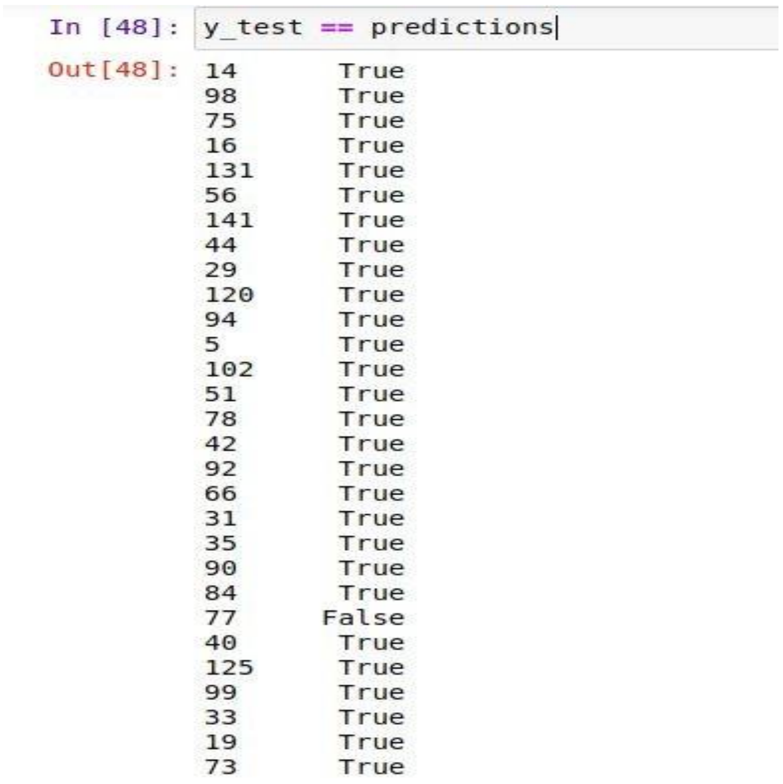
```
In [47]: confusion_matrix(y_test, predictions)
Out[47]: array([[17,  0,  0],
                [ 0, 18,  1],
                [ 0,  0, 14]])
```

In this confusion matrix we have diagonal 17+18+14= 49 which is the true value and the rest one is a wrong prediction.



```
In [51]: (y_test == predictions)[77]
Out[51]: False

In [52]: y_test[77]
Out[52]: 'versicolor'
```

We can find the 1 wrong prediction using y_test prediction.



```
In [48]: y_test == predictions

Out[48]: 14       True
         98       True
         75       True
         16       True
         131      True
         56       True
         141      True
         44       True
         29       True
         120      True
         94       True
         5        True
         102      True
         51       True
         78       True
         42       True
         92       True
         66       True
         31       True
         35       True
         90       True
         84       True
         77      False
         40       True
         125      True
         99       True
         33       True
         19       True
         73       True
```

Conclusion:- In this project we formulated the task of predicting which species of iris a particular flower belongs to by using physical measurements of the flower. We used a dataset of measurements that was annotated by an expert with the correct species to build our model, making this a supervised learning task. There were three possible species, setosa, versicolor, or virginica, which made the task a three-class classification problem. The possible species are called classes in the classification problem, and the the species of a single iris is called its label.The Iris dataset consists of two NumPy arrays: one containing the data, which is referred to as X in scikit-learn , and one containing the correct or desired outputs which is called y . The array X is a two-dimensional array of features, with one row perdata point and one column per feature. The array y is a one-dimensional array, which here contains one class label, an integer ranging from 0 to 2, for each of the samples.We split our dataset into a training set, to build our model, and a test set, to evaluate how well our model will generalize to new, previously unseen data We chose the LinerRegression algorithm This is implemented in the LinerRegression, which contains the algorithm that builds the model as well as the algorithm that makes a prediction using the model.We instantiate the class, setting parameters. Then we built the model by calling the fit method, passing the training data (X_train) and training outputs (y_train) parameters. We evaluated the model using the score method, which computes the accuracy of the model. We applied the score method to the test set data and the testset labels and found that our model is about 97.333% accurate, meaning it is correct 97%of the time on the test set.

This gave us the confidence to apply the model to new data (in our example, newflower measurements) and trust that the model will be correct about 97% of the time.