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

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: p
  import pandas.util.testing as tm
```

Exploratory Data Analysis

- Importance of domain knowledge.
- Data Set
- `Features` / `input` are all dependent
- `Output variable` - independent
- Objective

IRIS Dataset

Objective: Classify a new flower as belonging to one of the 3 classes given the 4 features.

```
!wget https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/data/iris.csv
```

```
--2021-05-20 10:37:09--  https://raw.githubusercontent.com/uiuc-cse/data-fa14/gh-pages/da
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.1
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443.
HTTP request sent, awaiting response... 200 OK
Length: 3716 (3.6K) [text/plain]
Saving to: 'iris.csv'

iris.csv            100%[===================>]   3.63K  --.-KB/s    in 0s

2021-05-20 10:37:10 (57.3 MB/s) - 'iris.csv' saved [3716/3716]
```

```
iris = pd.read_csv("iris.csv")
iris.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
iris.shape
```

```
(150, 5)
```

```
iris.columns
```

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

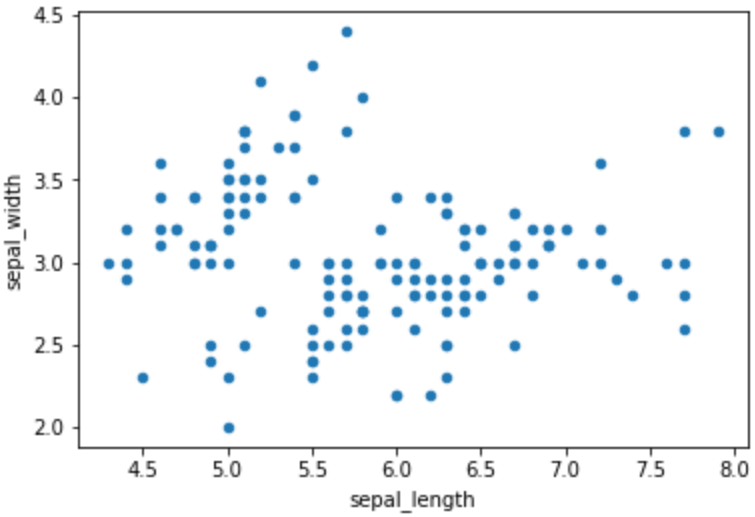
```
#How many data points for each class are present?
iris['species'].value_counts()
#balanced_dataset as all category has equal inputs
```

```
virginica     50
versicolor    50
setosa        50
Name: species, dtype: int64
```

2D scatter plot

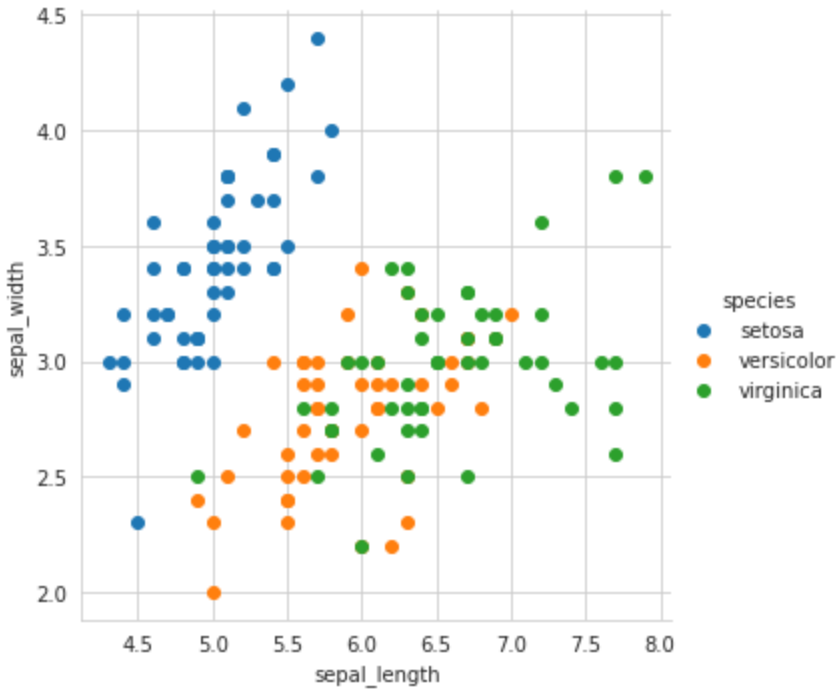
- Always see what is the `scale` of the plot
- What is `x-axis` and `y-axis` values?

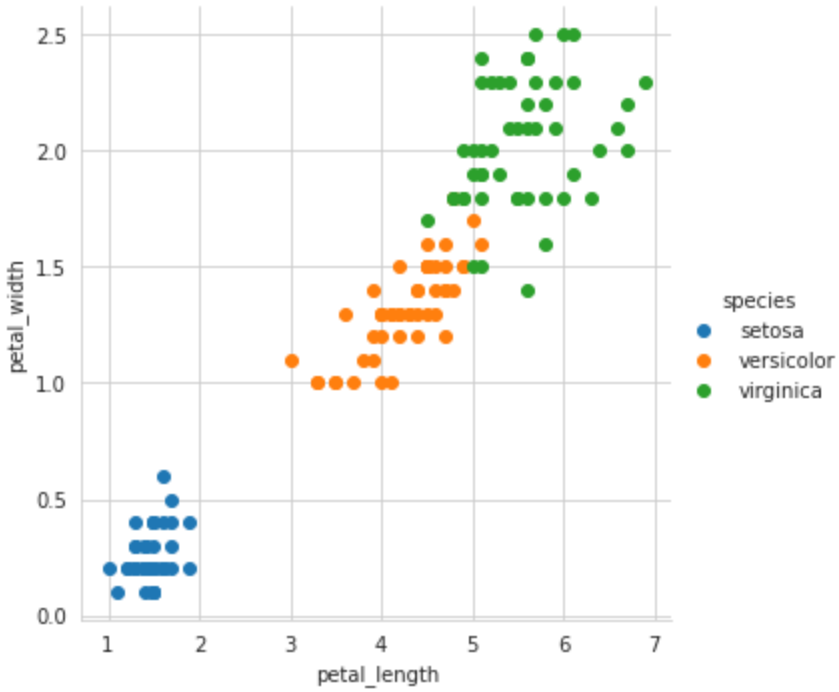
```
iris.plot(kind='scatter', x='sepal_length', y='sepal_width')
plt.show()
```




```
sns.set_style("whitegrid")
sns.FacetGrid(iris, hue="species", height=5) \
  .map(plt.scatter, 'sepal_length', 'sepal_width') \
  .add_legend()
plt.show()

sns.set_style("whitegrid")
sns.FacetGrid(iris, hue="species", height=5) \
```

```
  .map(plt.scatter, 'petal_length', 'petal_width') \
  .add_legend()
plt.show()
```





Observations

1. Using sepal_length and sepal_width features, we can distinguish Setosa flowers from others.
2. Seperating Versicolor from Viginica is much harder as they have considerable overlap.

3D Scatter plot

https://plot.ly/pandas/3d-scatter-plots/

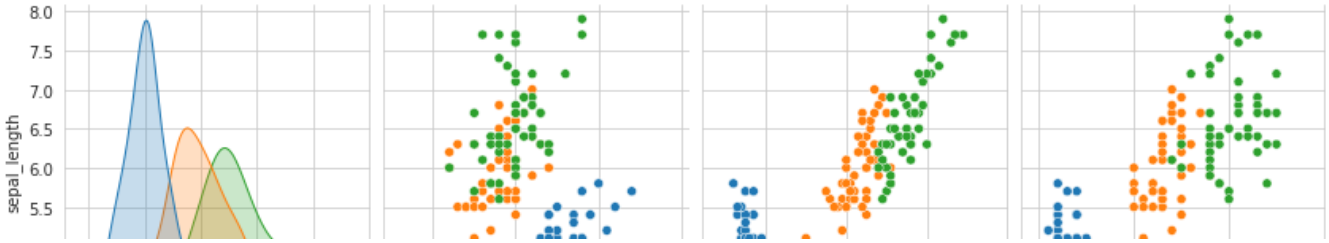
Needs a lot to mouse interaction to interpret data.

What about 4-D, 5-D or n-D scatter plot?

Pair Plot

For n>4 or n>5, pair plots are not much useful as the plot will grow a lot. Because it needs to compute nC_2 graphs

```
sns.set_style("whitegrid")
sns.pairplot(iris, hue='species', height=3)
plt.show()
```



Observations

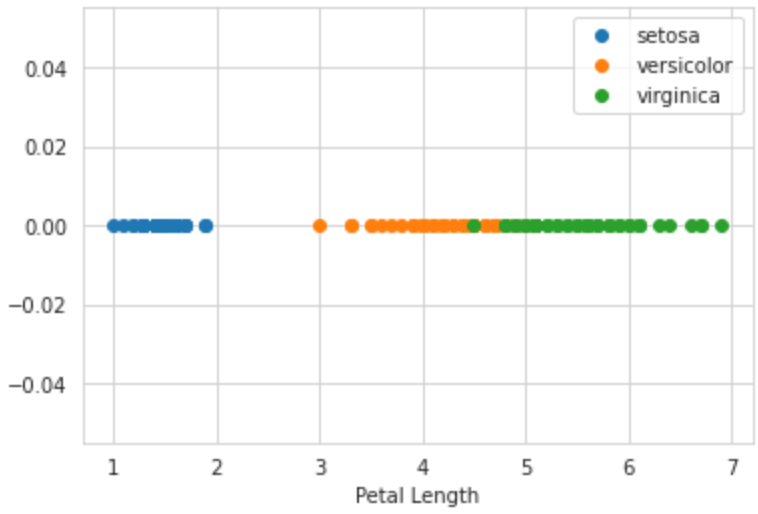
1. petal_length and petal_width are the most useful features to identify various flower types.
2. While Setosa can be easily identified (linearly seperable), Virnica and Versicolor have some overlap (almost linearly seperable).
3. We can find "lines" and "if-else" conditions to build a simple model to classify the flower types.

Histogram, CDF, PDF

1D Scatter Plot

Disadvantages of 1-D scatter plot: Very hard to make sense as points. 1D scatter plots are very hard to read. Histogram tells the density of values.

```
#1D  plot
setosa = iris[iris['species'] == 'setosa']
versicolor = iris[iris['species'] == 'versicolor']
virginica = iris[iris['species'] == 'virginica']

plt.plot(setosa['petal_length'], np.zeros_like(setosa['petal_length']), 'o', label='setosa')
plt.plot(versicolor['petal_length'], np.zeros_like(versicolor['petal_length']), 'o', label='ve
plt.plot(virginica['petal_length'], np.zeros_like(virginica['petal_length']), 'o', label='virg
plt.xlabel("Petal Length")
plt.legend()
plt.show()
```



Histogram

```
sns.FacetGrid(iris, hue='species', height=5) \
    .map(sns.distplot, 'petal_length') \
    .add_legend()
sns.FacetGrid(iris, hue='species', height=5) \
    .map(sns.distplot, 'petal_width') \
    .add_legend()
plt.show()

#lines drawn over histogram are Probability Distribution function -> smoothed form of histogra
#Observations:
# there is no setosa flower with petal length > 2 and petalwidth > 1.5
# if pl > 4.7, then it is virginica. Bcoz green bar height is more than orange. Some of the ou
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
```

Univariate Analysis

Eg : Which of my 4 variable is more useful than other variables for the categorization.

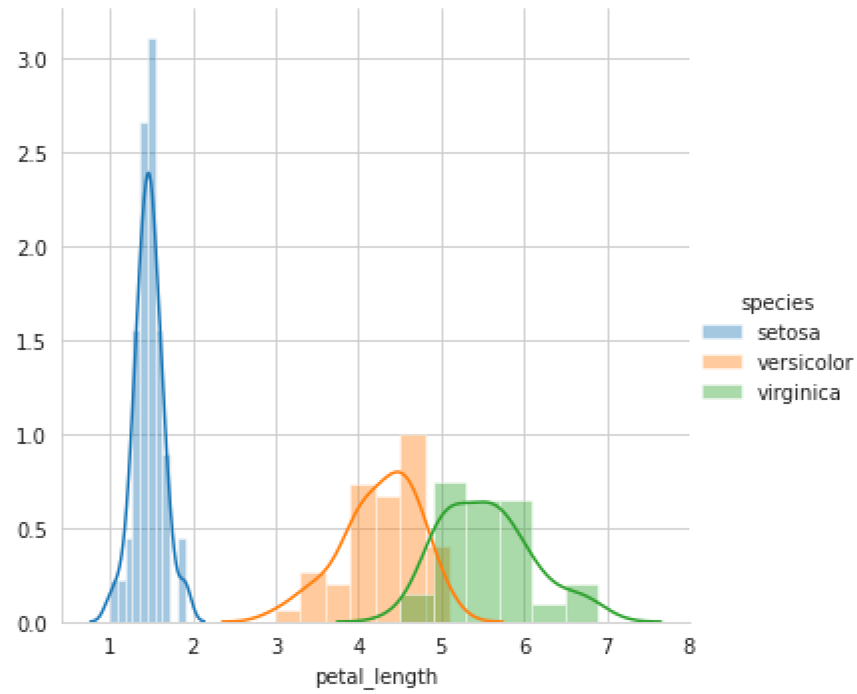
We can use PDF or histogram

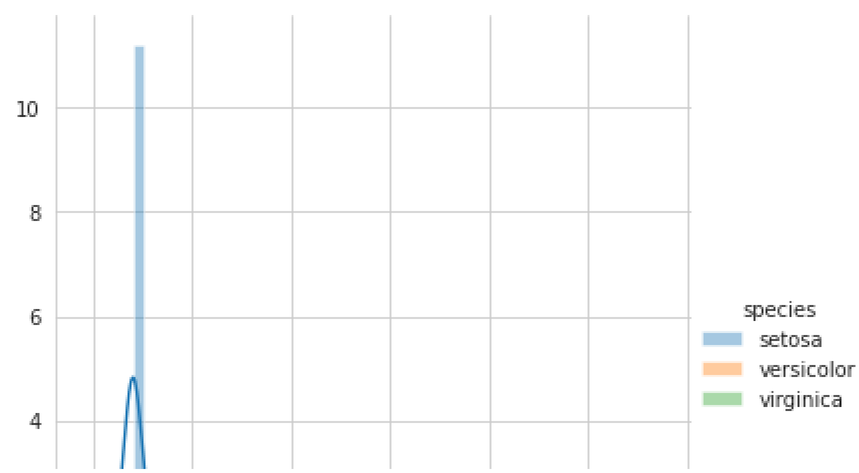
PDF tell the % of points available for a value for a category

It is 1D density plot

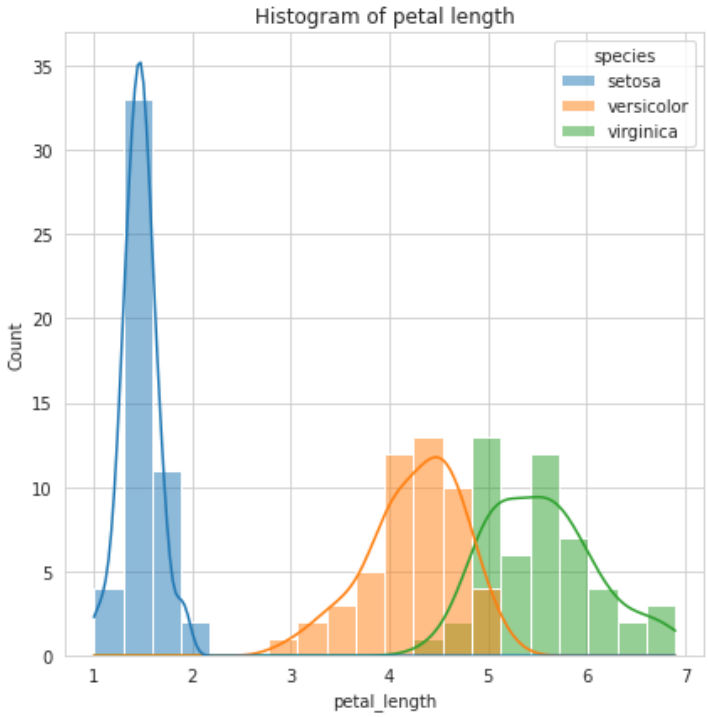
```
sns.FacetGrid(iris, hue='species', height=5) \
    .map(sns.distplot, 'petal_length') \
    .add_legend()
sns.FacetGrid(iris, hue='species', height=5) \
    .map(sns.distplot, 'petal_width') \
    .add_legend()
sns.FacetGrid(iris, hue='species', height=5) \
    .map(sns.distplot, 'sepal_length') \
    .add_legend()
sns.FacetGrid(iris, hue='species', height=5) \
    .map(sns.distplot, 'sepal_width') \
    .add_legend()
plt.show()

#pl > pw >> sl >> sw
#Petal length is better compared to others


#the above code creates 4 plots one below other.. So I did below one
```

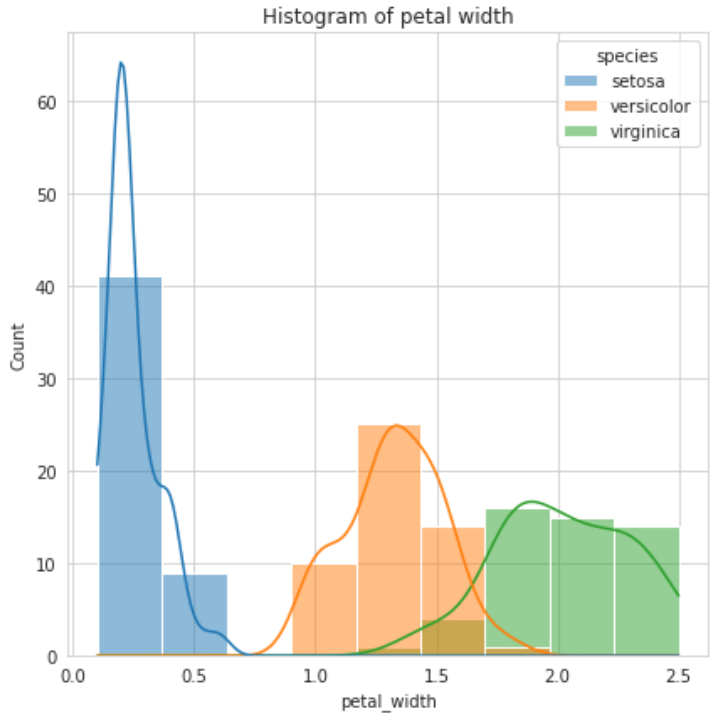
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `dis
  warnings.warn(msg, FutureWarning)
```

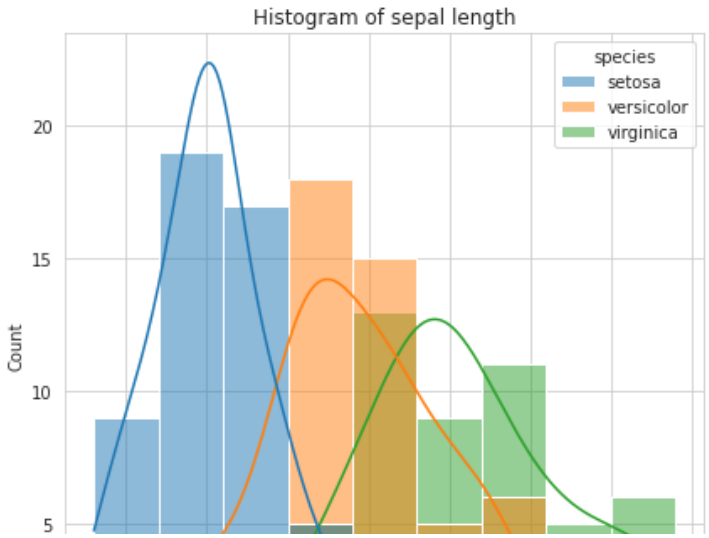


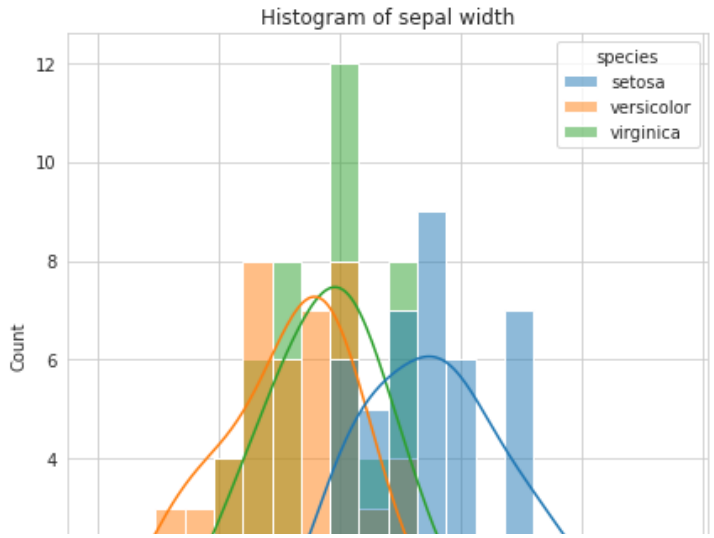


```
fg, axes = plt.subplots(2,2, figsize=(15,15))
sns.histplot(data=iris, x='petal_length', hue='species', bins=20, kde=True, ax=axes[0,0])
axes[0,0].set_title('Histogram of petal length')
sns.histplot(data=iris, x='petal_width', hue='species', kde=True, ax=axes[0,1])
axes[0,1].set_title('Histogram of petal width')
sns.histplot(data=iris, x='sepal_length', hue='species', kde=True, ax=axes[1,0])
axes[1,0].set_title('Histogram of sepal length')
sns.histplot(data=iris, x='sepal_width', hue='species', bins=20, kde=True, ax=axes[1,1])
axes[1,1].set_title('Histogram of sepal width')
plt.show()
```









Cumulative Distributive Function

CDF : there are **82%** of setosa flowers that have **petal length <= 1.6**

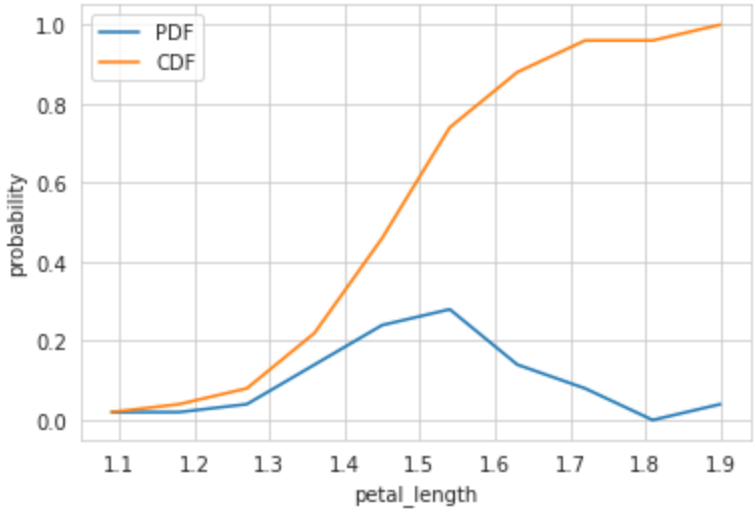
100% of setosa flowers have **Petal length < 1.9**

Differtiation of CDF = integration of PDF (calculus)

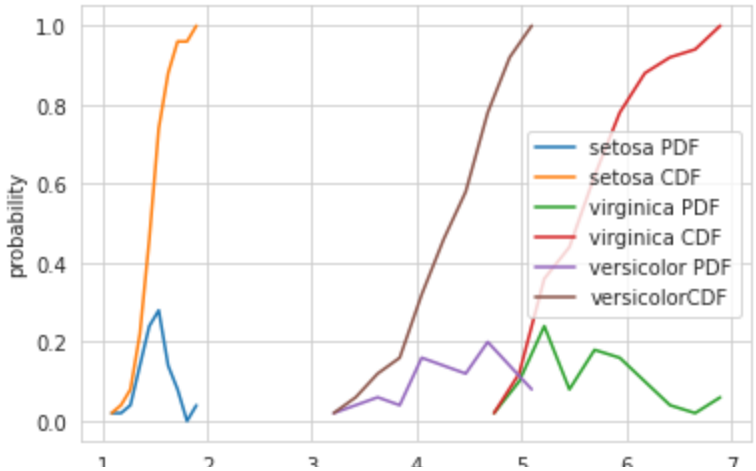
```
counts, bin_edges = np.histogram(setosa['petal_length'], bins=10, density=True)
print(counts,' ', bin_edges)
print(sum(counts))

pdf = counts/sum(counts)
print(pdf)

cdf = np.cumsum(pdf)
print(cdf)

plt.plot(bin_edges[1:], pdf, label="PDF")
```

```
plt.plot(bin_edges[1:], pdf, label="PDF")
plt.plot(bin_edges[1:], cdf, label="CDF")
plt.xlabel('petal_length')
plt.ylabel('probability')
plt.legend()
plt.show()
```

```
[0.22222222 0.22222222 0.44444444 1.55555556 2.66666667 3.11111111
 1.55555556 0.88888889 0.         0.44444444]   [1.   1.09 1.18 1.27 1.36 1.45 1.54 1.63
11.11111111111111
[0.02 0.02 0.04 0.14 0.24 0.28 0.14 0.08 0.   0.04]
[0.02 0.04 0.08 0.22 0.46 0.74 0.88 0.96 0.96 1.  ]
```




```
#for all species, cdf and pdf

count, bin_edges = np.histogram(setosa['petal_length'], bins = 10, density=True)
pdf = count/sum(count)
cdf = np.cumsum(pdf)
plt.plot(bin_edges[1:], pdf, label="setosa PDF")
plt.plot(bin_edges[1:], cdf, label="setosa CDF")


count, bin_edges = np.histogram(virginica['petal_length'], bins = 10, density=True)
pdf = count/sum(count)
cdf = np.cumsum(pdf)
plt.plot(bin_edges[1:], pdf, label="virginica PDF")
plt.plot(bin_edges[1:], cdf, label="virginica CDF")


count, bin_edges = np.histogram(versicolor['petal_length'], bins = 10, density=True)
pdf = count/sum(count)
cdf = np.cumsum(pdf)
plt.plot(bin_edges[1:], pdf, label="versicolor PDF")
plt.plot(bin_edges[1:], cdf, label="versicolorCDF")

plt.xlabel('petal_length')
plt.ylabel('probability')
plt.legend()
plt.show()
```



Based on the plot, we can say that,

1. 100% of all setosa flowers will have petal_length <= 1.9
2. 95% of all versicolor flowers will have petal_length <= 5

Mean, Standard Deviation, variance

Why "Mean"?

It tells about the central tendancy. Eg : Average behaviour of the flowers

Setosa's average petal length is much smaller than virginica, versicolor. But virginica & versicolor's petal length are much closer.

What if somebody entered a wrong value as '50' for setosa or due to data corruption? This is an **outlier point**. Now the **mean** becomes 2.451 . This is the **disadvantage** of mean.

Spread/standard deviation is the range of values where the most of the points lie. Eg : If the mean is 1.4 and spread is 0.5, then **from 0.9 to 1.9**, most of the points lie. The thinner the spread, the useful the data points are. *Setosa's petal length's spread is thinner than virginica & versicolor's petal length*. It doesn't mean that there are no points outside the range. That spread is called **standard deviation** which says how far are my points from the mean. Square of **std** is the **variation**

$$variance = 1/n \sum_{i=1}^{n} (x_i - \mu)^2$$

Just by looking into mean & std, we can assume how the data may look like and what they try to convey. But **outlier** can damage the result.

```
mean_s_pl = setosa['petal_length'].mean()
mean_vi_pl = virginica['petal_length'].mean()
mean_ve_pl = versicolor['petal_length'].mean()

print("Setosa mean : ", mean_s_pl)
print("Virginica mean : ", mean_vi_pl)
print("Versicolor mean : ", mean_ve_pl)

print("\nNow with outlier")
```

```
mean_s_pl_outlier = np.mean(np.append(setosa['petal_length'], 50))
print("Setosa mean : ", mean_s_pl_outlier)
```

```
Setosa mean :  1.464
Virginica mean :  5.552
Versicolor mean :  4.26

Now with outlier
Setosa mean :  2.4156862745098038
```

```
std_s_pl = setosa['petal_length'].std()
std_vi_pl = virginica['petal_length'].std()
std_ve_pl = versicolor['petal_length'].std()
print("Setosa std : ", std_s_pl)
print("Virginica std : ", std_vi_pl)
print("Versicolor std : ", std_ve_pl)
```

```
Setosa std :  0.1735111594364455
Virginica std :  0.5518946956639835
Versicolor std :  0.46991097723995806
```

```
#let's check how many points are in std range for setosa
s_in_std_range = setosa[(setosa["petal_length"] >= (mean_s_pl-std_s_pl)) & (setosa["petal_leng
vi_in_std_range = virginica[(virginica["petal_length"] >= (mean_vi_pl-std_vi_pl)) & (virginica

print("% of setosa in std's range : ", 100*(s_in_std_range/len(setosa["petal_length"])), '%' )
print("% of virginica in std's range : ", 100*(vi_in_std_range/len(virginica["petal_length"]))
```

```
% of setosa in std's range :  80.0 %
% of virginica in std's range :  70.0 %
```

Median

Median of setosa's Petal length with/with_out outlier is ***same***. This is what we want. All medians tend to look like **mean**.

But what is median?
It is **middle value** from the list after sorting (any order). if 'n' is odd, middle elt is $(n+1)/2$, if 'n' is even, middle elt is average of middle 2 elements $avg(n/2, (n+1)/2)$

Why Median is not affected by outliers? Because, outliers exist in the extreme. What if there are many outliers (i.e) half of the values are corrupted? Only at this time, median will get corrupted.

```
median_s_pl = np.median(setosa['petal_length'])
median_vi_pl = np.median(virginica['petal_length'])
median_ve_pl = np.median(versicolor['petal_length'])
print("Setosa Median : ", median_s_pl)
print("Virginica Median : ", median_vi_pl)
print("Versicolor Median : ", median_ve_pl)

#adding an outlier
```

```
median_s_pl_outlier = np.median(np.append(setosa['petal_length'], 50))
print("\nSetosa median (after outlier) : ", median_s_pl_outlier)
```

```
Setosa Median :  1.5
Virginica Median :  5.55
Versicolor Median :  4.35

Setosa median (after outlier) :  1.5
```

Percentiles & Quartiles

Percentile is where do you lie in the sorted list. Value in the middle of a list is the **50th percentile** of the list. It is same as median of the list.

What is 10th percentile? In the sorted array, pick the 10th value in array. It says that about 10% of points in the list are less than this value and 90% of points are greater than this value.

25, 50, 75 percentiles are called quartiles.

```
reqd_percentiles = np.arange(0,100,25) #0, 25, 50, 75
quartiles_s = np.percentile(setosa['petal_length'], reqd_percentiles)
quartiles_vi = np.percentile(virginica['petal_length'], reqd_percentiles)
quartiles_ve = np.percentile(versicolor['petal_length'], reqd_percentiles)

print("percentiles of setosa in ",reqd_percentiles," are ", quartiles_s)
print("percentiles of virginica in ",reqd_percentiles," are ", quartiles_vi)
print("percentiles of versicolor in ",reqd_percentiles," are ", quartiles_ve)

#get a particular percentile value
print("95th percentile of versicolor is ", np.percentile(versicolor['petal_length'], 95))
#it is evident from the CDF and PDF. We concluded that about 95% of values are less than 5
```

```
percentiles of setosa in  [ 0 25 50 75]  are  [1.    1.4   1.5   1.575]
percentiles of virginica in  [ 0 25 50 75]  are  [4.5   5.1   5.55  5.875]
percentiles of versicolor in  [ 0 25 50 75]  are  [3.   4.   4.35 4.6 ]
95th percentile of versicolor is  4.9
```

IQR(Inter Quartile Range) and MAD(Median Absolute Deviation)

Median Absolute Deviation :
$Median(\ |x_i - \tilde{x}|\ )$ where $\tilde{x}$ is the median of the input array

Taking mean will cause problems due to outlier points. So we are taking medians of the absolute values of distance of median from the point. It is similar to **standard deviation**. But since **std** will be affected by outlier points, we can make use of **MAD**

Inter Quartile Range :
It is $75^{th}\ percentile\ value - 25^{th}\ percentile\ value$
In this short range, 50% of the points lie.

```
mad_s = robust.mad(setosa['petal_length'])
mad_vi = robust.mad(virginica['petal_length'])
mad_ve = robust.mad(versicolor['petal_length'])

print("MAD of Setosa : ", mad_s)
print("MAD of versicolor : ", mad_ve)
print("MAD of virginica : ", mad_vi)
```

```
MAD of Setosa :  0.14826022185056031
MAD of versicolor :  0.5189107764769602
MAD of virginica :  0.6671709983275211
```

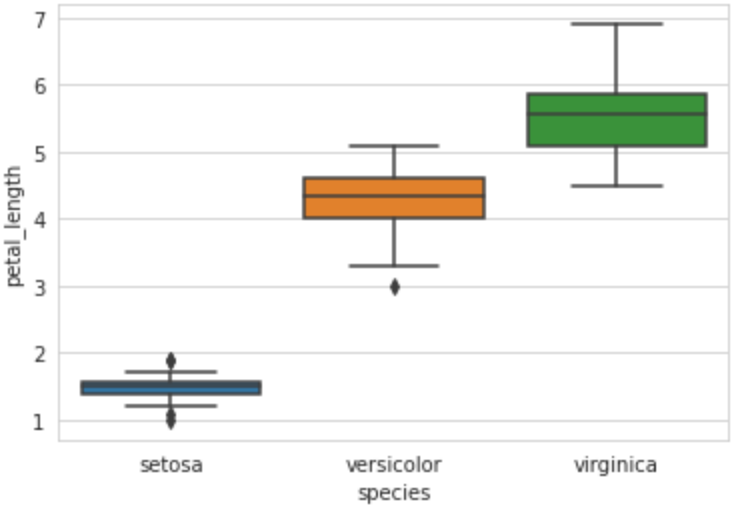
Box Plot & Whikers

In histogram, we don't understand where the 25^{th}, 50^{th} and 75^{th} values lie. We can get those values in PDF & CDF plots, but we have to map the y values to x values. We can use box plots.

Box Lines in the box are 25, 50 and 75 percentile values. We can see how values span and see how to write if else condition in code to classify the classes.

Whiskers are max & min value. But **sns** uses some complex mechanism like **1.5 x IQR** for the whiskers

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

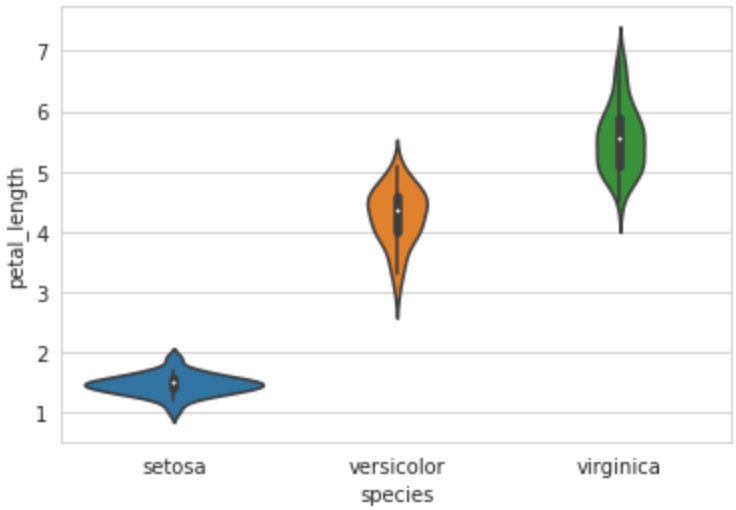


Violin Plots

It's a combination of **histogram** & **box plots**.

Denser values are fatter and sparse values are thin.

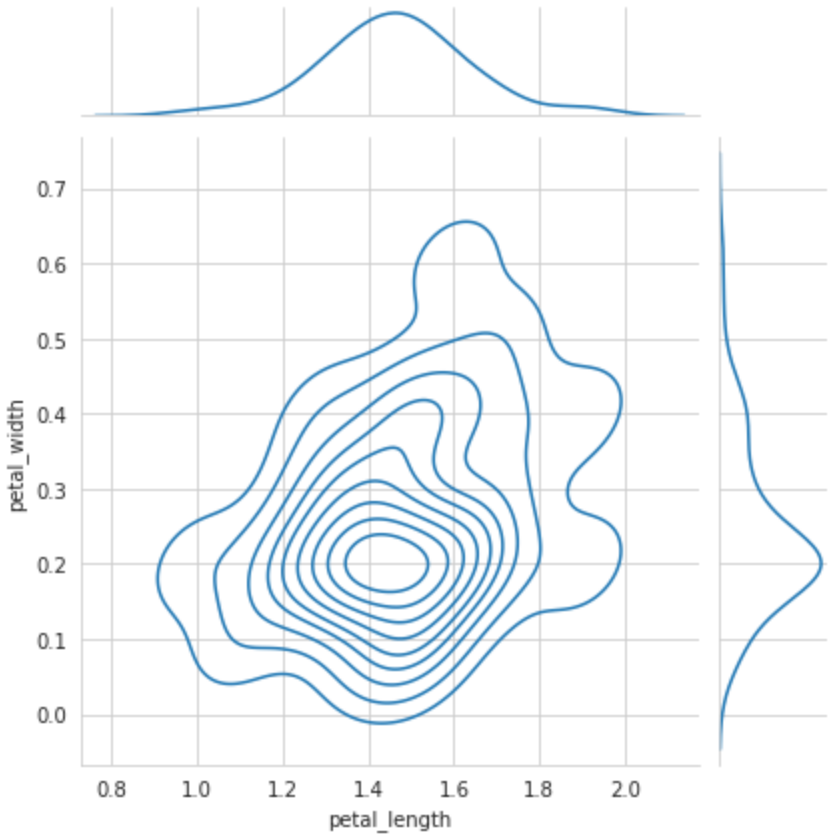
```
sns.violinplot(data=iris, x='species', y='petal_length', size=8)
plt.show()
```



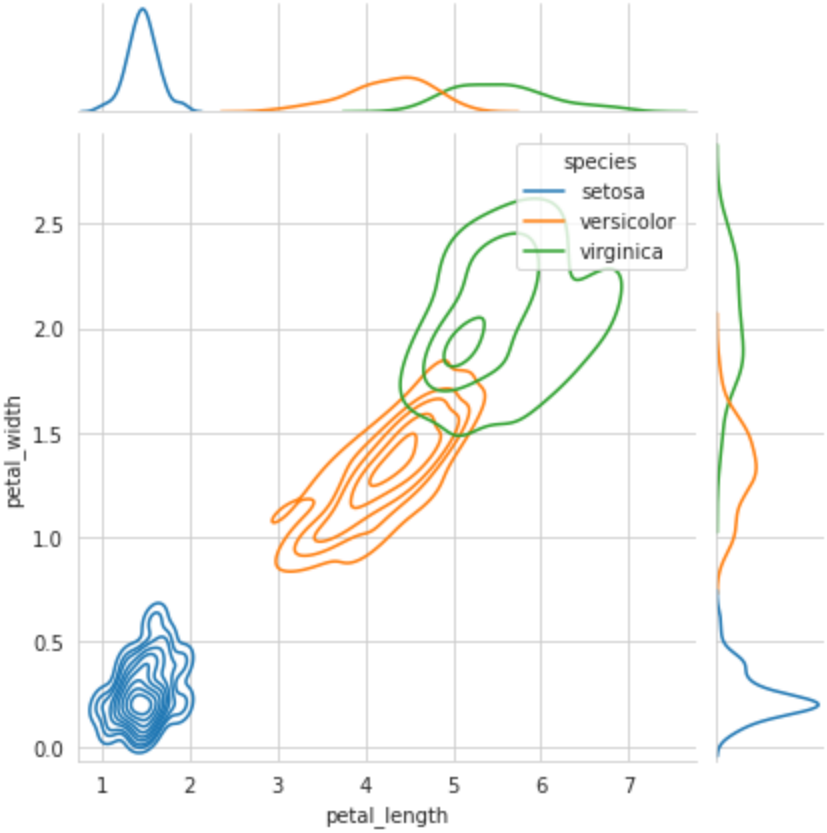
Multivariate probability density, contour plot.

It is 2D density plot

```
sns.jointplot(x="petal_length", y="petal_width", data=setosa, kind="kde");
plt.show();
```




```
sns.jointplot(x="petal_length", y="petal_width", data=iris, kind="kde", hue='species', );
plt.show();
```



Summary

Explain your findings/conclusions in plain english Never forget your objective (the probelm you are solving) .

Perform all of your EDA aligned with your objectives.

```
iris_virginica_SW = virginica.iloc[:,1]
iris_versicolor_SW = versicolor.iloc[:,1]
from scipy import stats
print(stats.ks_2samp(iris_virginica_SW, iris_versicolor_SW))
x = stats.norm.rvs(loc=0.2, size=10)
print(stats.kstest(x,'norm'))
x = stats.norm.rvs(loc=0.2, size=100)
print(stats.kstest(x,'norm'))
x = stats.norm.rvs(loc=0.2, size=1000)
print(stats.kstest(x,'norm'))
```

```
Ks_2sampResult(statistic=0.26, pvalue=0.06779471096995852)
KstestResult(statistic=0.3532623033220886, pvalue=0.12743969438828925)
KstestResult(statistic=0.12373155119872403, pvalue=0.0858476177060033)
KstestResult(statistic=0.07686100470629365, pvalue=1.3862298520313892e-05)
```