## 20.1 Introduction to IRIS dataset and 2D scatter plot

### Exploratory Data Analysis - Definition

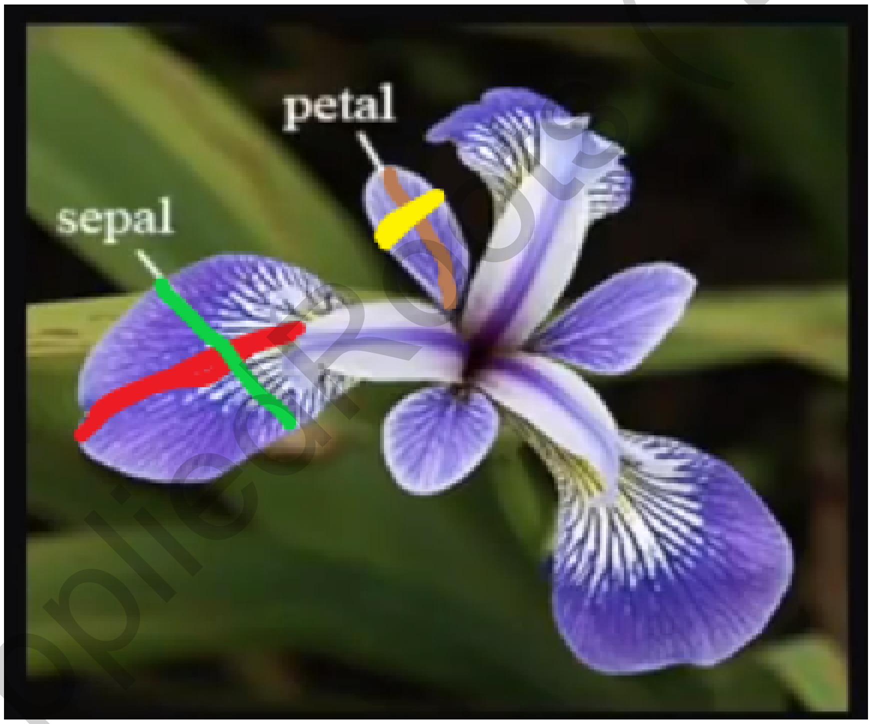
- ➢ Exploratory Data Analysis is an approach of analyzing the datasets to summarize the main characteristics, often with the visual methods.
- ➢ Whether a statistical model is used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modelling or hypothesis testing task.

### IRIS Dataset Description

The IRIS dataset consists of 5 columns. They are

1) Sepal Length
2) Sepal Width
3) Petal Length
4) Petal Width
5) Species

Below is the image of the IRIS flower that was shown at the timestamp 4:57 in the video.



| | | |
|---|---|---|
| Red Line | → | Indicates the Sepal Length |
| Green Line | → | Indicates the Sepal Width |
| Brown Line | → | Indicates the Petal Length |
| Yellow Line | → | Indicates the Petal Width |

The first 4 features denote different dimensions of the IRIS flower and they all accept only continuous numerical values. The ‘Species’ feature denotes the class to which the flower belongs to and it is a discrete feature. We have 3 classes of IRIS flower and they are

1) Setosa
2) Versicolor
3) Virginica

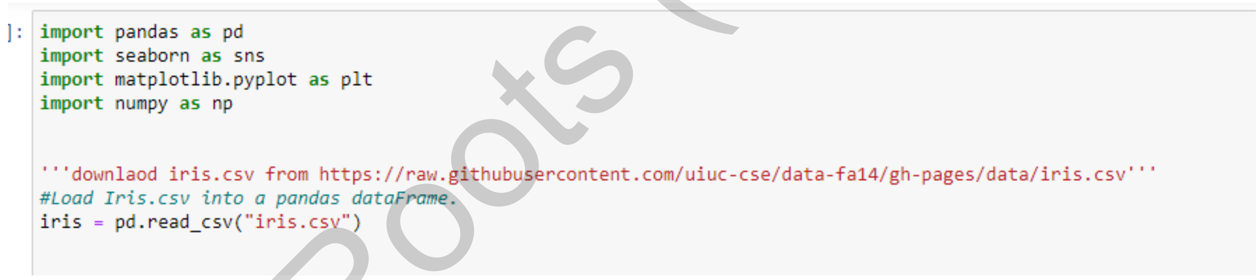
Our main objective here is to classify a given species into any one of these 3 categories.

## Reading a dataset into a Pandas Dataframe

### read_csv()

➢ The function read_csv() is used to read the dataset present in a CSV file into a pandas dataframe.
➢ The reason for using pandas dataframes is because data importing and data manipulations are easier with the functions and attributes defined in Pandas module.

Below is an example on how to read the dataset into a pandas dataframe and it has been discussed at the timestamp 12:30 in the video.

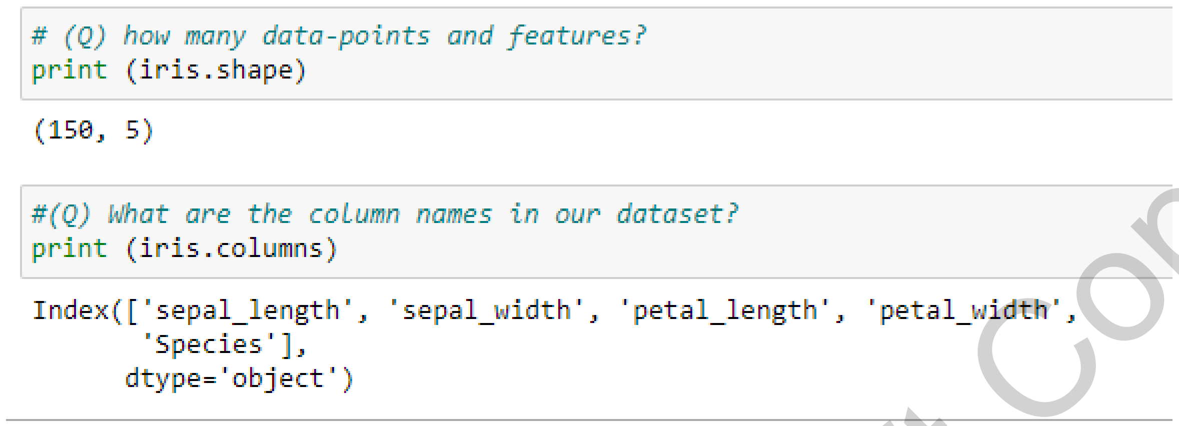


```
]: import pandas as pd
   import seaborn as sns
   import matplotlib.pyplot as plt
   import numpy as np


   '''downlaod iris.csv from https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv'''
   #Load Iris.csv into a pandas dataFrame.
   iris = pd.read_csv("iris.csv")
```

### shape and columns attributes

➢ The ‘shape’ attribute is used to return the number of rows and columns present in the dataframe. The result is returned in the form of a tuple with two values.
➢ The ‘columns’ attribute is used to return the names of all the columns of the dataframe.

Below is an example that shows us the usage of these two attributes and it was discussed starting from the timestamp 13:10.



```
# (Q) how many data-points and features?
print (iris.shape)
```

```
(150, 5)
```

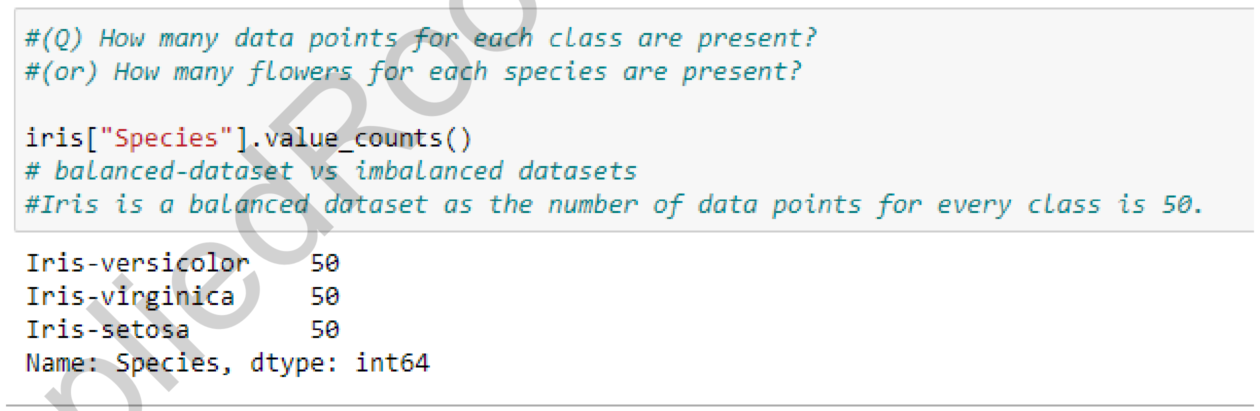
```
#(Q) What are the column names in our dataset?
print (iris.columns)
```

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

## value_counts()

- ➢ Whenever a categorical column in a pandas dataframe consists of multiple categories, then in order to get the count of rows/data points for each category, we use the **value_counts()** function in Pandas.
- ➢ It is recommended to use this function only for categorical features, but not for the continuous numerical features, as it is not a good practice.

Below is an example that illustrates the usage of **value_counts()** function and it was discussed at the timestamp 14:45.



```
#(Q) How many data points for each class are present?
#(or) How many flowers for each species are present?

iris["Species"].value_counts()
# balanced-dataset vs imbalanced datasets
#Iris is a balanced dataset as the number of data points for every class is 50.
```

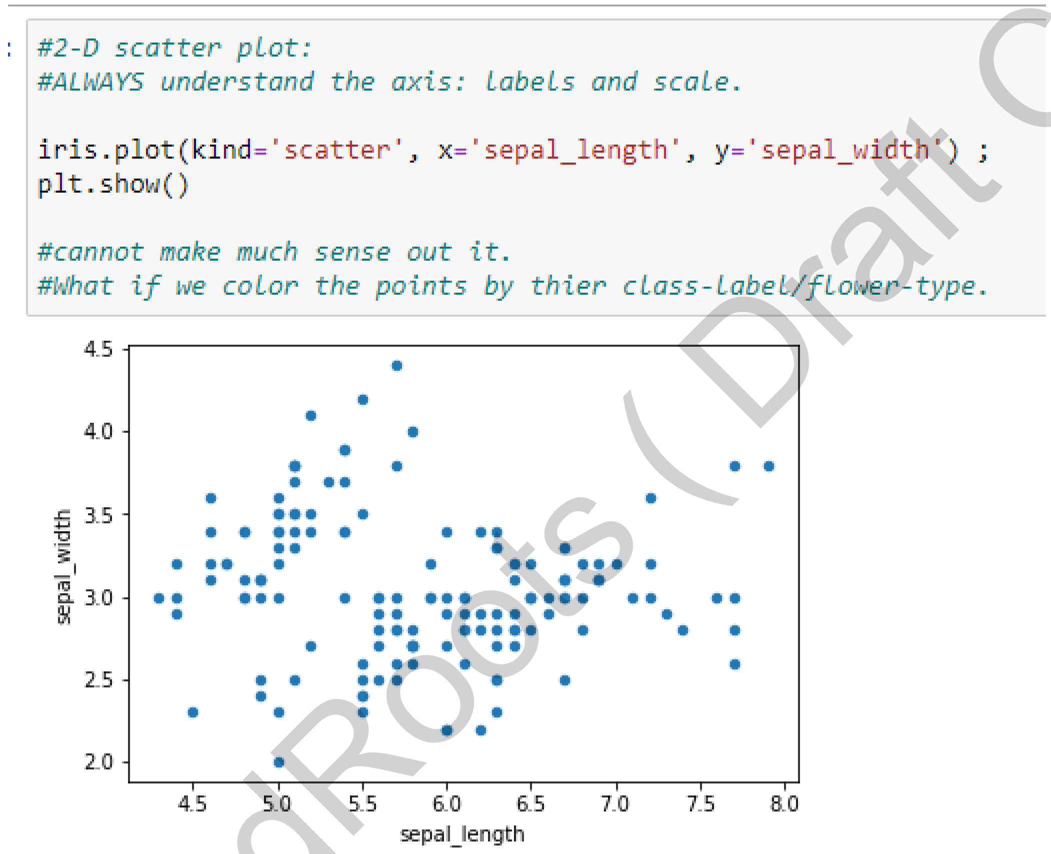
```
Iris-versicolor    50
Iris-virginica     50
Iris-setosa        50
Name: Species, dtype: int64
```

## 2-D Scatter Plot

- ➢ Scatter plots use dots to represent values for two different numerical variables. The position of each dot on the horizontal and the vertical axes indicate values for an individual data point.

- ➢ Scatter plots are used to observe the relationships between two numeric variables.
- ➢ A scatter plot can be used when one of the two variables is an independent variable and the other is a dependent variable. Or we also can use when both the variables are independent.
- ➢ The dots in a scatter plot not only report the values of individual data points, but also the patterns when the data is taken as a whole.

Below is an example that illustrates how to plot a scatter plot using plot() function in Pandas and it was discussed at the timestamp 17:45.



```
#2-D scatter plot:
#ALWAYS understand the axis: labels and scale.

iris.plot(kind='scatter', x='sepal_length', y='sepal_width') ;
plt.show()

#cannot make much sense out it.
#What if we color the points by thier class-label/flower-type.
```

In this example, we are building a scatter plot using the plot() function in Pandas. The value passed to the 'kind' parameter indicates what kind of plot we need. The value 'scatter' indicates scatter plot. The values passed to the 'x' and 'y' parameters are those column names with which we want to build the scatter plots.

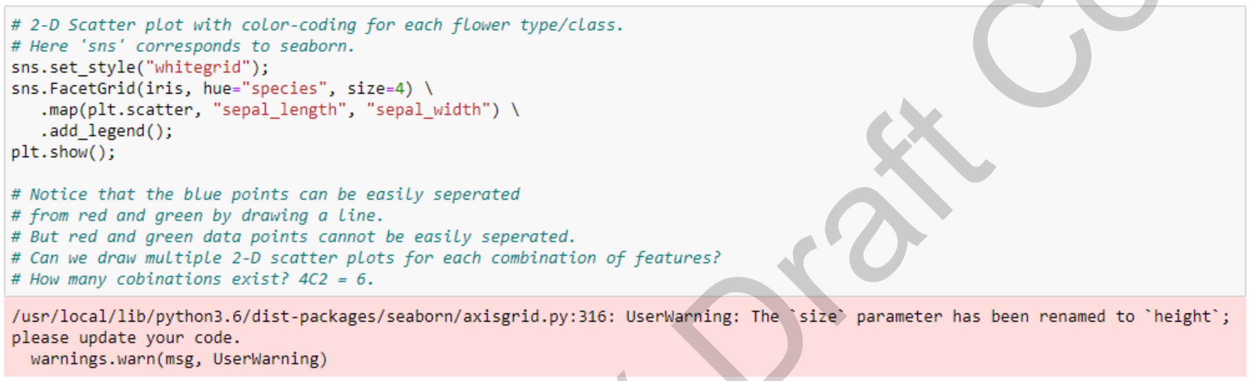
**Note**: The point of intersection of axes in coordinate geometry is the origin, but in this plot, it is not the origin. It is because in plotting, for both the features (ie., 'sepal_length' and 'sepal_width'), the maximum and minimum values are obtained and the only region in between these limits is displayed in the plot.

For every plot, we should clearly observe the scale of the axes and the point of intersection of the axes and also the legends.

You can go through the documentation of **pandas.plot()** here to explore more about this function. The statement **plot.show()** is used to display our final resultant plot and it should be the last statement among all the plotting steps.

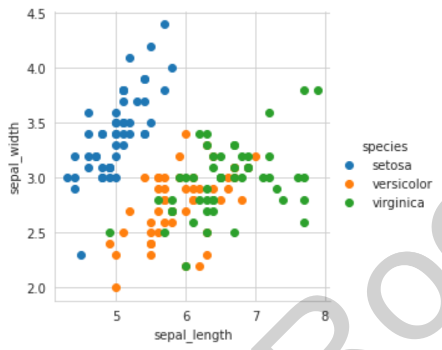
When we observe the above plot, we see all the points belonging to different classes are in the same color. So it is difficult for one to differentiate the points belonging to one class from the points belonging to other classes. So we shall apply color coding, so that points belonging to one class will have the same color and in this way, each class is assigned with a unique color.

Below is an example of such a plot which was discussed at the timestamp 20:50.



```
# 2-D Scatter plot with color-coding for each flower type/class.
# Here 'sns' corresponds to seaborn.
sns.set_style("whitegrid");
sns.FacetGrid(iris, hue="species", size=4) \
   .map(plt.scatter, "sepal_length", "sepal_width") \
   .add_legend();
plt.show();

# Notice that the blue points can be easily seperated
# from red and green by drawing a line.
# But red and green data points cannot be easily seperated.
# Can we draw multiple 2-D scatter plots for each combination of features?
# How many cobinations exist? 4C2 = 6.
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/axisgrid.py:316: UserWarning: The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



In this example, we are using the seaborn library to build the same plot, but in a much more attractive and an understandable format.

Here the first statement is **sns.set_style()** which indicates how the background of our plot should look like. As we have set the style to **'whitegrid'**, the background is white in color with a grid(ie., the black horizontal and vertical lines). We also can use values like **'darkgrid'**, **'dark'** or **'white'** as the background styles. You can refer to the documentation of **set_style()** here.

The next statement is **sns.FacetGrid()** which initializes a multi-plot grid for plotting conditional relationships. You always have to pass only a pandas dataframe as the data input and the value passed to the **'hue'** parameter indicates the column name on the basis of whose values, we have to assign the colors. Here as we are passing 'species' as the value to the 'hue' parameter, as this parameter contains 3 values (ie., 'Iris-setosa', 'Iris-versicolor' and 'Iris-virginica'), all the points belonging to the

‘Iris-setosa’ class will have one color, the points belonging to the ‘Iris-versicolor’ class will have other color and all the points belonging to the ‘Iris-virginica’ class will have another color.

The **‘size’** parameter was deprecated and is now replaced by **‘height’** and it takes only numerical values, which indicates the height of the plot. The more the value, the more large the plot would be.

Next is the **map()** function which takes a function name as the first argument and then applies that function to the following mentioned column names. Here it applied the **plt.scatter()** function in between the **‘sepal_length’** and **‘sepal_width’** columns.

The function **add_legend()** adds the labels to the plot about which color indicates which class of points.

When we look at the final plot obtained, we see the that the points belonging to ‘Iris-setosa’ class are easily separable by a line, whereas the points belong to the ‘Iris-versicolor’ and the ‘Iris-virginica’ classes are not easily separable by a line.

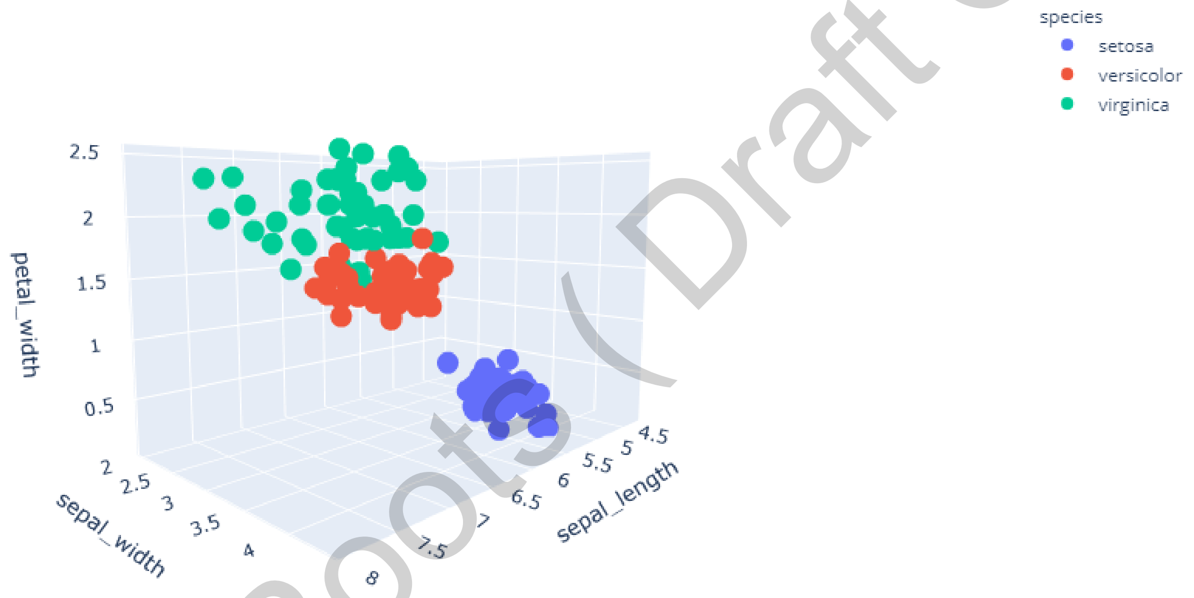
You can explore **sns.FacetGrid()** by referring to the documentation here.

## 20.2 3D Scatter Plots

A 3D scatter plot als works the same was as a 2D scatter plot, but the only difference was in a 2D scatter plot, we have used only 2 features, whereas in a 3D scatter plot, we have to use 3 features(one on 'X' axis, one on 'Y' axis and the other on 'Z' axis).

For plotting, the maximum number of features we use is 3. Beyond that it is difficult to visualize and also represent it on a paper or by using any visualization software.

Below is an example of how a 3D scatter plot looks like when we use 'sepal_length', 'sepal_width' and 'petal_width'.



In this plot, we are representing 'sepal_length' on one axis, 'sepal_width' and 'petal_width' on two different axes.

## 20.3 Pairplots

As it is difficult for us to build scatter plots with more than 3 dimensions, we apply a hack here and this hack is known as pairplot. Here pair plots deal with plotting 2D scatter plots using all the 4 numerical features (taken 2 at a time). That is where we get 2D scatter plots with (SL, SW), (SW, SL), (SL, PL), (PL, SL), (SL, PW), (PW, SL), (SW, PL), (PL, SW), (PW, SW), (SW, PW), (PL, PW), (PW, PL) combinations. And we get histograms each for SL, SW, PL and PW features.

| | | |
|---|---|---|
| SL | → | Sepal Length |
| SW | → | Sepal Width |
| PL | → | Petal Length |
| PW | → | Petal Width |

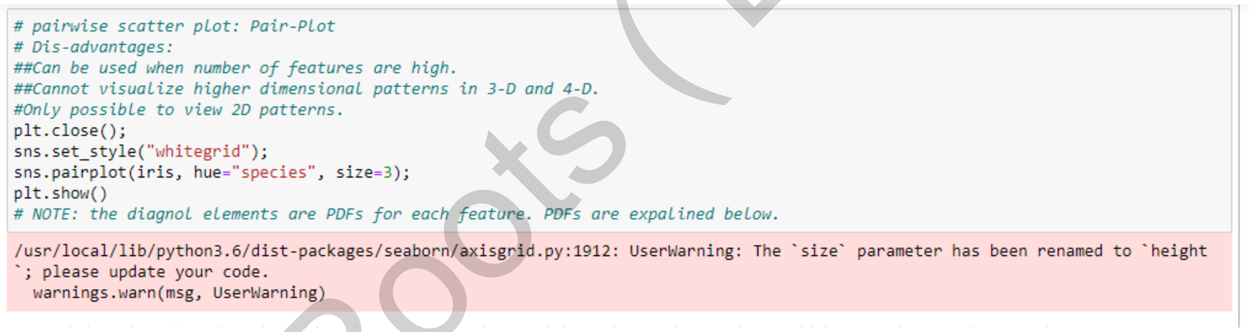
Here as we are using 4 numerical features for building pairplots,

Total number of scatter plots = $2*4C_2 = 2*6 = 12$

Total number of univariate plots(ie., histograms/PDF) = 4

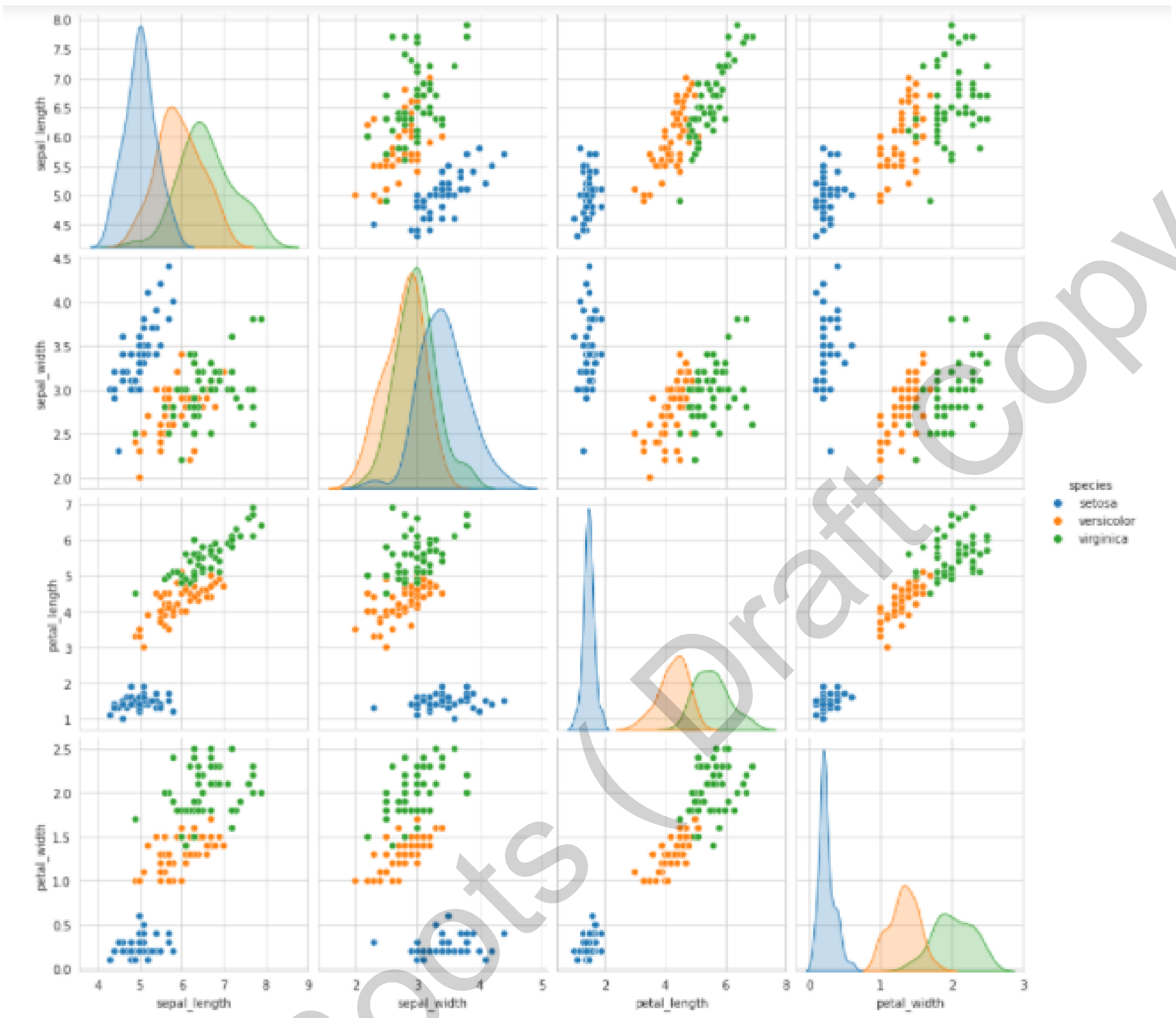
Total number of plots obtained = 12 + 4 = 16

The univariate plots occur in the diagonal positions among the plots. So let us look at the example that was discussed at the timestamp 0:05



```
# pairwise scatter plot: Pair-Plot
# Dis-advantages:
##Can be used when number of features are high.
##Cannot visualize higher dimensional patterns in 3-D and 4-D.
#Only possible to view 2D patterns.
plt.close();
sns.set_style("whitegrid");
sns.pairplot(iris, hue="species", size=3);
plt.show()
# NOTE: the diagnol elements are PDFs for each feature. PDFs are expalined below.
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/axisgrid.py:1912: UserWarning: The `size` parameter has been renamed to `height
`; please update your code.
  warnings.warn(msg, UserWarning)
```



From the above plots, we could easily conclude that the ‘Petal_Length’ and the ‘Petal_Width’ features are much more useful in identifying the IRIS flowers of different classes. The ‘Iris-setosa’ flowers can be easily distinguished and are easily separable whereas in case of ‘Iris-versicolor’ and ‘Iris-virginica’, there is a little overlap and they cannot be distinguished perfectly all the time.

When we look at the code above, we are first setting the background style to **‘whitegrid’** and then in the **sns.pairplot()** function, we are passing the dataframe as the first argument. We are passing **‘species’** as the value for the **‘hue’** parameter which indicates the points belonging to each class of the ‘species’ column should be represented in a different color. The **‘size’** parameter denotes the height of the plot.

When we clearly observe the plots, we can see the **‘petal_length’** and the **‘petal_width’** features play a key role in separating the **‘Iris-setosa’** flowers easily from the other two species. As we are able to separate the ‘Iris-setosa’ species from the other species easily, we can derive a condition from the scatter plot between ‘petal_length’ and ‘petal_width’ as

```
if petal_length<2 && petal_width<1:
    then ‘Iris-setosa’
```

Now for separating the points of ‘Iris-versicolor’ from the points of ‘Iris-virginica’, we can write the condition as

```
if peta_width<2 && petal_width>1 && petal_length<5 && petal_length>2.5:
    then ‘Iris-versicolor’
```

This condition still gives a few misclassifications, but in around 90-95% of the cases, it gives the correct classification.

So from the above plots, we can conclude that the ‘petal_length’ and the ‘petal_width’ features are majorly useful in classifying the points. Just by using these two features and simple if-else conditions, we could easily separate the flowers to a major extent. There are a few overlaps, but still it is accepted.

## 20.4 Limitations of Pairplots

As far as the number of dimensions in the dataset is small, we can go for pairplots. But if the number of dimensions is high, then we get a huge number of scatter plots and it is difficult to make analysis and draw conclusions from such a huge number of plots. This is a major disadvantage with the pairplots.

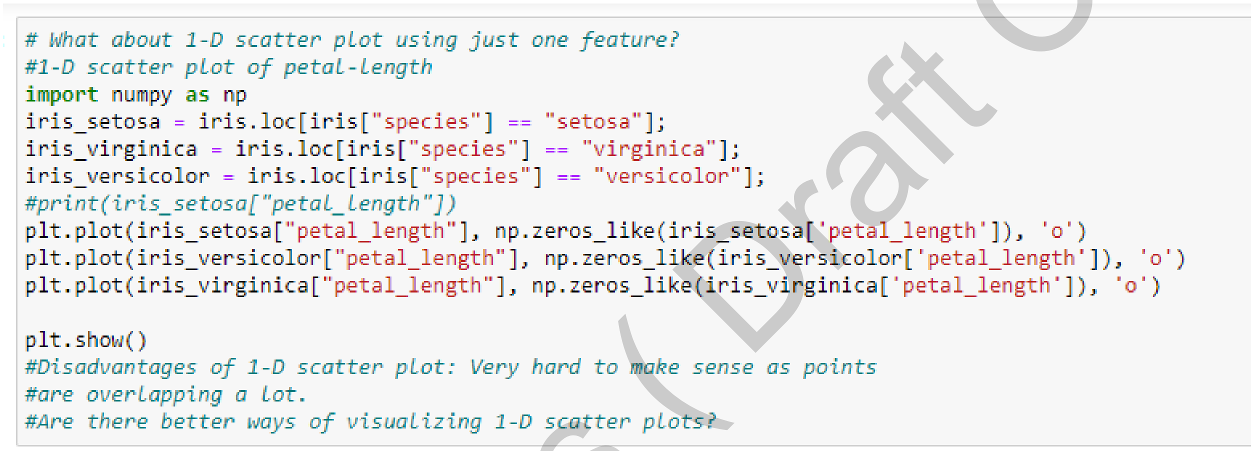
One way to get rid of this huge dimensionality is by reducing the number of dimensions. We have techniques like PCA, T-SNE, etc that reduce the dimensionality of the dataset and these techniques will be covered in this course in the future lectures.

## 20.5 Histogram and Introduction to PDF (Probability Density Function)

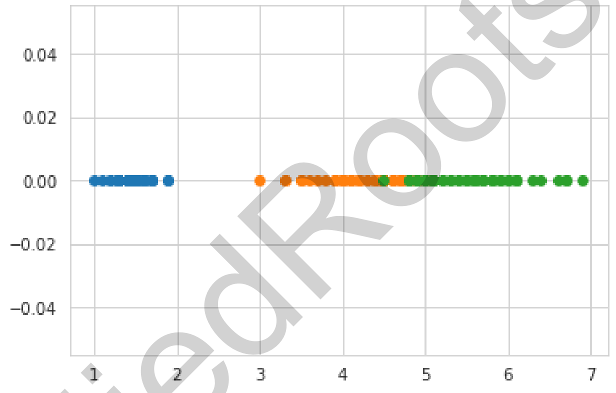
### 1D Scatter plots

So far we have seen 2D scatter plots, 3D scatter plots and pairplots. Now let us look at 1D scatter plot. We shall plot a 1D scatter plot using the **'petal_length'** column. We need at least 2 axes in order to represent a plot. So we shall take the **'petal_length'** values as the 'X' coordinates and zeros as the 'Y' coordinates.

Below is the code and the 1D scatter plot graph that was discussed starting from the timestamp 0:35.



```
# What about 1-D scatter plot using just one feature?
#1-D scatter plot of petal-length
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 a lot.
#Are there better ways of visualizing 1-D scatter plots?
```



In the above code, we are first separating the points belonging to each class into a separate data frame and then building the 1D scatter plots with them. As all the 'Y' coordinates are the same, we see all the points at the same level on the graph.

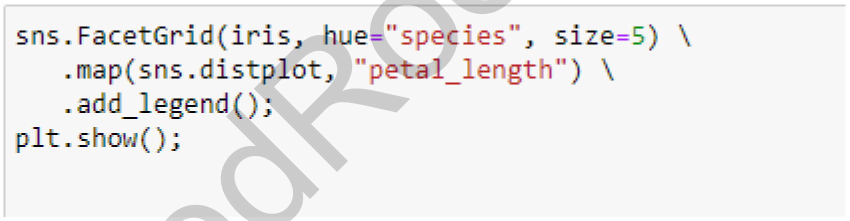
But the problem here with the 1D scatter plots is that we could see a huge overlap of the points belonging to the same classes. We do not know how many points of a particular class are present in an interval. In between 1 and 2 on the 'X' axis, we see a huge number of points of **'Iris-setosa'** class. Also we see a huge number of points of **'Iris-versicolor'** and **'Iris-virginica'** classes overlapping in between the values 3.5 and 6 on the 'X' axis. As we aren't clear about the number of points, we could not draw any conclusions from the 1D scatter plots. Hence in order to perform univariate analysis, we go with another type of plot known as **‘Histogram’**.

## Histogram

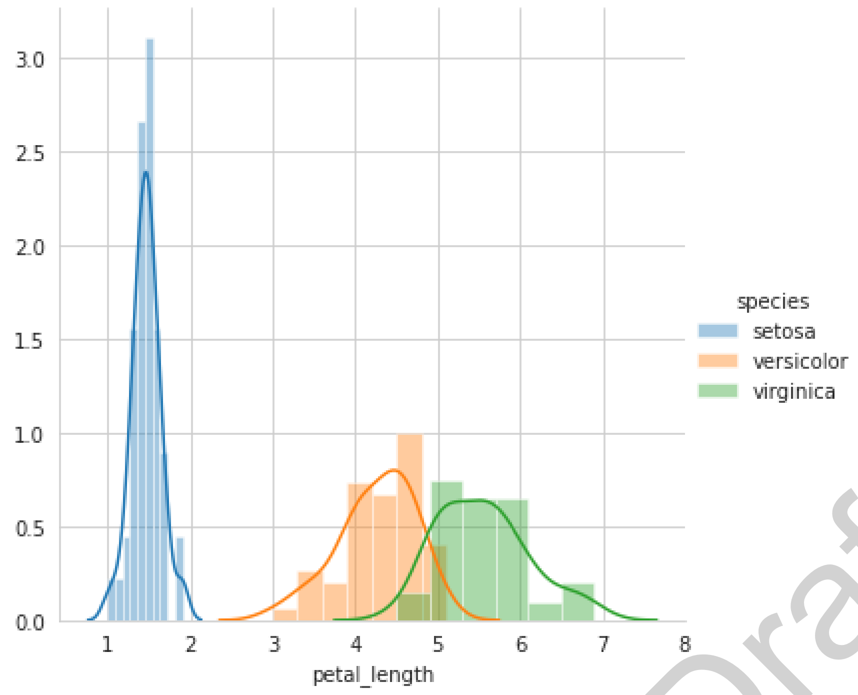
- ➢ A Histogram is an accurate representation of the distribution of numerical data.
- ➢ To construct a histogram, the first step is to divide the entire range of values into a series of intervals and then count how many values fall into each interval.
- ➢ The bins are usually specified as consecutive, non-overlapping intervals of a variable. The bins must be adjacent and are often of equal size. Histogram is used when we have a large number of data points.
- ➢ In a histogram, the ‘X’ axis represents the class interval and the ‘Y’ axis represents the class/bin frequencies. The number of points present in a bin is called bin frequency. The width of each bin is called bin width and it is the same for all the bins.
- ➢ The voice of the bin-width determines the number of class intervals. The shape of the histogram gets affected by the bin-width and the starting point.
- ➢ PDF and Histogram are the best tools to be used for univariate analysis. PDF is the smoothened form of a histogram.
- ➢ To smooth the histogram, we use something called Kernel Density Estimation(KDE) which results in the curve we are seeing for the PDF.

  **Note**: PDF and KDE will be discussed in much more detail in the Statistics chapter of the future lectures.

  Below is the code snippet to build a distribution plot and it was discussed at the timestamp 5:00.



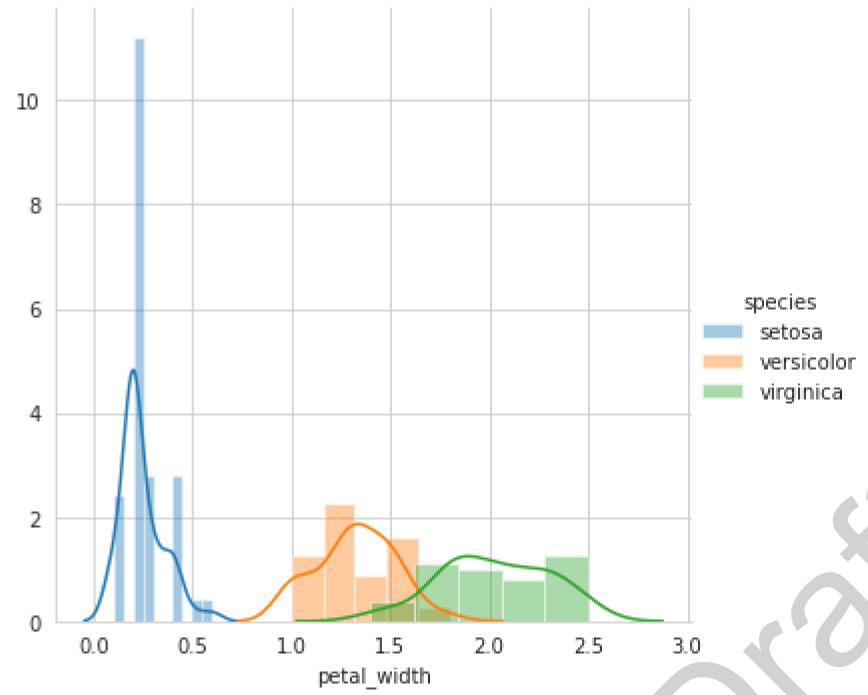
```
sns.FacetGrid(iris, hue="species", size=5) \
   .map(sns.distplot, "petal_length") \
   .add_legend();
plt.show();
```



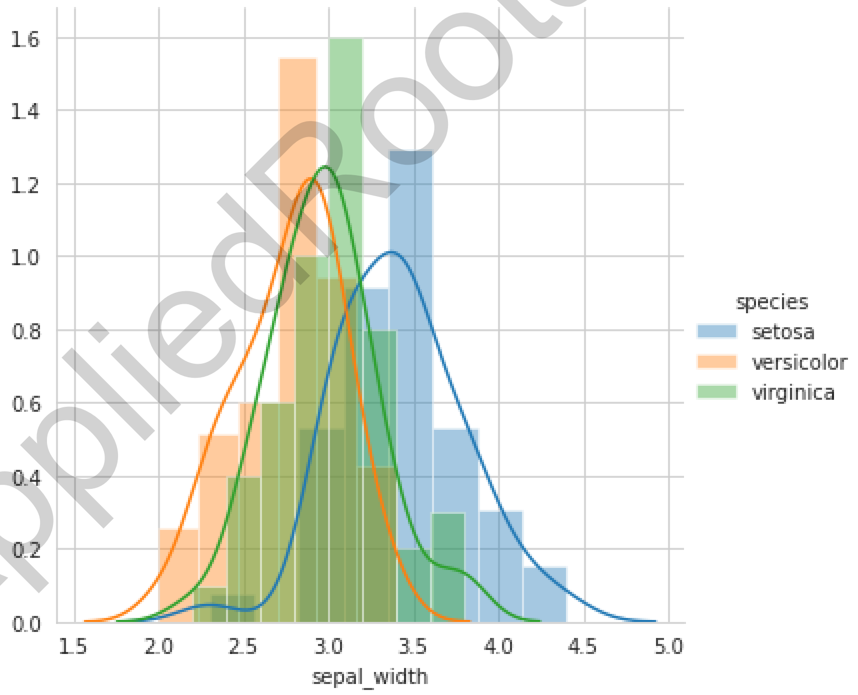
When we use the ‘petal_length’ feature, we could see the ‘Iris-setosa’ class gets separated easily. We can make the below conclusions.

```
if PL<2:
    Then ‘Setosa’
if PL < 4.7:
    Then ‘versicolor’
else:
    ‘Virginica’
```

```
sns.FacetGrid(iris, hue="species", size=5) \
   .map(sns.distplot, "petal_width") \
   .add_legend();
plt.show();
```

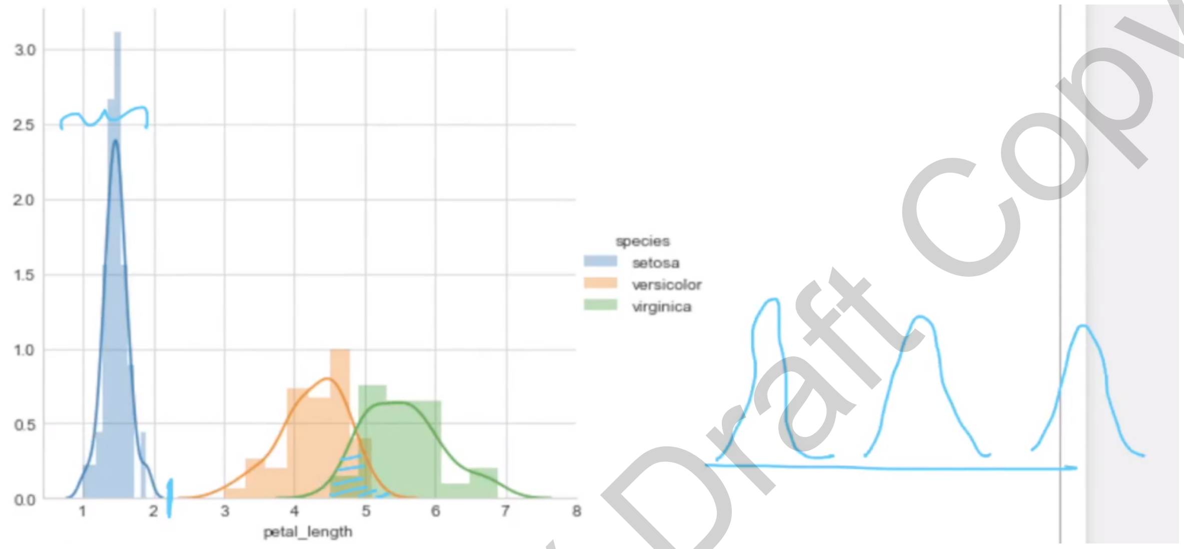


```
sns.FacetGrid(iris, hue="species", size=5) \
   .map(sns.distplot, "sepal_length") \
   .add_legend();
plt.show();
```

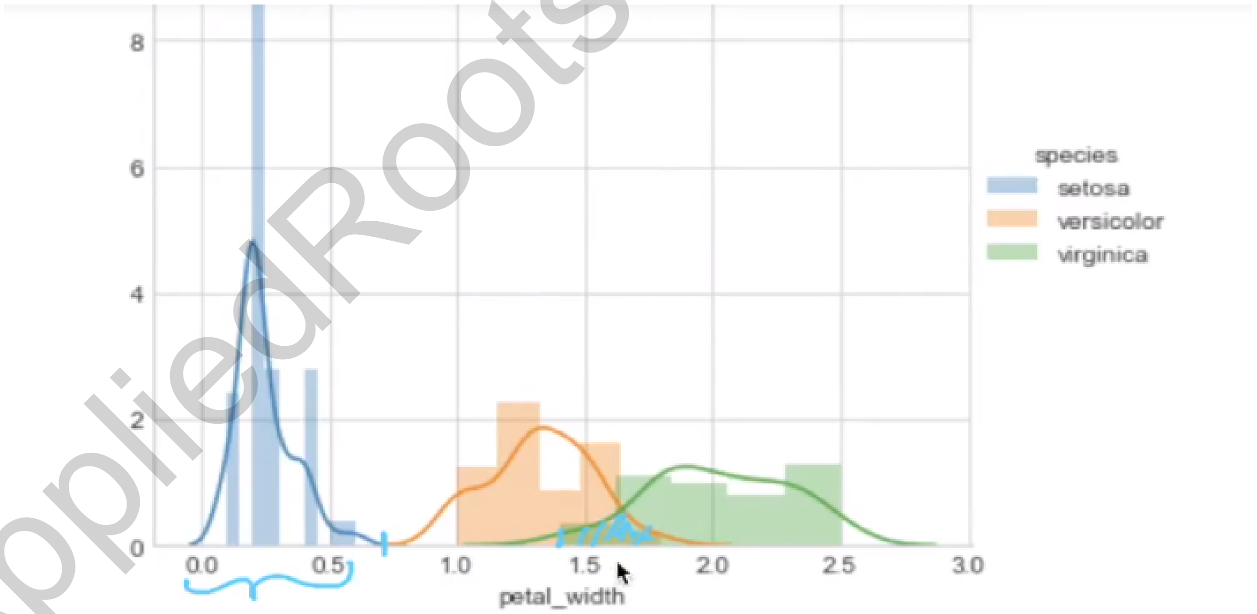


## 20.6 Univariate Analysis using density function

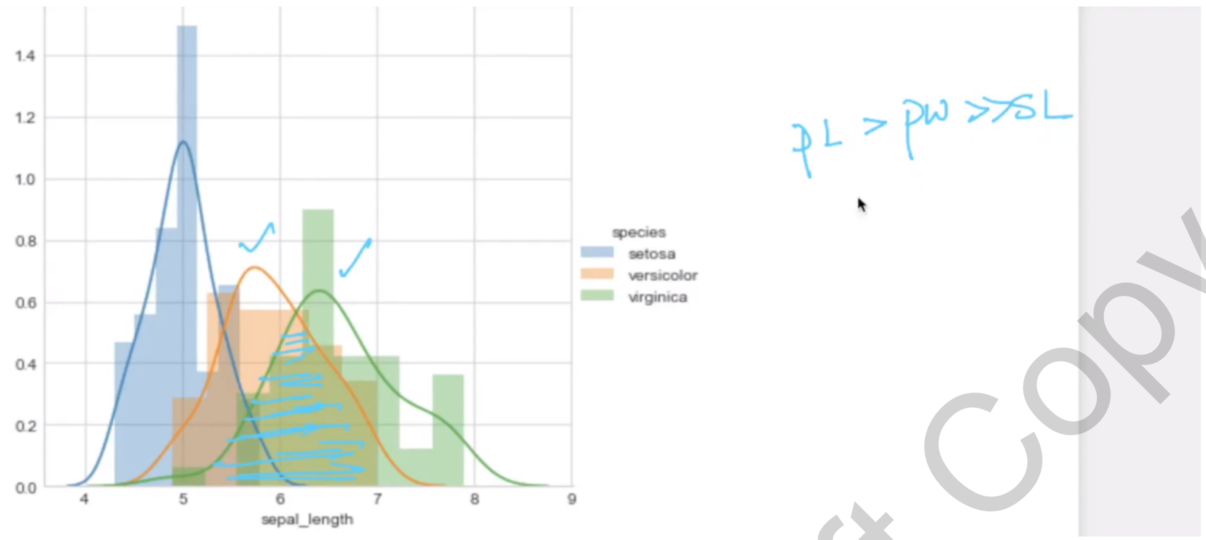
We use the univariate plots like PDFs, histograms to perform univariate analysis. Below are the plots of the univariate analyses of all the 4 features of Iris dataset and are discussed starting from the timestamp 0:50.



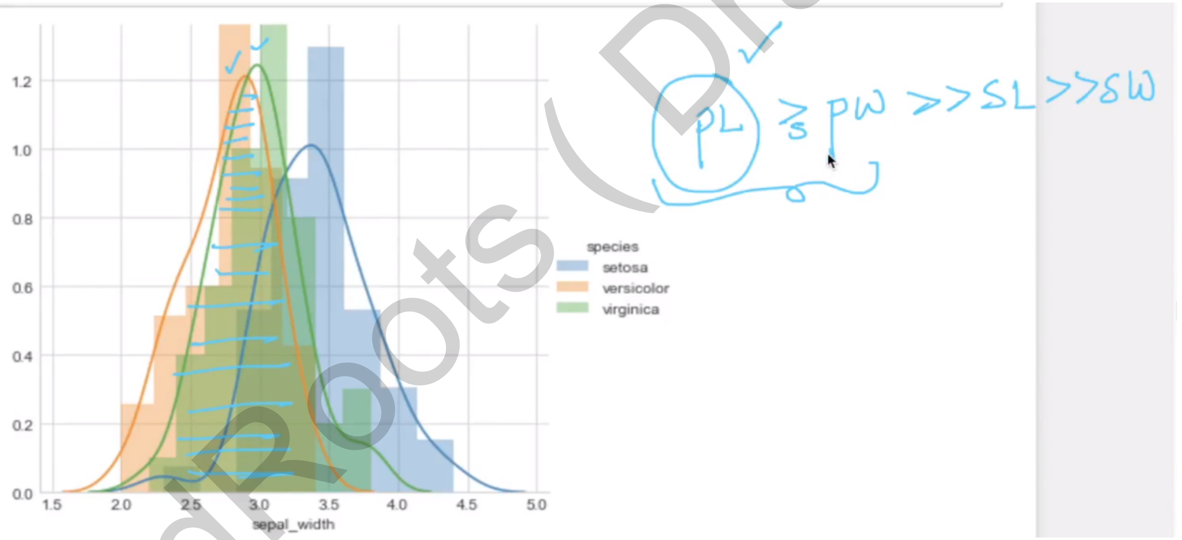
From the above plot, we can see that by using the 'petal_length' values, we could separate the 'Setosa' species from the other 2 species. We observe a little overlap between the 'Versicolor' and the 'Virginica' groups.



Even from the univariate analysis of the 'petal_width' column, we can clearly say that the 'Setosa' species can be separated from the other 2 species. Also we see a small overlap between the 'Versicolor' and 'Virginica' groups.



Here from the univariate analysis of the ‘sepal_length’, we observe there are overlaps not only between ‘Versicolor’ and ‘Virginica’, but also the ‘Setosa’ species points get overlapped with these two.



In the univariate analysis of ‘sepal_width’, the overlapped region is increased among all the 3 categories of species.

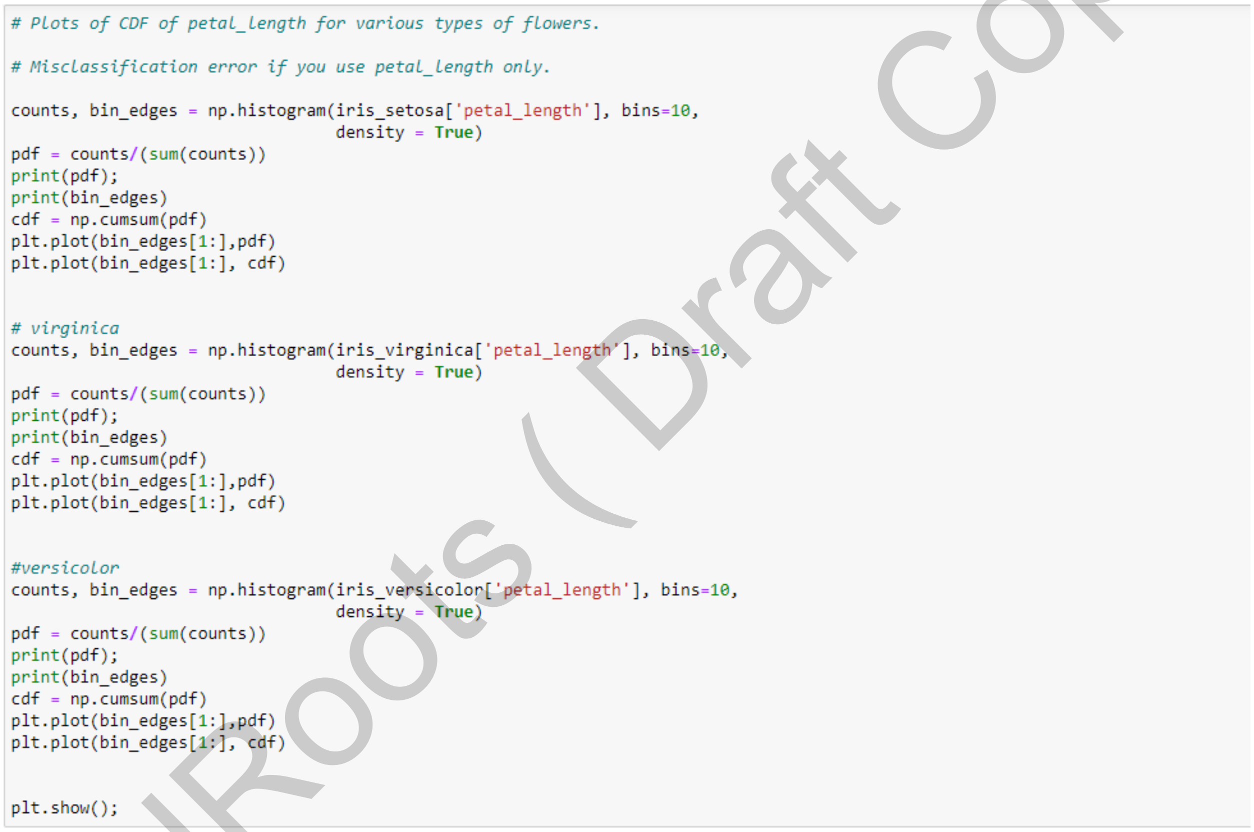
So we can say that in order to classify the points belonging to different species, the order of preference of the features, useful in decision making is

PL>PW>SL>SW

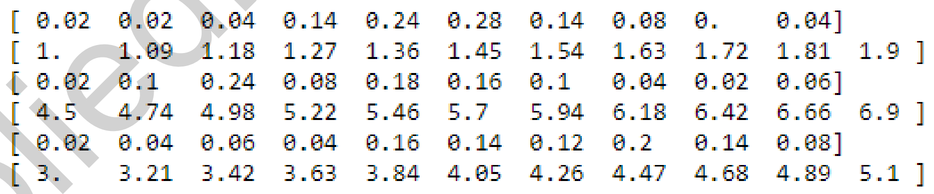
## 20.7 Cumulative Distribution Function (CDF)

- CDF value at a particular point ‘$x_1$’ on the ‘X’ axis denotes the probability.proportion of the points with the ‘x’ value less than or equal to ‘$x_1$’.
- The CDF value at a point ‘x’ is equal to the sum of probabilities of all values less than or equal to ‘x’.

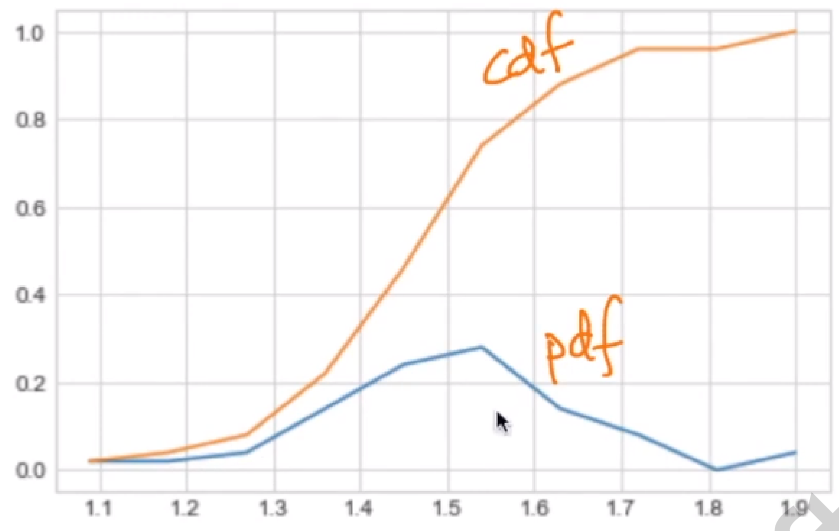
Below are the code snippet and the graphs for the CDF that were discussed starting from the timestamp 0:17.

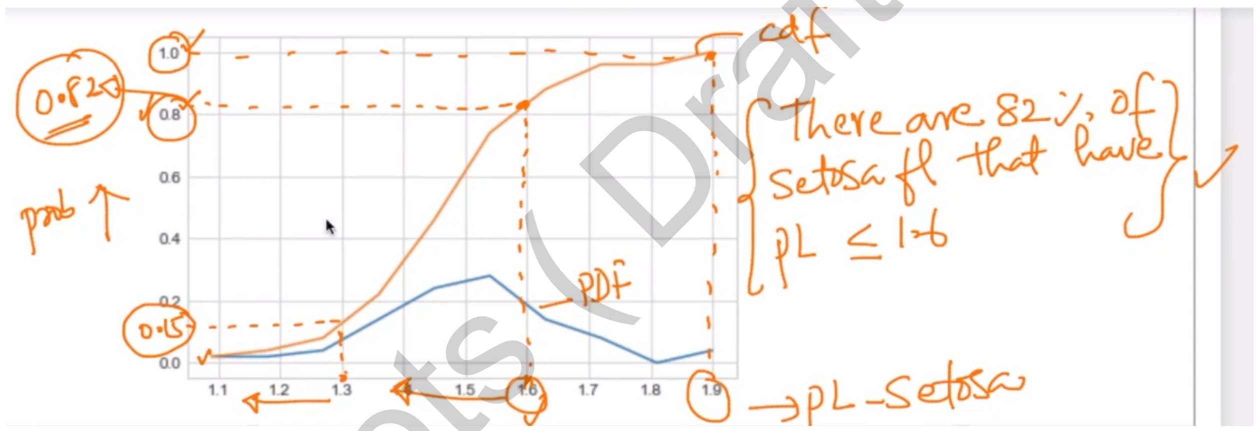


```
# Plots of CDF of petal_length for various types of flowers.

# Misclassification error if you use petal_length only.

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 ]
```

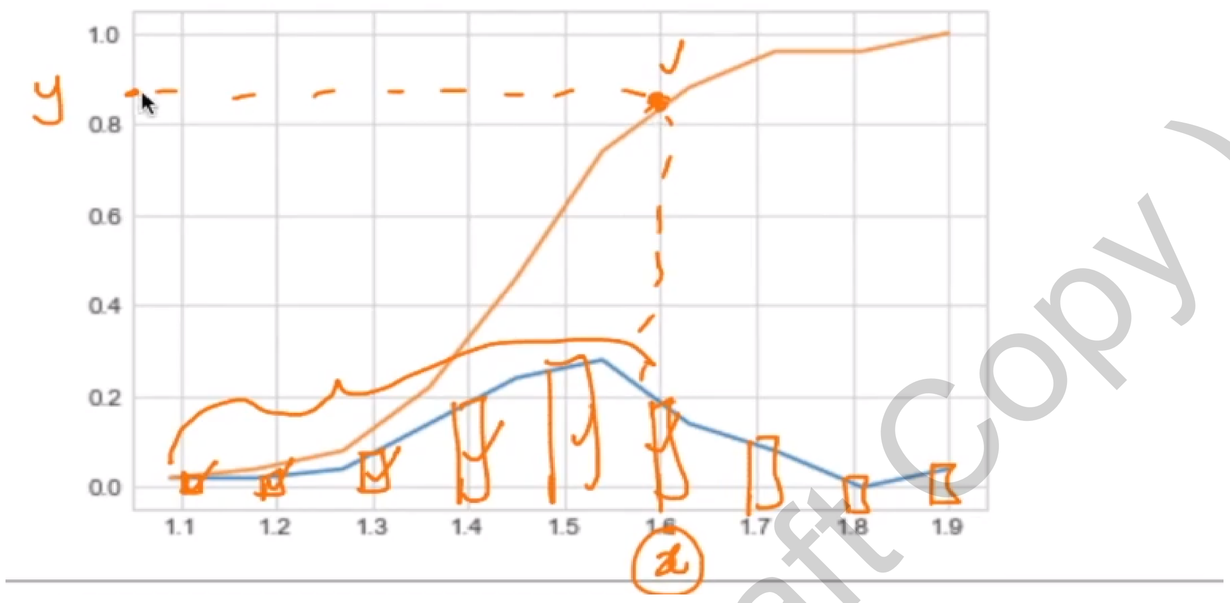




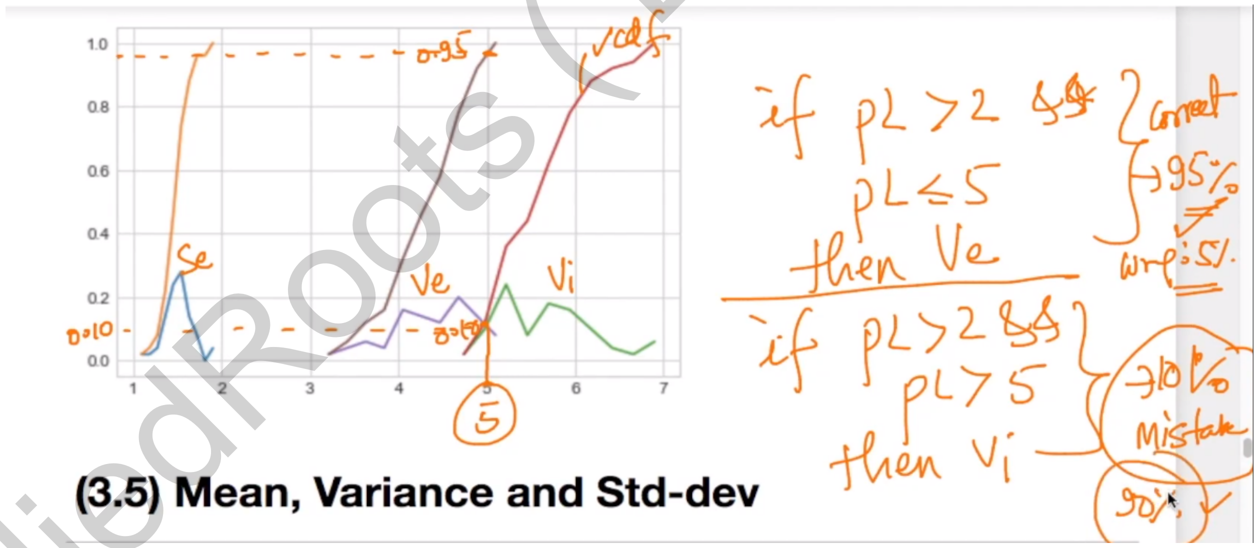
Here we are plotting the CDF and PDF for the ‘petal_length’ of the ‘Iris-setosa’ flower species.

The value of CDF for petal_length = 1.3 is 0.15. It means 15% of the ‘Iris-setosa’ flowers have a petal length of less than or equal to 1.3.

The value of CDF for petal_length = 1.6 is 0.82. It means 82% of the ‘Iris-setosa’ flowers have a petal length of less than or equal to 1.6.



Here the blue curve represents the PDF of the 'petal_length' values of 'Iris-setosa' flowers and the orange curve represents the CDF. The CDF value at a particular point is equal to the sum of all the PDF values present before that point.



(3.5) Mean, Variance and Std-dev

Here we make conclusions about the class labels as

**if PL<=2:**

**then 'Setosa'**

**if PL> 2 && PL<=5:**

**then 'Versicolor'**

**if PL>2 && PL>5:**

**then 'Virginica'**

We can get this information for decision making only with PDF, but not with the CDF.

## 20.8 Mean, Variance and Standard Deviation

### Mean

- ➢ Mean is the total sum of all the values divided by the total number of values.
- ➢ Mean is one of the central measures of central tendency.

Let us assume the values we have are $x_1, x_2, x_3, \ldots x_n$, then the mean is defined as

**Mean(μ) = $(x_1+x_2+x_3+\ldots+x_n)/n$**

Where 'n'→ number of points/values

### Standard Deviation

- ➢ Standard Deviation is the measure of the amount of variation of the values from the mean. Standard Deviation measures the spread of the data.
- ➢ Lower Standard Deviation indicates, the values are present closed to the mean, whereas higher Standard Deviation indicates, the values are present far away from the mean.

**Standard Deviation(σ) = $\text{sqrt}(((x_1-\mu)^2+(x_2-\mu)^2+(x_3-\mu)^2+\ldots+(x_n-\mu)^2)/n)$**

Where 'n'→ number of points/values, 'μ' → Mean

### Variance

- ➢ Variance is the squared deviation of the values from the mean.
- ➢ The more the variance is, the more the deviation would be. It is always recommended to have the variance as low as possible.
- ➢ Variance is the square of the Standard Deviation.

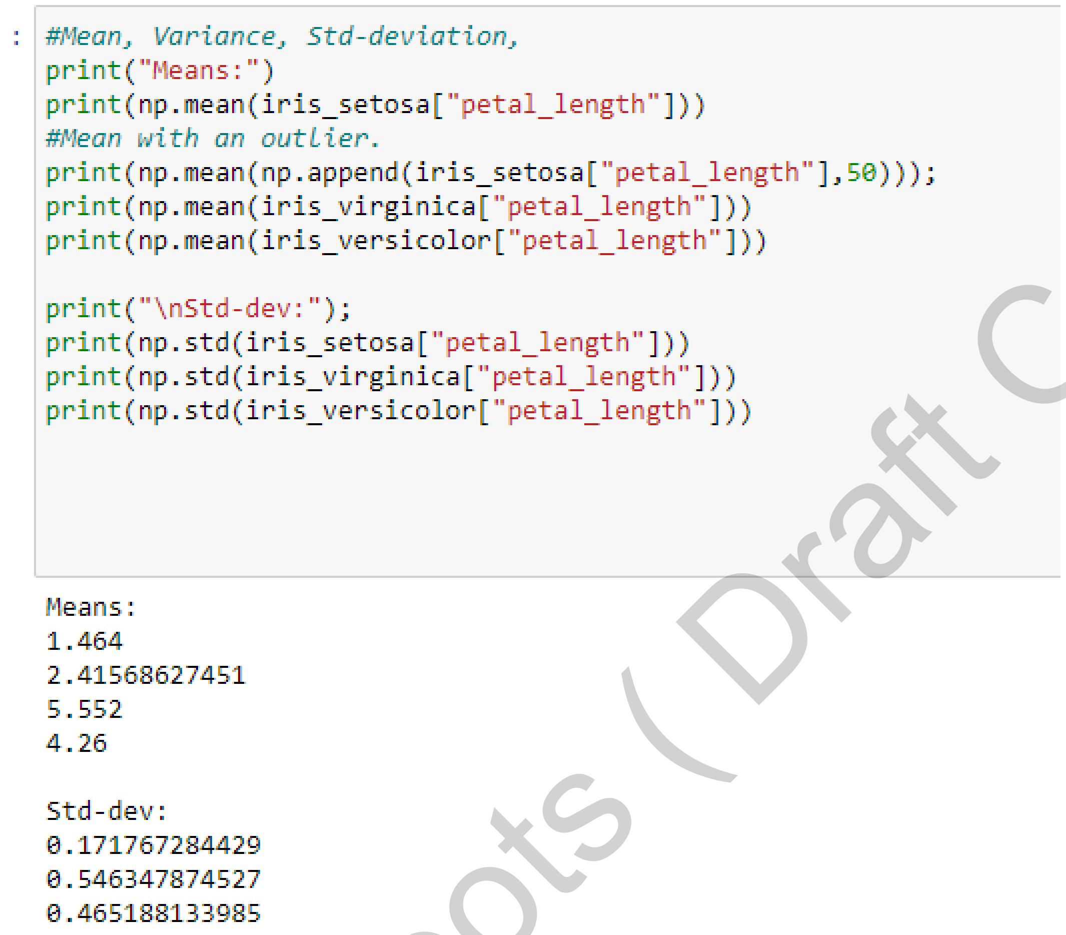
**Variance($\sigma^2$) = $((x_1-\mu)^2+(x_2-\mu)^2+(x_3-\mu)^2+\ldots+(x_n-\mu)^2)/n$**

Where 'n'→ number of points/values, 'μ' → Mean

### Outliers

- ➢ Outliers are not just the greatest and the least values, but the values that are very different from the pattern established by the rest of the data.
- ➢ The statistics like mean, standard deviation and variance are easily affected by the presence of Outliers.
- ➢ In such cases, we have to remove the outliers from the data and then perform compute the statistics like mean and standard deviation.
- ➢ In case, if you do not want to remove the outliers, then instead of the mean and the standard deviation, you have to go for the statistics like median and median absolute deviation, which aren't easily affected by the presence of the outliers.

Below is the code snippet that was discussed in the video starting from the timestamp 1:30.



```
#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
2.41568627451
5.552
4.26

Std-dev:
0.171767284429
0.546347874527
0.465188133985
```

In this code snippet, we are computing the mean for the **'petal_length'** column values of the **'Iris-setosa'**, **'Iris-versicolor'** and **'Iris-virginica'** species separately for each. In the 3rd statement, we are adding the value **'50'** which is an outlier to the **'petal_length'** column values and then when we compute the mean, we could see the mean got changed.

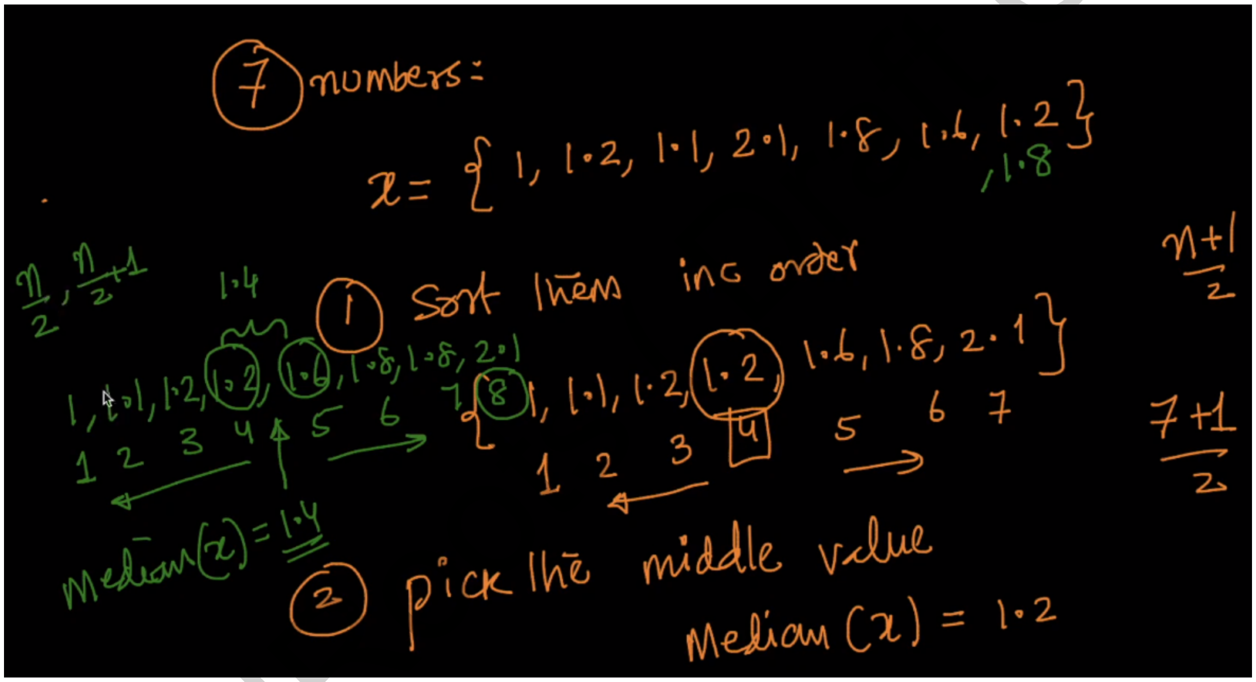
The metrics like Mean, Variance and Standard Deviation easily get affected with the presence of outliers in the data. Hence in order to handle this issue is

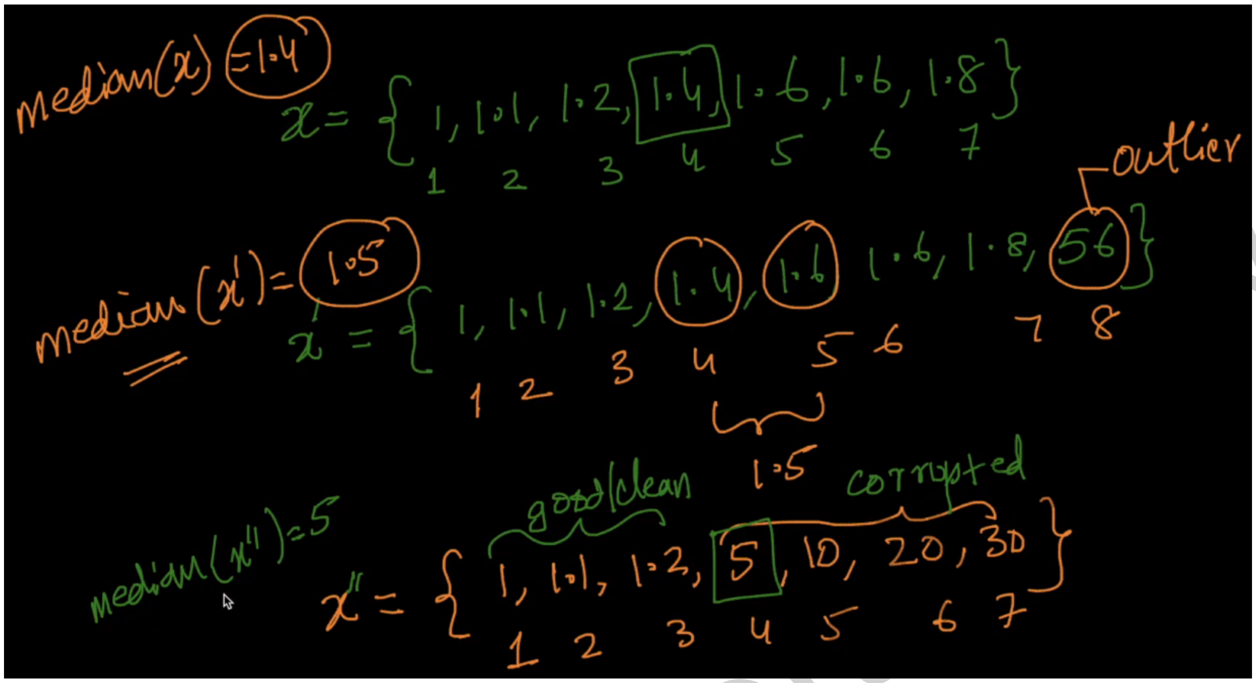
a) Remove the outliers from the data and then compute the statistics Mean and Standard Deviation.
b) If you do not want to remove the outliers, then better go for the statistics Median and Median Absolute Deviation.

## 20.9 Median

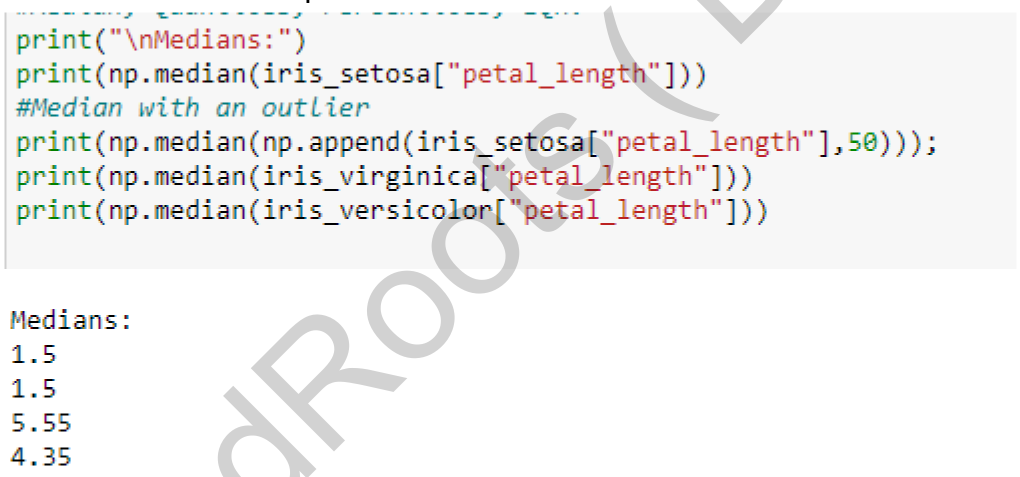
- ➢ Median is the value that lies in the middle among all the values/data points when the dataset is sorted.
- ➢ If the dataset has an odd number of points/values, then the middle value is the median.
- ➢ If the dataset has an even number of points/values, then the median is obtained by adding the two numbers in the middle and dividing the sum by 2.
- ➢ The median doesn't get affected even in the presence of outliers. Only if 50% or more values are outliers in the given dataset, then only the median gets affected.

Below are the examples that were discussed starting from the timestamp 2:45




median(x) = 1.4
x = { 1, 1.1, 1.2, 1.4, 1.6, 1.6, 1.8 }
1 2 3 4 5 6 7
outlier
median(x') = 1.5
x' = { 1, 1.1, 1.2, 1.4, 1.6, 1.6, 1.8, 56 }
1 2 3 4 5 6 7 8
1.5
good/clean
corrupted
median(x'') = 5
x'' = { 1, 1.1, 1.2, 5, 10, 20, 30 }
1 2 3 4 5 6 7

Below is the code snippet on how to compute the median in Python and it was shown at the timestamp 9:50.



```
print("\nMedians:")
print(np.median(iris_setosa["petal_length"]))
#Median with an outlier
print(np.median(np.append(iris_setosa["petal_length"],50)));
print(np.median(iris_virginica["petal_length"]))
print(np.median(iris_versicolor["petal_length"]))
```

```
Medians:
1.5
1.5
5.55
4.35
```

In the above code snippet, we are computing the median on the ‘petal_length’ column values for each species of Iris flowers.

## 20.10 Percentiles and Quantiles

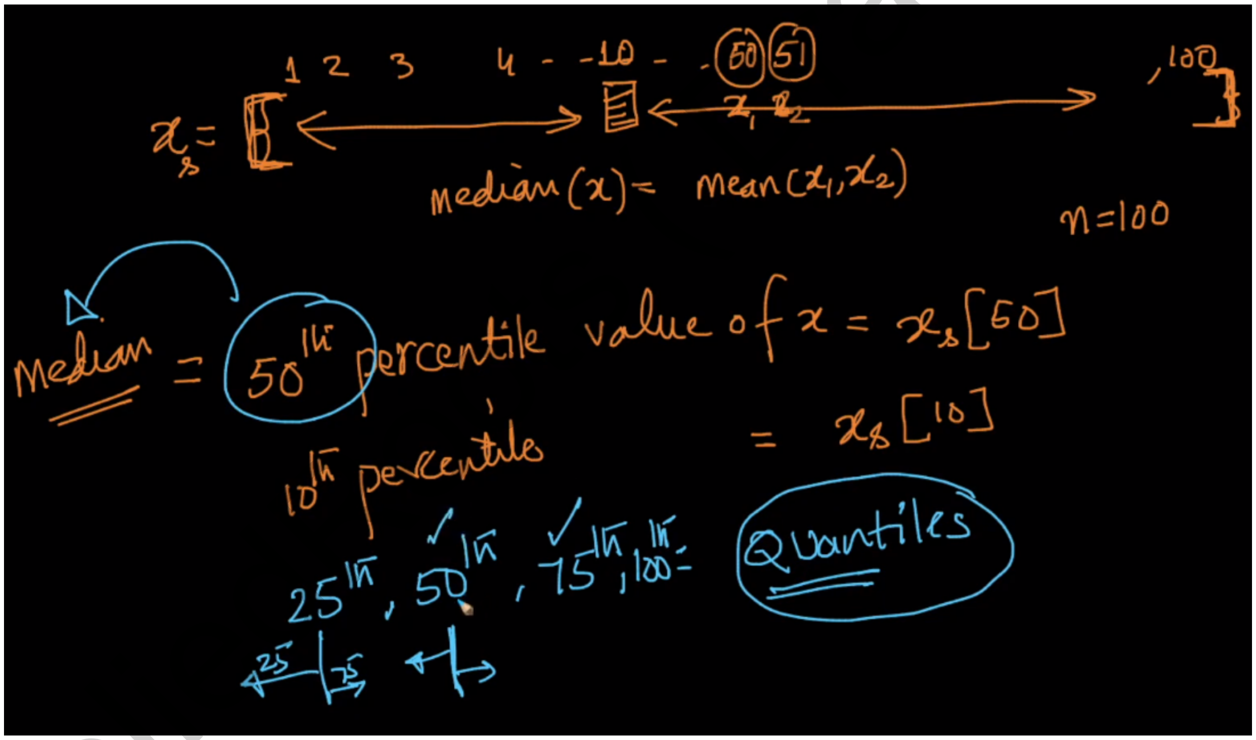
### Percentile

- ➢ Percentile is a score below which a certain specified percentage of points/values fall and the remaining points/values fall above this score.
- ➢ For example, the kth percentile indicates that about 'k'% of the points in the dataset fall below the kth percentile and the remaining '(100-k)'% of the points in the dataset fall above the kth percentile.

### Quantiles

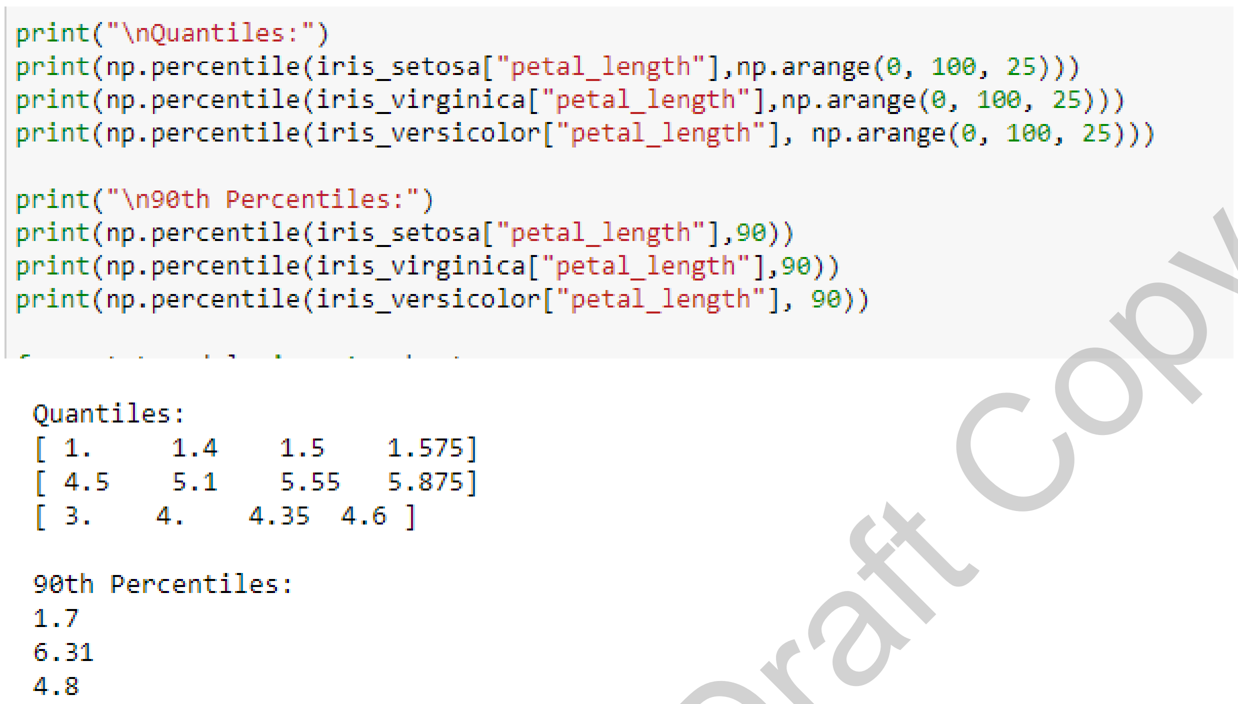
The 25th, 50th, 75th and the 100th percentiles are called the Quantiles.

The below example was discussed at the timestamp 0:25 where the concept of percentiles and quantiles was explained.



1 2 3 4 - -10 - - 50 51 , 100
$x_s$ =
median(x) = mean($x_1$,$x_2$)
n=100
Median = 50th percentile value of x = $x_s$[50]
10th percentile = $x_s$[10]
25th, 50th, 75th, 100th
Quantiles

Below is the code snippet discussed at the timestamp 4:25 which explains how to compute the percentiles and quantiles in Python.



```
print("\nQuantiles:")
print(np.percentile(iris_setosa["petal_length"],np.arange(0, 100, 25)))
print(np.percentile(iris_virginica["petal_length"],np.arange(0, 100, 25)))
print(np.percentile(iris_versicolor["petal_length"], np.arange(0, 100, 25)))

print("\n90th Percentiles:")
print(np.percentile(iris_setosa["petal_length"],90))
print(np.percentile(iris_virginica["petal_length"],90))
print(np.percentile(iris_versicolor["petal_length"], 90))
```

```
Quantiles:
[ 1.     1.4    1.5    1.575]
[ 4.5    5.1    5.55   5.875]
[ 3.    4.    4.35  4.6 ]

90th Percentiles:
1.7
6.31
4.8
```

In this example, we are computing the quantiles and the 90th percentile of the 'petal_length' column for each of the IRIS flower classes separately.

## 20.11 Interquartile Range (IQR) and Median Absolute Deviation (MAD)

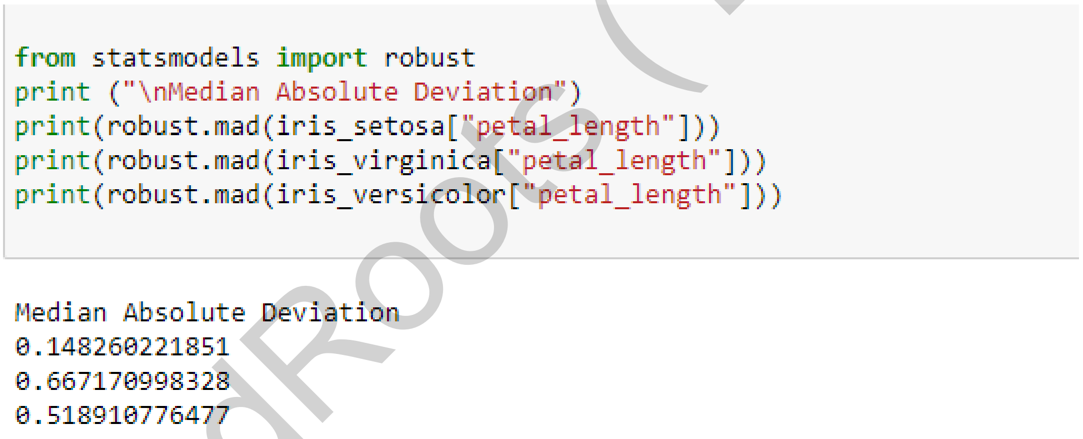
### Median Absolute Deviation (MAD)

- ➢ Median Absolute Deviation (MA
- ➢ D) is defined as the median of the deviations of the points from their median.
- ➢ MAD is also one of the measures of variability of quantitative data.
- ➢ As the mean and the standard deviation gets easily affected by the presence of outliers, we use median and MAD in place of mean and standard deviation.
- ➢ Median and the MAD are more robust to the outliers when compared to the mean and the standard deviation.

  Let ‘M’ be the median of all the values, then

  **MAD = median(|($X_i$-M)|)**
- ➢ Median is an alternative to the Mean and MAD is an alternative to the standard deviation.

Below is the code snippet that shows how to compute the MAD in Python and it was discussed at the timestamp 3:10.



```
from statsmodels import robust
print ("\nMedian Absolute Deviation")
print(robust.mad(iris_setosa["petal_length"]))
print(robust.mad(iris_virginica["petal_length"]))
print(robust.mad(iris_versicolor["petal_length"]))
```

```
Median Absolute Deviation
0.148260221851
0.667170998328
0.518910776477
```

We do not have direct functions to compute the MAD metric either in Numpy or in Scipy. Hence we are importing the ‘robust’ and are using the mad() function in it, to compute the MAD metric.

### Interquartile Range(IQR)

- ➢ IQR is defined as the difference between the 75th and the 25th percentiles. That is, the difference between the 3rd and the 1st quartiles.

  **IQR = $Q_3$ - $Q_1$**
- ➢ IQR is a rule of thumb used to identify the outliers present in the dataset.

According to the rule of thumb, if we have any points lying outside the interval **[$Q_1$-(1.5*IQR), $Q_3$+(1.5*IQR)]**, then those points are termed out as the outliers.
Also there are many other techniques like Local Outlier Factor(LOF), RANSAC, etc which are used for detecting the outliers and these techniques are discussed along with the Machine Learning topics.
**Note**: LOF, RANSAC are used when the data points are in vector form, whereas the IQR is used if the data points are scalars.

## 20.12 Boxplot with Whiskers

- ➢ So far we have learnt about percentiles, quantiles and the Interquartile range. Now when we look at the Histograms/PDFs while performing the univariate analysis, we could see the frequency distribution of the values and the spread of the values, but we could not obtain the 25th, 50th, 75th percentiles and the Interquartile range from such plots.
- ➢ The CDFs can give these values, but not directly. We manually have to check at what value of a numerical feature, is the CDF value equal to 0.25, 0.5 and 0.75. Though the Histograms/PDFs could explain the density of the points, they could not give any information about the quantiles. So in order to obtain the quantiles and the Interquartile range, we do have some plots and boxplot is one among such plots.
- ➢ The kind of analysis we can make with a boxplot is checking how many points are having the values below the 25th percentile, how many points have the values above the 75th percentile and how many points have the values in the IQR, etc.
- ➢ Regarding the whiskers, there is no standard way of drawing it. Typically, we take **($Q_3$+(1.5*IQR))** as the value for the upper whisker and **($Q_1$-(1.5*IQR))** as the value for the lower whisker.
- ➢ A Boxplot displays the summary of a large amount of data in five numbers. These numbers include the Median, the Upper Quantile($Q_3$), the Lower Quantile($Q_1$), the minimum and the maximum data values.

### Advantages of Boxplot

1) **Handles Large Data Easily**

 Organizing the data in a boxplot by using five concepts is an efficient way of dealing with large data, which is unmanageable for the other plots like a line plot or a stem plot or a leaf plot.

2) **Exact values are not retained**

 The boxplot doesn't keep the exact values and the details of the distribution results. A box plot shows only a simple summary of the distribution of results, so that we can quickly view it and compare it with the other data.

 We have to use a box plot in combination with other statistical graph methods like histogram for more thorough and detailed analysis of the data.

3) **Clear Summary**

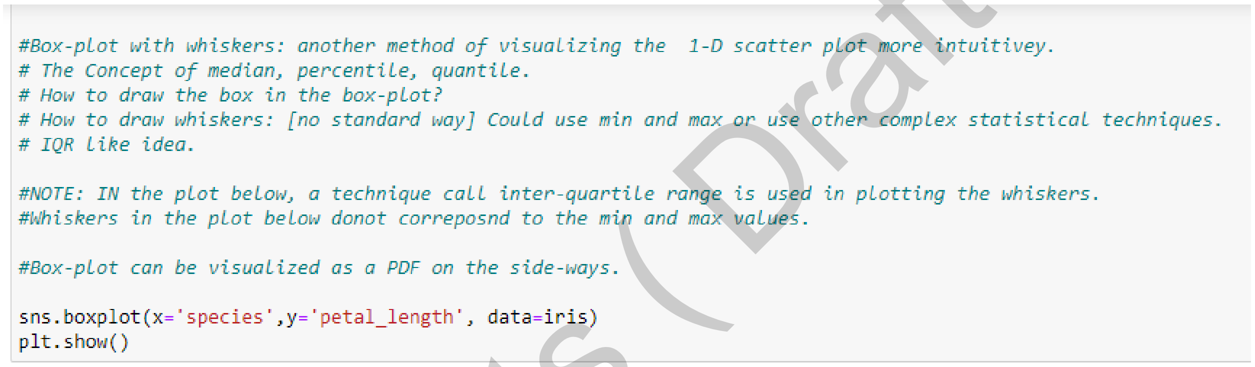
 A box plot is a highly visually effective way of viewing a clear summary of one or more sets of data.It is particularly useful for quickly summarizing and comparing different sets of results from different experiments. A box plot allows a graphical display of the distribution of results and provides indications of symmetry within the data.

4) **Displays Outliers**

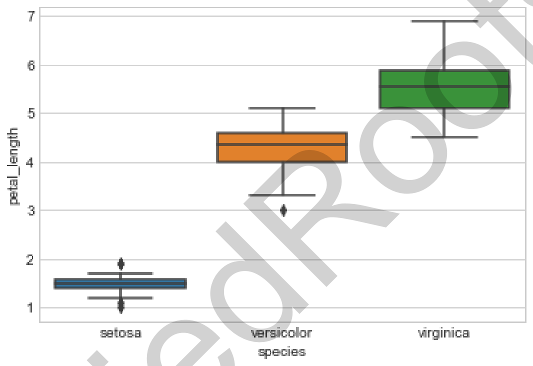
A box plot is one of very few statistical graph methods that show outliers. There might be one outlier or multiple outliers within a set of data, which occurs both below and above the minimum and maximum data values.

By extending the minimum and the maximum data values by 1.5 times the Interquartile range, the box plot delivers outliers. Any values that fall outside of the extended minimum and the maximum values, are known as outliers and these outliers can be easily determined using a box plot.

Below is an example of the code snippet and the graph of a boxplot that was discussed starting from the timestamp 1:20.

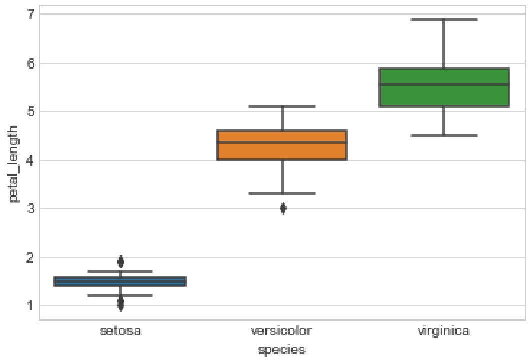


```
#Box-plot with whiskers: another method of visualizing the  1-D scatter plot more intuitivey.
# The Concept of median, percentile, quantile.
# How to draw the box in the box-plot?
# How to draw whiskers: [no standard way] Could use min and max or use other complex statistical techniques.
# IQR like idea.

#NOTE: IN the plot below, a technique call inter-quartile range is used in plotting the whiskers.
#Whiskers in the plot below donot correposnd to the min and max values.

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

sns.boxplot(x='species',y='petal_length', data=iris)
plt.show()
```



In this video, we are plotting a boxplot graph for the 'peta_length' column values for different classes of the 'Iris' flowers.

```
#Box-plot with whiskers: another method of visualizing the  1-D scatter plot more intuitivey.
# The Concept of median, percentile, quantile.
# How to draw the box in the box-plot?
# How to draw whiskers: [no standard way] Could use min and max or use other complex statistical techniques.
# IQR like idea.

#NOTE: IN the plot below, a technique call inter-quartile range is used in plotting the whiskers.
#Whiskers in the plot below donot correposnd to the min and max values.

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

sns.boxplot(x='species',y='petal_length', data=iris)
plt.show()
```



Here the ‘x’ parameter in **sns.boxplot()** denotes the class column with class values. A separate boxplot gets created for each of the classes.

The ‘y’ parameter takes the numerical column, which we use in plotting the boxplot graph. The ‘data’ parameter takes the dataframe as an input.

## 20.13 Violin Plot

Violin plots are the plots that have the benefit of both box plot and Probability Density Function (PDF). It is a combination of both these plots.

### Advantages of Violin Plot

1) **Violin plot is like a box plot but better**

    Box plots show Median, ranges, variations effectively. They allow comparing groups of different sizes. But the box plots are misleading. They aren't affected by the data's distribution. Whenever the data changes, the statistical summaries (like the median and the range) might change, but the box plots remain the same.

    This is when we have to use a violin plot. A violin plot carries all the information that a box plot would carry and it literally has a box plot inside the violin, but doesn't fall into the distribution trap.

2) **Violin graph is like a density plot, but better**

    The violin shape of the violin plot comes from the data's density plot. We can just turn that density plot sideway and put it on both sides of the box plot, mirroring each other.

    Reading the violin shape is exactly how we read a density plot. The thicker part indicates the values in that section of the violin have higher frequency and the thinner part implies lower frequencies.

3) **Violin graph is visually intuitive and attractive**

    To compare different sets, their violin plots are placed side by side. They don't sit on top of one another. Violin plots are easy to read. The dot in the middle represents the median. The box represents the interquartile range. The shape of the violin displays the frequencies of the values.

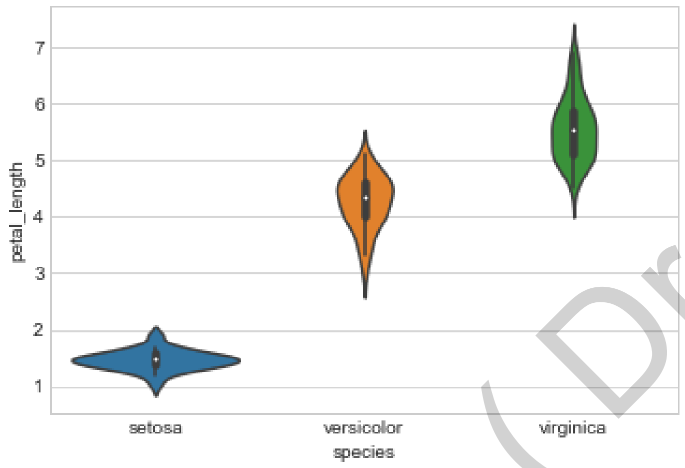
    Different violins are the different sets. We don't even have to fill the chart with colors and patterns to distinguish sets, which actually makes it less distracting and easier to read.

4) **Violin graph is non-parametric**

    Unlike the bar graphs with the mean and the error bars, violin plots contain all data points.This make them an excellent tool to visualize samples of small sizes.Violin plots are perfectly appropriate even if the data doesn't come from the normal distribution. They work well to distinguish both the qualitative and the quantitative data.

Below is an example of the violin plot and it's code snippet that were discussed starting from the timestamp 1:20.

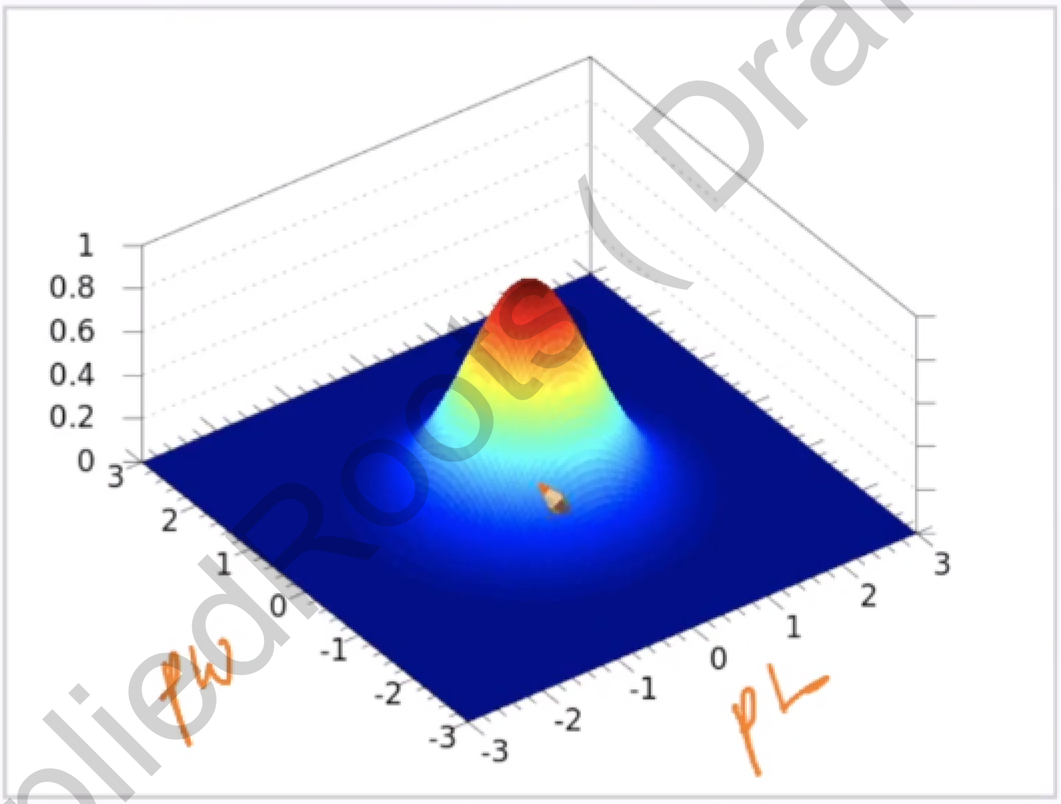
```
# A violin plot combines the benefits of the previous two plots
#and simplifies them

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

sns.violinplot(x="species", y="petal_length", data=iris, size=8)
plt.show()
```



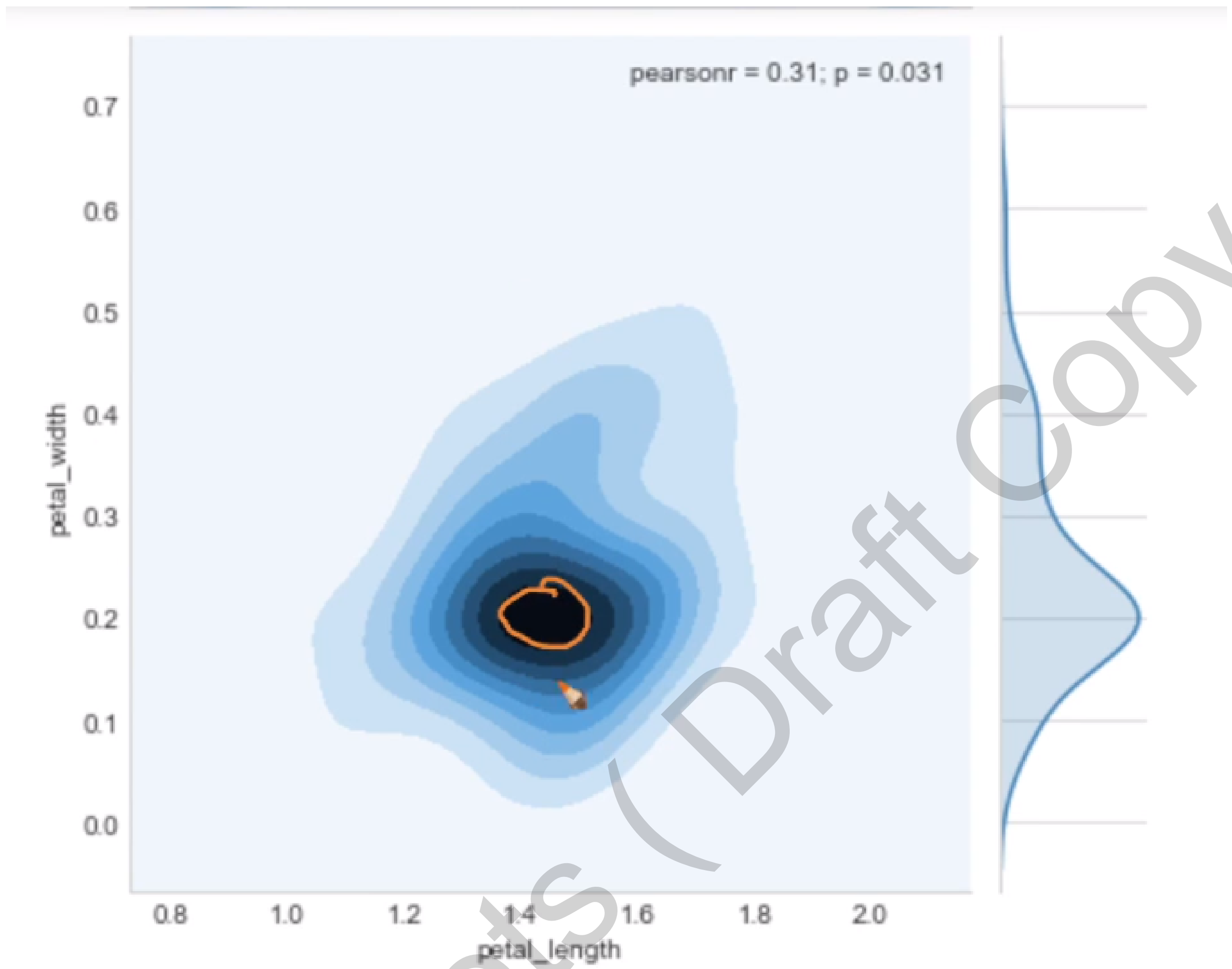
## 20.15 Multivariate Probability Density, Contour Plot

So far we have learnt about 2D density plots with the density distribution of only one variable. Now the Contour plot here deals with plotting the density functions using 2 variables at a time. This plot is now a 3D plot, with values of one variable on the ‘X’ axis and the other variable values on the ‘Y’ axis and the PDF values on the ‘Z’ axis. Below is an example of code snippet and the graph of a contour plot.

```
#2D Density plot, contors-plot
sns.jointplot(x="petal_length", y="petal_width", data=iris_setosa, kind="kde");
plt.show();
```



Gaussian curve with a 2-dimensional domain



In this plot, we can see the darker region has more density of points when compared to the lighter region. Also all the points present in a particular region with the same color are at the same height. The PDFs present at the top and the right side of this contour plots are the PDFs of the individual features 'petal_length' and 'petal_width'.