When there is data every where, then we need something to visualize that data and there comes the need of this topic called data visualization.

To clean the data we follow a process called Dimensionality Reduction. we have certain algorithms to do this task:—

1. PCA
2. TSNE
3. LDA

etc. } They help us to convert large amount of data into a 2-D or 3D data in order to visualize them

This part of data visualization is a predominant one, in initial EDA

Exploratory Data Analysis (EDA)
process of getting data

→ there are various packages in python for data visualization, but the most prominent is MatPlotLib.

→ we also use Seaborn, which is also built on top of support of MatPlotLib

⊛ Matplotlib Architecture:

There are `3' diff layers in this :-

1. Backend layer
2. Artist layer
3. Scripting Layer

Backend layer:

what is Backend layer?

This is the bottom most layer of a figure which contains implementation of several functions that are required for plotting.

3 classes of Backend layer:

→ FigureCanvas

the layer / surface on which the figure will be drawn

→ Renderer

the classification that takes care of the drawing on the surface.

→ Event

to handle the keyboard and mouse events

Artist layer:-

note: Compared to Artist layer, we dont work much with the Backend layer.

→ This is the second layer / middle

eg:- paper is the Figure Canvas and
→ sketch pen as renderer

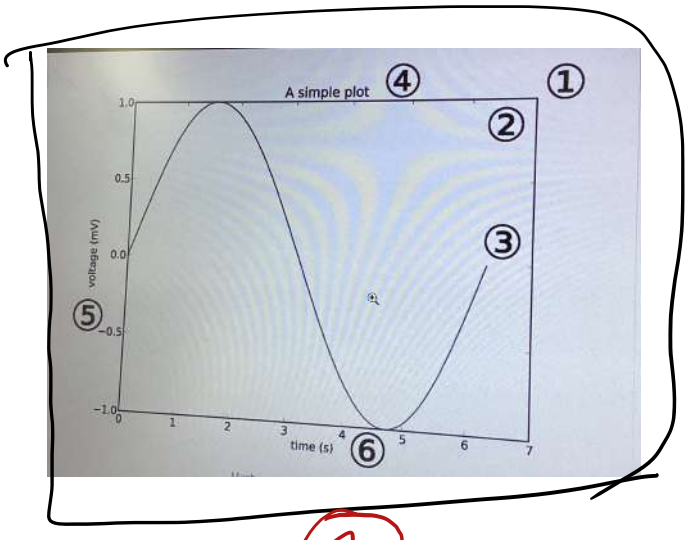
→ Then the hand of the painter is the Artist layer.

which has certain functions, it knows how to sketch to get the exact figure.

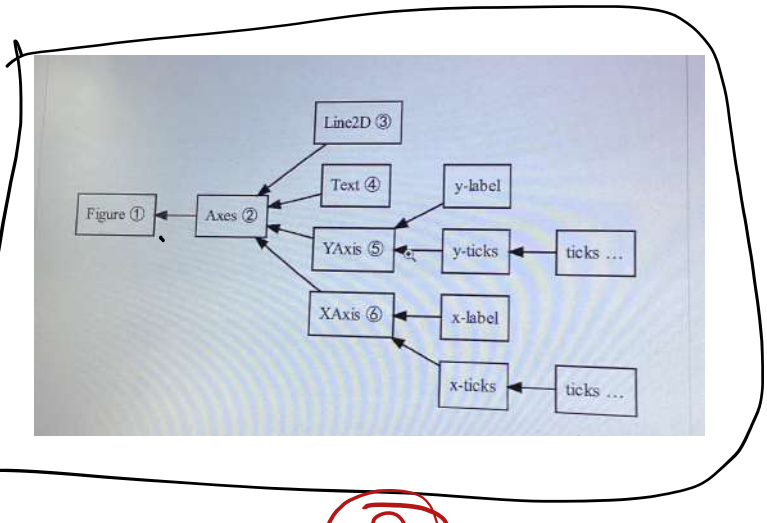
→ Just like in Backend layer we have 3 classes. Even, in Artist layer we have few classes like :—

1. Figure
2. Axes
3. Axis

→ Figure is the topmost one
→ And we can add multiple no. of axes to a figure, upon which the plot is done.



1



2

Scripting Layer:

→ This is the top most layer on which majority of our codes will stay around.

→ For day to day works we almost rely on this scripting layer for matplotlib.

→ The methods in this layer, almost automatically takes care of layers and all we need to care about is the current state (figure & subplot).

Hence, AKA | Stateful interface.

Common Terms:-

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Axes: (class that has attributes)

→ it is the entire area of a single plot in the figure.

→ it contains:

1. adding title
2. giving labels

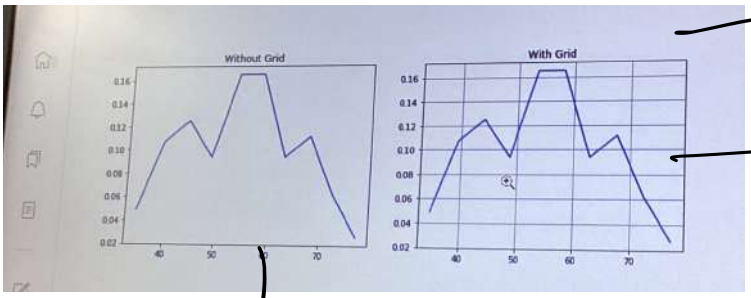
③ selecting the values for diff types of plots on each axes. etc..

→ we can have multiple axes in a single plot, by which we can combine multiple plots into a single figure

Grid:

→ when a grid is enabled in a plt we can see horizontal and vertical lines at the background.

→ This can be enabled plt.grid()
→ used for having a rough estimation.



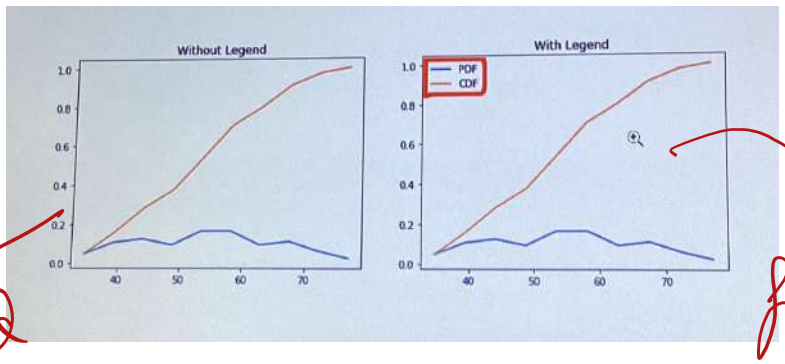
→ ex:- grid → with
→ without

Legend:

→ it is a labeled list of representation of diff diff plots which are present in a figure (on that

particular axis)

→ mostly helpful when we have diff plots in a particular figure



in this figure, legend is not there

in this figure, we have a legend (PDF) its helpful to view diff plots easily.

Subplots:

→ when we want '2' or more plots in a single image, we can make use of subplot

→ `plt.subplot(xyz)`

xyz is a '3' digit integer

x = no. of rows

y = no of columns

~~2~~ = index number of that plot

eg:-

plt.subplot(131)

- 1 → row
- 3 → columns
- 1 → index value

note: In addition to subplot you can also use gridspec

Title:

→ we can set a title to a plot using

plt.title()

→ Xlabel: to set the label for x axis

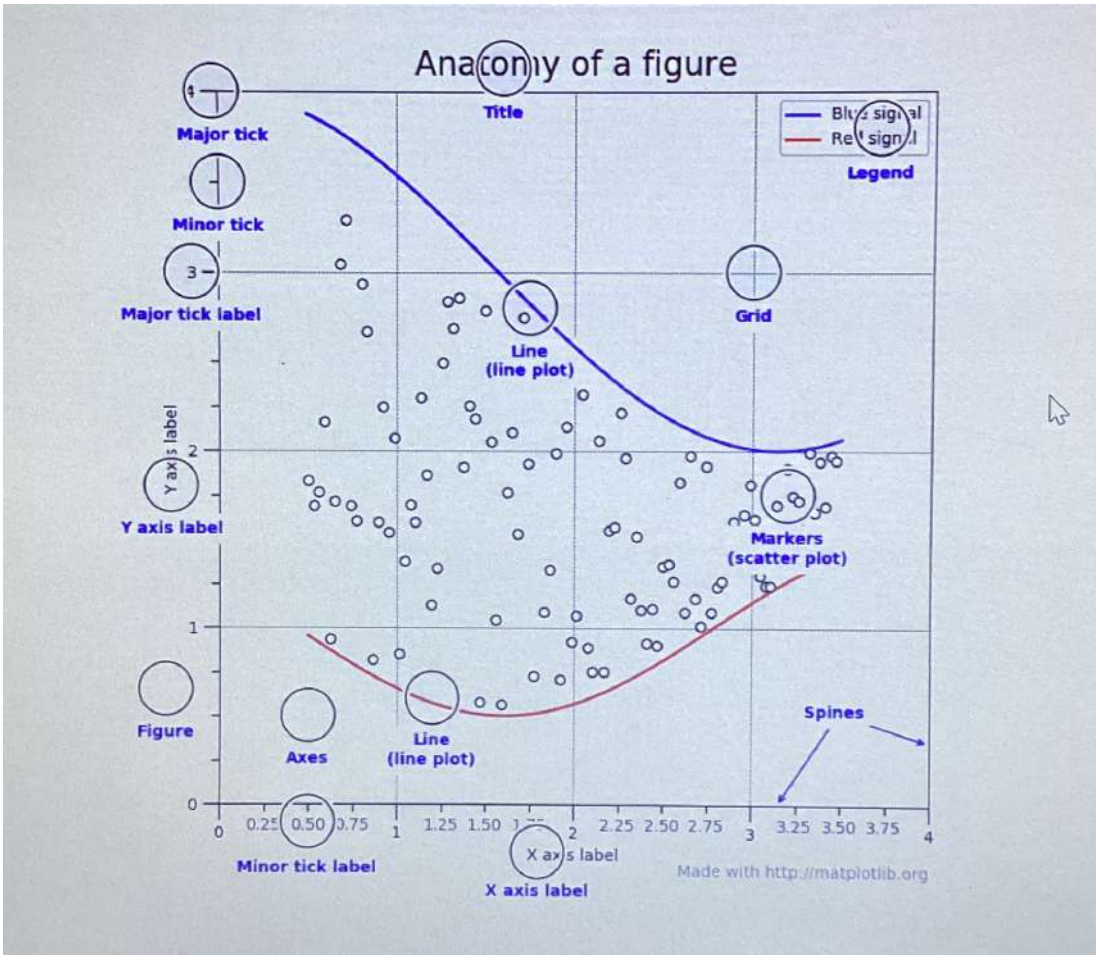
plt.xlabel()

→ Ylabel: to set the label for y axis.

plt.ylabel()

For a broad view, you can refer the below

pic:-



Diff types of analysis:-

There are `3' types of analysis

(1) Univariate: In univariate analysis we will be using a single feature to analyze almost of its properties.

(2) Bivariate: Comparing exactly b/w `2' features

③ Multivariate: Comparing more than 2 Variables

We will cover the below plots:

1. scatter plot (B)
2. pair plot (M)
3. Box plot (U)
4. Violin plot (U)
5. Distribution plot (U)
6. Join plot (U & B)
7. Bar chat (B)
8. Line plot (B)

Scatter plot :-

→ one of the most commonly used plot for simple data visualization.

→ available in 2D and 3D also

→ 2D here is used

find patterns
clusters
seperability of the data.

1st x

plt.scatter(x,y)

eg:- on the iris dataset we perform the scatterplot()

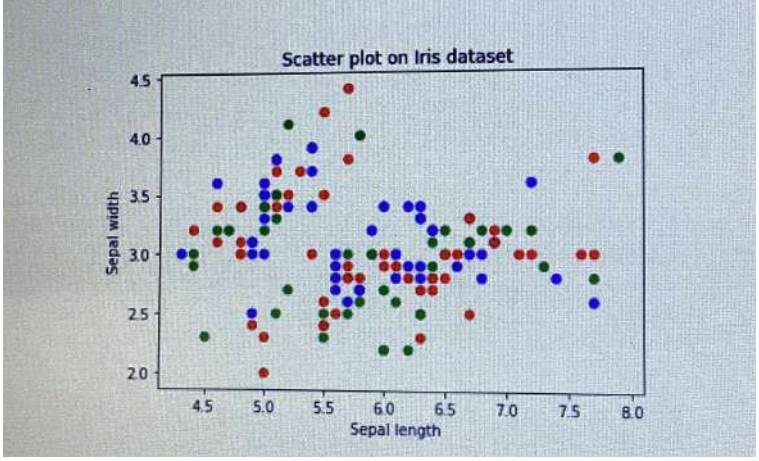
```
plt.scatter(iris['sepal_length'], iris['sepal_width'])
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.title('scatter plot on Iris dataset')
```

This gives the o/p having plotted the respective [x] and [y] values.

let's try to change the colour of the plot.

```
plt.scatter(iris['sepal length'], iris['sepal width'],
            colour = ['r', 'b', 'g'])
plt.xlabel('sepal length')
plt.ylabel('sepal width')
plt.title('scatter plot on Iris dataset')
```

→ Colors are assigned to points on basis of how they were present in dataset.



→ if we want colors acc to the class
→ in this case, we have '3' class label

→ This is where seaborn pops up, with its visualisation stuff.

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

(1) Using seaborn, we are setting the grid upon which we plot as a "whitegrid"

(2) The parameter hue in the facetGrid decides the colour of each datapoints.

(3) we used "Species" column in "hue", so that we got this plot colored in this manner.

Note: This makes Seaborn a bit more superior than matplotlib when it comes to visualization.

⇒ for 3d scatter plts we can use plot.ly

Pair plot :-

→ we can Scatter plot for 2D, 3D but for 4-D we have to use pairplot

→ if we have 'n' features, pair plot will create us (n x n) figure

↙

diagonal plots will be histogram plots of the feature coresponding to that row

↙

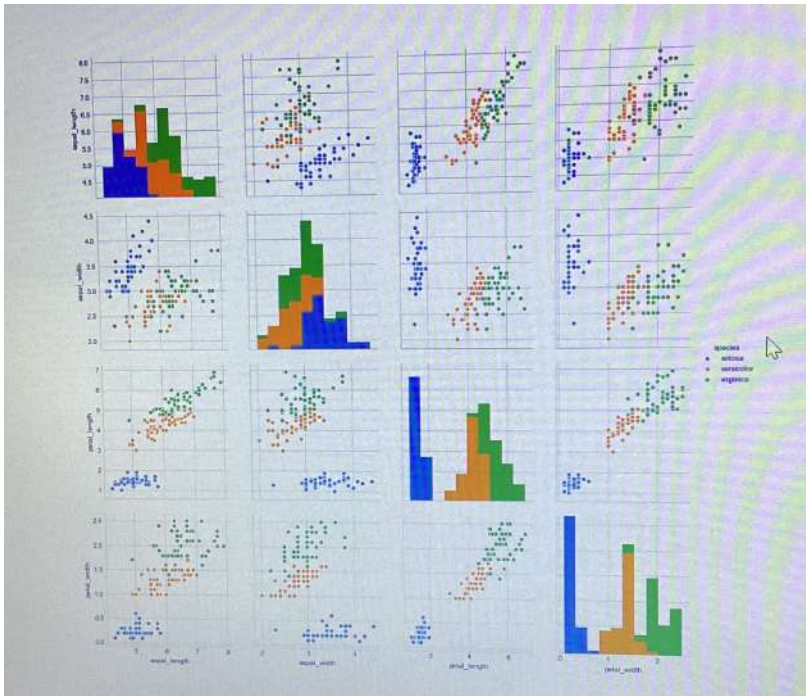
Rest of plots are the :-

(1.) Combination of feature from

each row in y-axis

② feature from each column in X-axis

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



→ By performing pairplot we can observe which two features can well explain the data

→ and then we can use scatter plot b/w those 2 features to explore further.

note: since, we will get nxn plots for n features, pairplot may become complex when we have more no. of feature.

⇒ In such case we use Dimensionality Reduction (DR) to map data into 2D plane and visualize it using our

Very own 2d scatter plot

Box plots :-

→ It can used mostly to obtain more of the statistical details about the data.

→ The st. lines at the maximum and minimum are also called as Whiskers

→ points outside of the whiskers are considered to be outliers

→ Boxplot is available in seaborn library.

→ This plot actually gives us the represtn in the form of Quartiles.

- 25^{th}
- 50^{th}
- 75^{th}

(findout in Google)

IQR - Inter Quartile Range

↳ Using boxplot we can see this also !!!

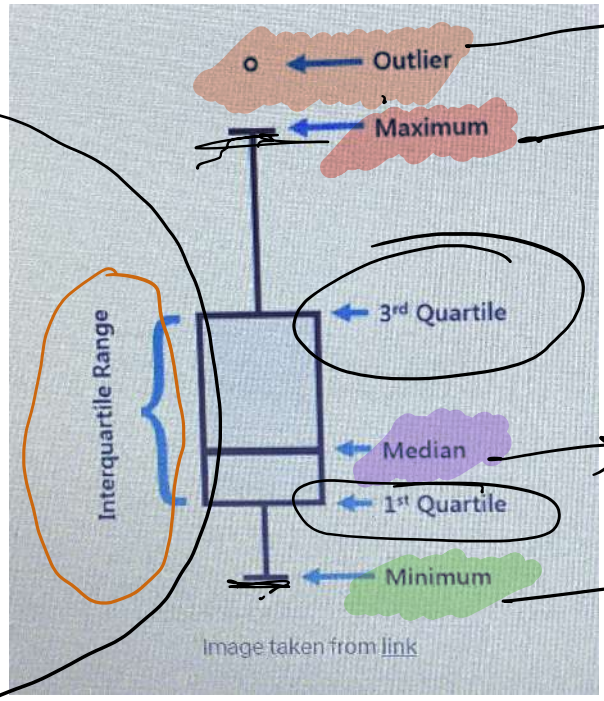
Code: x → Variable to be predicted

y → independent feature

note: because, this uses only 1 indep variable, we call it a Univariate

⇒ Using the box plot, we can see how much of data is present in 1st Quartile and how much as Outliers.

note: we use these plots to find out outliers and to clean them before sending them to a training model. As they impact the ML procell.

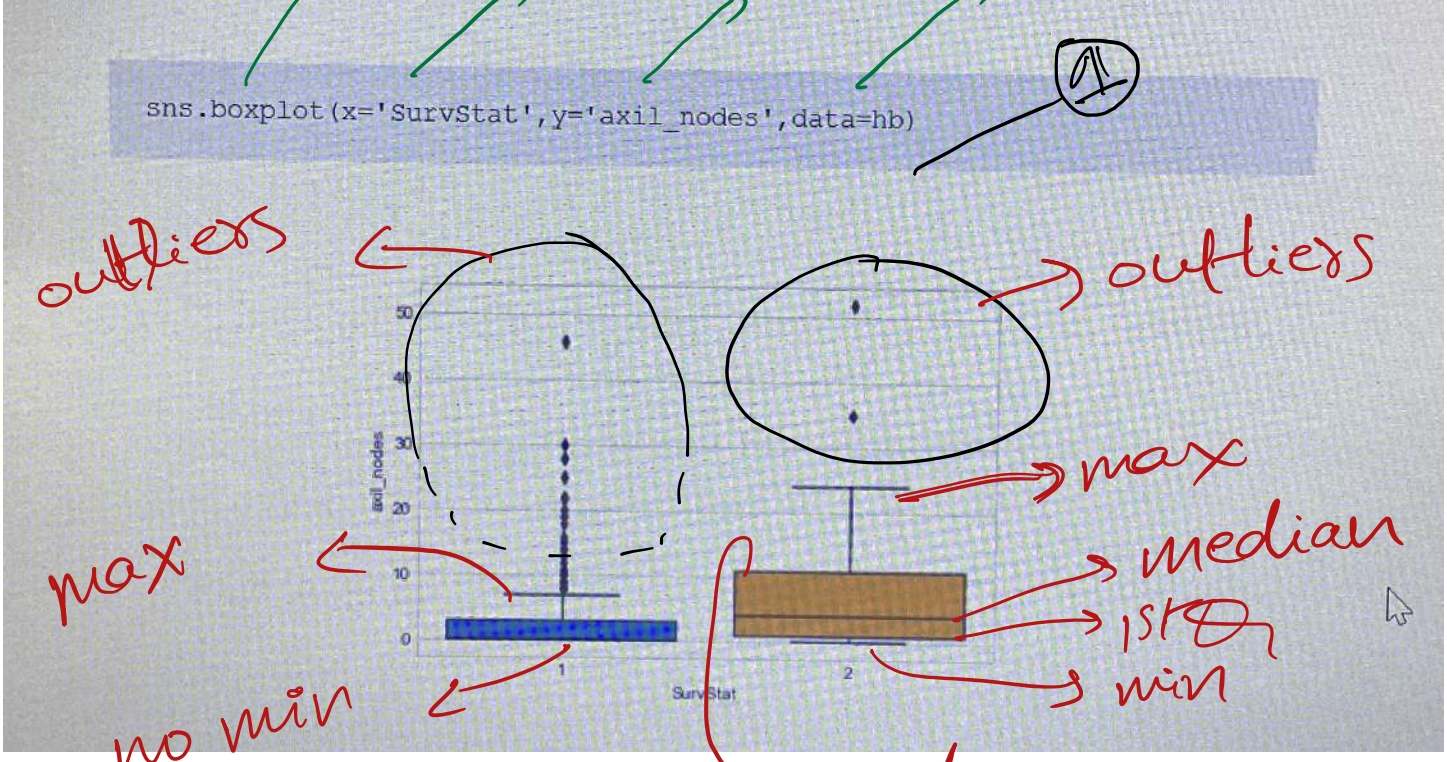


outliers

counted by 2

func → on X axis → on Y axis → the dataframe we use

```
sns.boxplot(x='SurvStat',y='axil_nodes',data=hb)
```



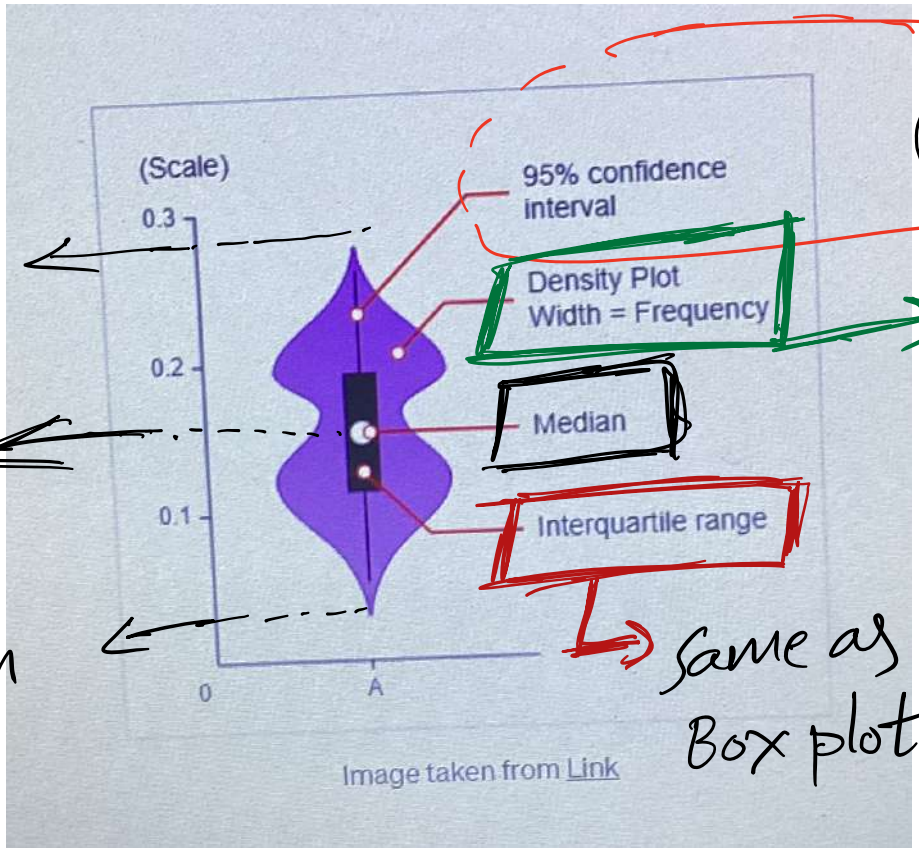
Violin plots :-

→ The violin plots can be inferred as a combination of Box plot at the middle and distribution plots [Kernel Density Estimation]

→ This can give us the details of distribution like wether the distribution is multimodal, skewness etc..

→ gives us 95% confidence interval.

→ This plot is also from seaborn package.



max

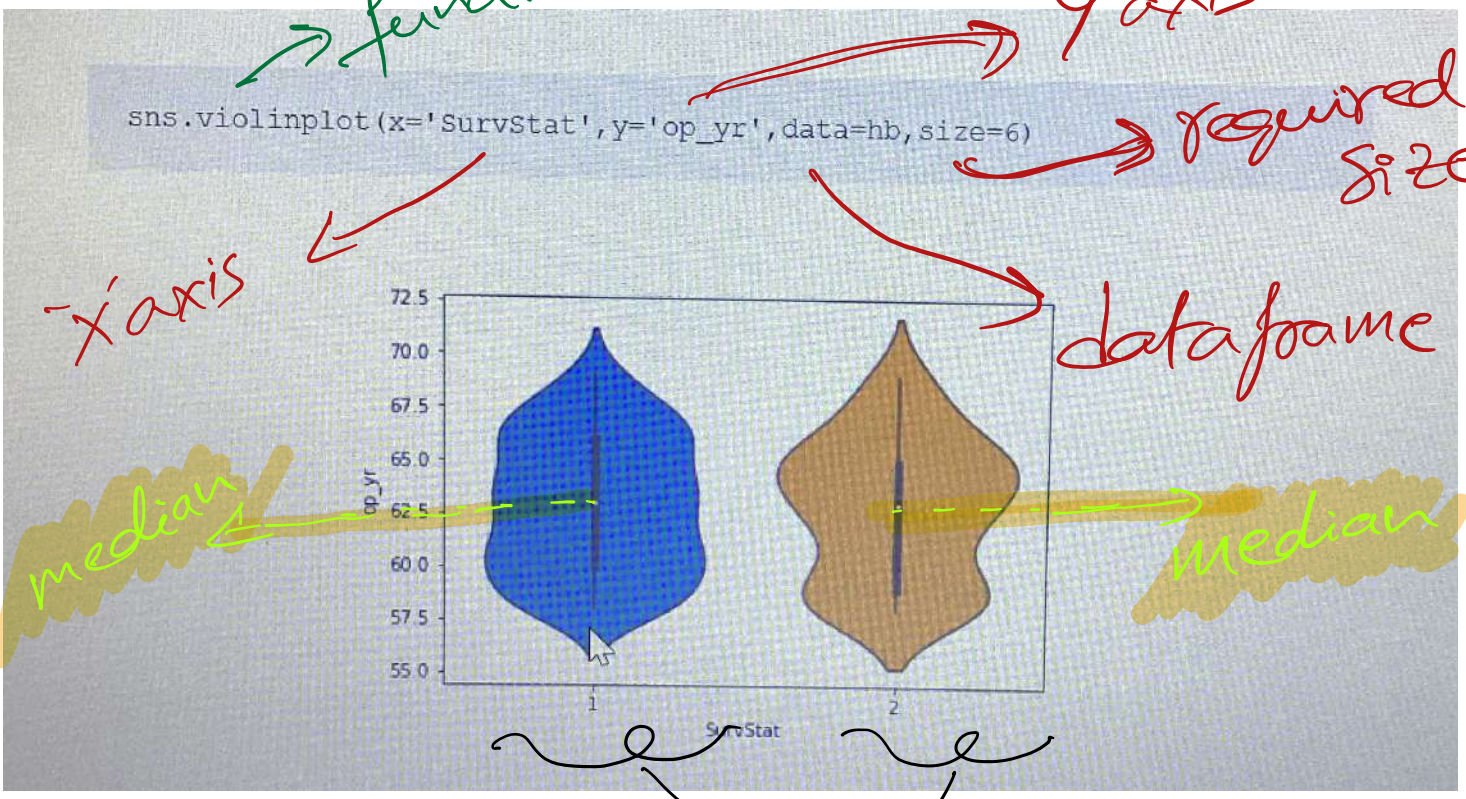
max

median value

min

More / less denser

Same as Box plot



function

'y' axis

required size

'x' axis

dataframe

median

median

density plot width

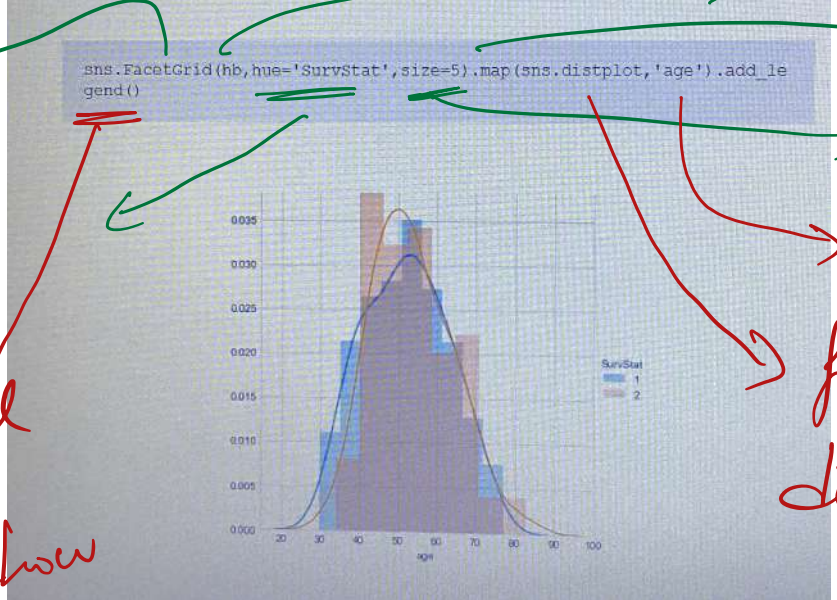
Distribution plot :-

→ one of the best univariate plot
→ we use this plot when we have only one variable (most of the times)
→ This plot gives us a combination of

PDF + Histogram in a single feature

→ The sharp block like structures are called histograms and the smooth curve is called **Probability Distribution Function (PDF)**

→ The pdf of a curve can help us to identify the underlying distribution of that feature which is one major takeaway from data visualization / EDA



clarity of many graphs

Join Plot:

→ In a single figure we can do both single variate and bivariate analysis.

→ The main plot will give us a bivariate analysis,

→ whereas on the top and right side we will get univariate plots of both the variables that were considered.

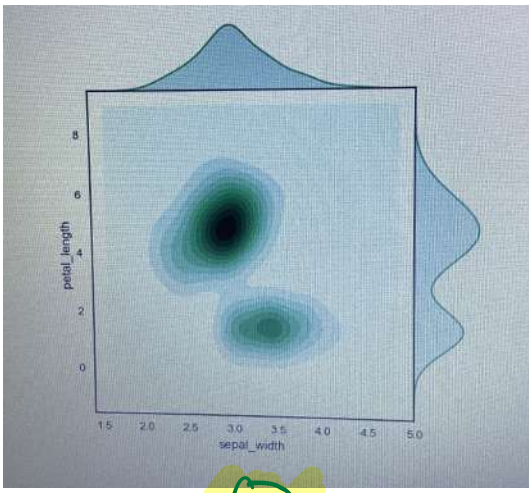
→ There is Kind parameter in Seaborn's join plot function.

→ The plot in the figure below is KDE (Kernel Distribution Estimation)

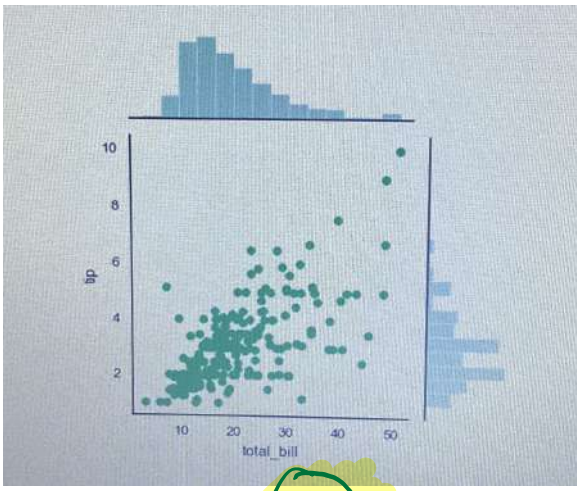
→ More points, it will be dark and if less points it will be in light color

note: The '2' most important plots we use will be scatter plot for bivariate and

distribution plot for univariate and since we are getting both in a single plot as shown below, it will make our work more easier.



(1)



(2)

Bar chart :-

→ used to analyse data like sales figure every week, revenue from a product, no. of visitors to a site on each day of a week etc - - -

→ X = Position of bars
Y = length / height

plt.bar(x,y)

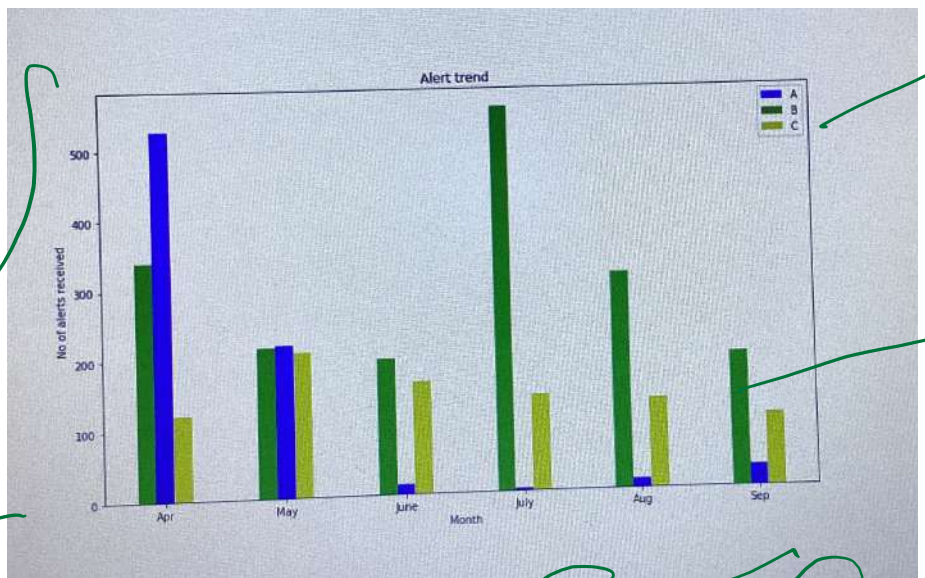
→ Ex:-

```
a=np.arange(6)
w=0.15

fig,ax=plt.subplots(figsize=(12,7),edgecolor='k')
p1=ax.bar(a,d,w,color='b')
p2=ax.bar(a-w,c,w,color='g')
p3=ax.bar(a+w,e,w,color='y')
ax.set_xticks(a)
ax.set_xticklabels(('Apr','May','June','July','Aug','Sep'))
ax.set_title('Alert trend')
ax.legend((p1[0],p2[0],p3[0]),('A','B','C'))
plt.xlabel('Month')
plt.ylabel('No of alerts received')

#plt.grid()

plt.show()
```



1. Using numpy, we generate '6' nums
2. we declare width (w) as some value
3. declaring subplots
4. for every dataset assigning proper colour
5. assigning x-axis thing like:-

① X ticks
② X tick labels
③ title

⑥ giving a legend
⑦ declaring what will be on (x) and (y) labels
⑧ grid for figure
⑨ show the plot

Line plots:

→ You can see this plot in every nook and corner of any sort of analysis b/w '2' variables.

→ graph will be connected by st. lines

→ we call plot multiple lines inside a single frame as show below where you need to add multiple plt.plot() commands with each line representing a diff color parameter

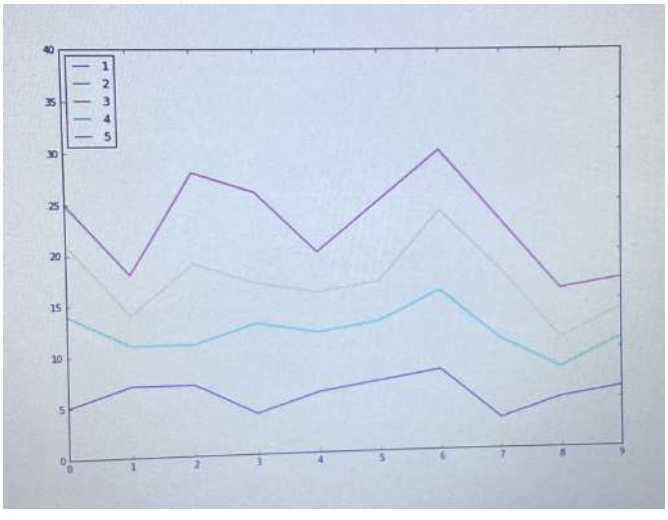
plt.plot(x,y)

→ These are used right from performing

distribution comparision using

QQplots till CV tuning elbow method

and analysing performance of a model using AUC curve



Note:

⇒ So far we saw some widely employed methods that are used to extract useful information / insight from the data.

⇒ we can also use some visualization tools to convey our find info / inference in form of plots to the audience.

HeatMap: (to visualize correlation pattern)

→ The heat map from seaborn library will create

a grid like plot along with an optional color bar.

→ We provide a 2D input matrix with certain values on each element to the heatmap and it exactly reproduces the output plot in the same shape as that of input matrix

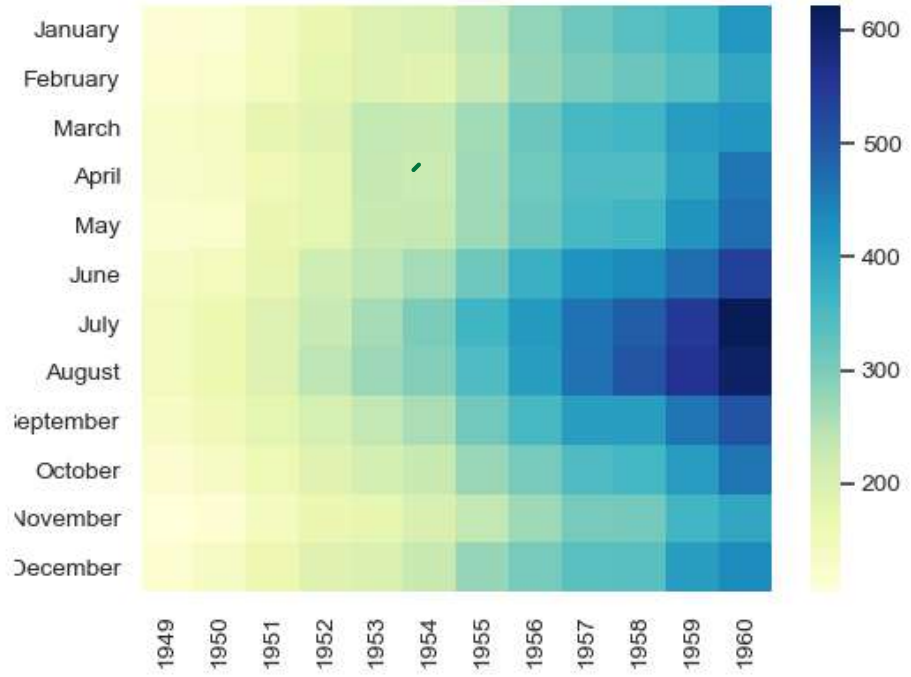
→ And each tile are colored based on the values provided in each elements of matrix to their corresponding tile.

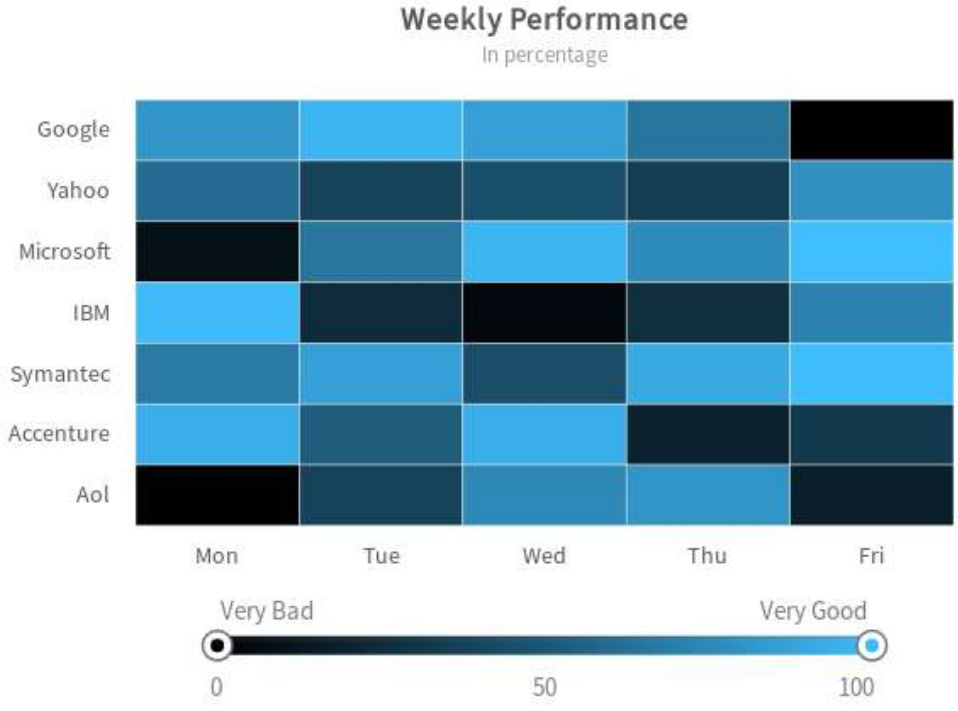
→ This can be much useful in cases where there will be a bigger matrix and we want to find which value is higher, lower and so on..... by simply looking at the diff colors toned used.

eg: weekly performance chart of different companies can be plotted using heatmaps.

note: In ML Applications, it can be used in representing confusion matrix of a model, used in hyperparameter tuning to plot

error values b/w '2' diff hyperparameters ek....





Word cloud:

→ It is nothing but creating an image

that containg all the words in a passage/ string with diff size, as per the frequency of occurence in that passage.

→ The word will appear bigger and bolder if it occurs more no. of times in a given passage.

note! (use)

⇒ we can conclude like what's discussed in a paragraph without reading it much.

⇒ The most repeating word will appear bold and bigger than the least occuring one.

It can be used a lot with ~~text~~ data analysis like, finding the common topics in a cluster while performing sentiment analysis and SEO optimizations etc...

below is the representation of "Word cloud"

Graphviz :

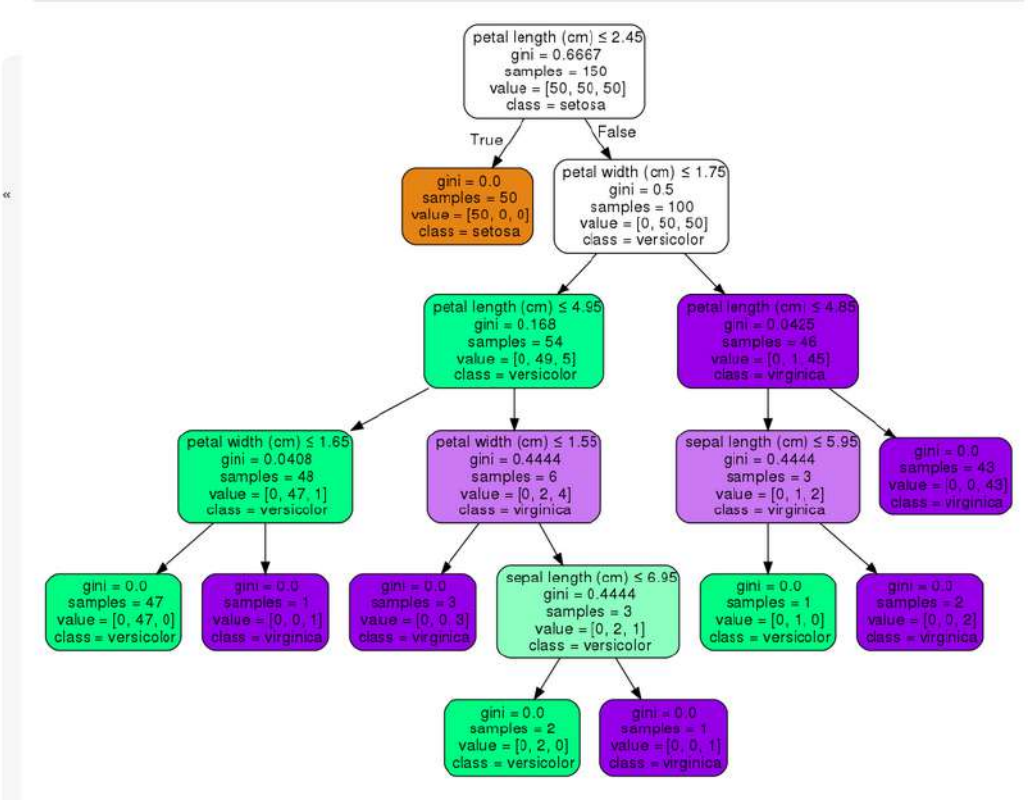
→ The Decision tree algorithms are one of the popular non-linear model.

→ If you want to view a linear model like linear regression you can do it simply using matplotlib / seaborn

→ whereas to visualize trees, we use a special tool called Graphviz

→ The Sklearn has a good implementation of a function called export graphviz which can help us to convert the decision tree model into dot format.

which is supported by graphviz tool.



Dendograms:

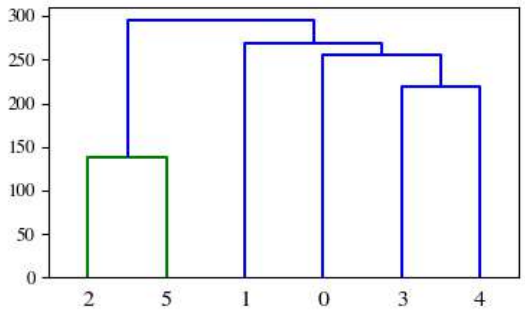
→ This is the method used to visualize the clusters formed when using the hierarchial clustering technique like :

Agglomerative clustering

→ The below figure is an example for dendrogram. Here each chain/link b/w

points, means they fall under same cluster.

Note: The dendogram structure may be too much complex when the number of datapoints are large.



Histogram :-

→ A histogram takes in a series of data and divides the data into a number of bins.

→ It then plots the frequency data points in each bin (ie interval of points)

→ It is useful in understanding the count of data ranges.

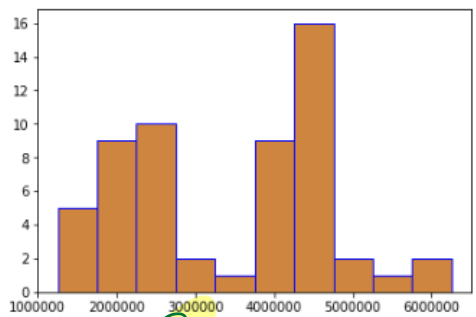
→ When to use :

Q. We should use histogram when we need the count of the variable in a plot.

→ The cumulative property gives us the end added value and helps us understand the increase in value at each bin.

Histogram

In [3]:
```
plt.hist(np_data['GrandCanyon'],
         facecolor='peru',
         edgecolor='blue',
         bins=10)
plt.show()
```



1

Components of a histogram

- **n:** Contains the frequency of each bin
- **bins:** Represents the middle value of each bin
- **patches:** The Patch object for the rectangle shape representing each bar

In [4]:
```
n, bins, patches = plt.hist(np_data['GrandCanyon'],
                            facecolor='peru',
                            edgecolor='blue',
                            bins=10)

print('n: ', n)
print('bins: ', bins)
print('patches: ', patches)
```

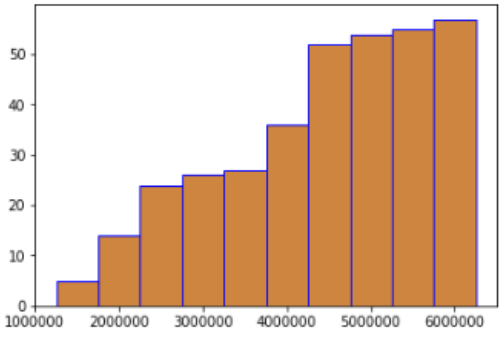
```
n:  [ 5.  9. 10.  2.  1.  9. 16.  2.  1.  2.]
bins:  [1253000.  1753123.8 2253247.6 2753371.4 3253495.2 3753619.  4253742.8
 4753866.6 5253990.4 5754114.2 6254238. ]
patches:  <a list of 10 Patch objects>
```

2

The cumulative property

If True, then a histogram is computed where each bin gives the counts in that bin plus all bins for smaller values. The last bin gives the total number of datapoints.

In [5]:
```
plt.hist(np_data['GrandCanyon'],
         facecolor='peru',
         edgecolor='blue',
         bins=10,
         cumulative=True)
plt.show()
```



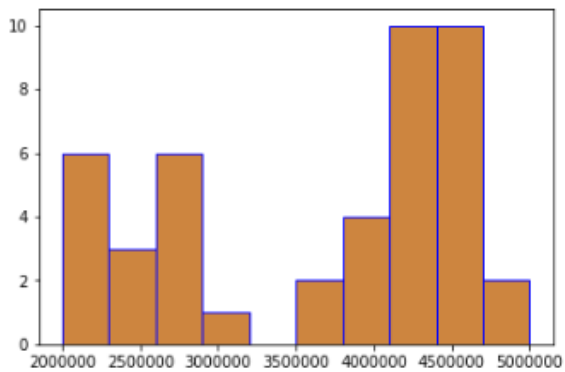
3

Check the histogram to a range of values

We only look at the data points within the range 2M-5M. This realigns the bins in the histogram

In [6]:
```
plt.hist(np_data['GrandCanyon'],
         facecolor='peru',
         edgecolor='blue',
         bins=10,
         range=(2000000, 5000000))

plt.show()
```



4

→ Multiple histograms are useful in understanding the distribution between 2 entity variables.

→ we can see that GrandCanyon had comparably more visitors than BryceCanyon.

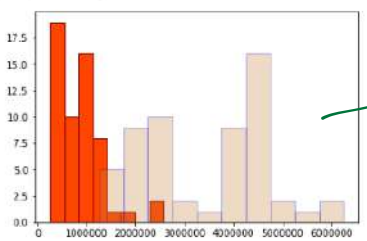
Multiple histograms

In [7]:
```
plt.hist(np_data['BryceCanyon'],
         facecolor='orangered',
         edgecolor='maroon',
         bins=8)

plt.hist(np_data['GrandCanyon'],
         facecolor='peru',
         edgecolor='blue',
         bins=10,
         alpha = 0.3)

plt.show()
```

1 2 3



4

Pie chart:

→ It is a circular plot which is divided into slices to illustrate numerical proportion.

→ The slice of a pie chart is to show the proportion of parts out of a whole.

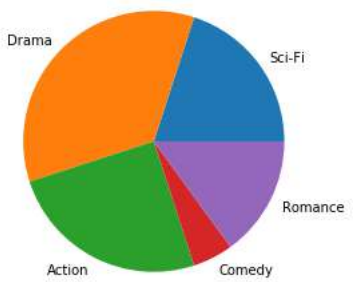
→ When to use: pie chart should be seldom used as it is diff to compare sections of the chart.

(*) Bar plot is used instead as comparing sections is easy.

Pie Chart

Pie chart shows distribution of data based on proportion of pie occupied.

```
In [3]: 1 plt.pie(t_mov['Percentage'],
        2         labels=t_mov['Sector'])
        3
        4 plt.axis('equal')
        5
        6 plt.show()
```



1

2

3

Pie chart components

- **wedges:** A list of Patch objects representing each wedge
- **texts:** List of Text objects representing the labels
- **autotexts:** List of Text objects for the numeric values - this is only available if the autopct value for the pie chart is not None

In [4]:
```
wedges, texts, autotexts = plt.pie(t_mov['Percentage'],
                                   labels=t_mov['Sector'],
                                   autopct='%.2f')

plt.axis('equal')

print('Wedges: ', wedges)
print('Texts: ', texts)
print('Autotexts: ', autotexts)
```

```
Wedges:  [<matplotlib.patches.Wedge object at 0x000001F40DD488E0>, <matplotlib.patches.Wedge object at 0x00
tplotlib.patches.Wedge object at 0x000001F40DD51B00>, <matplotlib.patches.Wedge object at 0x000001F40DD5B2E
hes.Wedge object at 0x000001F40DD5A490>]
Texts:  [Text(0.889919,0.646564,'Sci-Fi'), Text(-0.777817,0.777817,'Drama'), Text(-0.49939,-0.980107,'Actio
0.980107,'Comedy'), Text(0.980107,-0.49959,'Romance')]
Autotexts:  [Text(0.48541,0.352671,'20.00'), Text(-0.424264,0.424264,'35.00'), Text(-0.272394,-0.534604,'25
4,-0.534604,'5.00'), Text(0.534604,-0.272394,'15.00')]
```

Pie chart customizations

We set the values for the colors and autopct properties. The latter sets the format for the values to be displayed

In [5]:
```
colors = ['darkorange', 'sandybrown', 'darksalmon', 'orangered','chocolate']

plt.pie(t_mov['Percentage'],
        labels=t_mov['Sector'],
        colors=colors,
        autopct='%.2f')

plt.axis('equal')

plt.show()
```

The explode property

To highlight a particular wedge of the pie chart, we use explode to separate it from the rest of the chart.

The value for "explode" represents the fraction of the radius with which to offset each wedge.

In [6]:
```
explode = (0, 0.1, 0, 0, 0.3)

plt.pie(t_mov['Percentage'],
        labels=t_mov['Sector'],
        colors=colors,
        autopct='%.2f',
        explode=explode)

plt.axis('equal')

plt.show()
```

note:

There many different types of plots, acc to the requirements of the project we prefer wt plot to use.