FOUNDATION OF DATA ANALYSIS

Adithya Pandey
RA2111047010153

ASSIGNMENT

Q-1- Define Machine learning? Explain supervised and unsupervised learning in detail.

- Machine learning is a branch of artificial intelligence, concerned with design and development of algorithms that allow computers to evolve behaviours based on empirical data.
- Machine learning is a discipline that deals with the study of ~~materials~~ methods for pattern recognition in datasets undergoing data analysis.
- ML algorithm that iteratively learns from data and allows computers to find hidden insights without being explicitly programmed to look.

Supervised learning is an approach to creating artificial intelligence (AI), where a computer algorithm is trained on input data that has been labeled for a particular output. Below is a cancer disease detection dataset. Here, we have trained the machine learning model with input data that has been labeled for cancer diseases. Also, the problem is a binary class problem ie. the output has two labels cancer or non cancer.

S.No	Blood Report	Gene Variation	Protein Variation	Tumour	Output
1	0.5-0.9	0.7-0.9	5.6-5.7	Yes	Cancer
2	0.1-0.4	0.4-0.5	2.1-2.2	NO	Non-Cancer
....					
....					
1000000	0.1-0.3	0.4-0.49	2.1-2.212	NO	Non-Cancer

Unsupervised learning:
It's a type of learning where we don't give a target to our model while training i.e training model has only input parameter values. The model by itself has to find which way it can learn

Training data we are feeding is :-
1. Unstructured data:- May contain noisy (meaningless) data, missing values, or unknown data.
2. Unlabeled data:- Data only contains a value for input parameters, there is no targeted value (output). It is easy to collect as compared to labeled one in the supervised approach.

Types of Unsupervised learning:-
(i) Clustering: Broadly this technique is applied to group of data based on different patterns, our machine model finds.
(ii) Association: This technique is a rule based ML technique that finds out some very useful relations between parameters of a large data set.

SOME ALGORITHMS:
- K Means clustering
- DBSCAN - Density-Based spatial clustering of Applications with Noise
- BIRCH - Balanced Iterative Reducing and clustering using Hierarchies.
- Hierarchical clustering.

pply SVM, KNN, Linear Regression, and PCA on IRIS and iabetes dataset and evaluates it's performance in terms of accuracy. Write relevant code.

```
From sklearn import datasets
iris = datasets.load_iris()
iris.data
iris.target
```

☆PCA

```
From sklearn.decomposition import PCA
x_reduced = PCA(n_components=3).fit_transform(iris.data)
import matplotlib.pyplot as plt
From mpl_toolkits.mplot3d import Axes3D
From sklearn import datasets
From sklearn.decomposition import PCA
iris = dataset.load_iris()
species = iris.target ☆species
x_reduced = PCA(n_components=3).fit transform(iris.data)
```

☆SCATTERPLOT 3D

```
Fig = plt.figure()
ax = Axes3D(Fig)
ax.set_title('Iris dataset by PCA', size=14)
ax.scatter(x_reduced[:,0], x_reduced[:,1], x_reduced
           [:, 2], c=species)
ax.set_xlabel('First eigenvector')
ax.set_ylabel('second eigenvector')
ax.set_zlabel('Third eigenvector')
ax.w_xaxis.set_ticklabels(())
ax.w_yaxis.set_ticklabels(())
ax.w_zaxis.set_ticklabels(())
```

☆KNN

```
import numpy as np
From sklearn import datasets
np.random.seed(0)
iris = datasets.load_iris()
x = iris.data
y = iris.target
i = np.random.permutation(len(iris.data))
x_train = x[i[:-10]]
```

```
y_train = y[:-10]
x_test = x[-10:]
y_test = y[-10:]
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
knn.score(x_test, y_test)
```

Linear Regression

```
from sklearn import linear_model
linreg = linear_model.LinearRegression()
x_train = iris.data[:-20]
y_train = iris.target[:-20]
x_test = iris.data[-20:]
y_test = iris.target[-20:]
linreg.fit(x_train, y_train)
linreg.score(x_train, y_train)*100
```

SVM

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
iris = datasets.load_iris()
x = iris.data[:, :2]
y = iris.target
h = .05
svc = svm.SVC(kernel='poly', C=1.0, degree=3).fit(x, y)
x_min, x_max = x[:, 0].min() - .5, x[:, 0].max() + .5
y_min, y_max = x[:, 1].min() - .5, x[:, 1].max() + .5
h = .02
x, y = np.meshgrid(np.arange(x_min, x_max, h), np.arange
                   (y_min, y_max, h))
z = svc.predict(np.c_[x.ravel(), y.ravel()])
z = z.reshape(x.shape)
plt.contourf(x, y, z, alpha=0.4)
plt.contour(x, y, z, colors='k')
plt.scatter(x[:, 0], x[:, 1], c=y)
```

...n, Regression, and PCA ... performance

```
.score(x,y)*100
DIABETES
from sklearn import datasets
diabetes = datasets.load_diabetes()
diabetes.data
diabetes.target
*Linear Regression
From sklearn import linear_model
linreg = linear_model.LinearRegression()
From sklearn import datasets
diabetes = datasets.load_diabetes()
x_train = diabetes.data[:-20]
y_train = diabetes.target[:-20]
x_test = diabetes.data[-20:]
y_test = diabetes.target[-20:]
linreg.fit(x_train, y_train)
linreg.predict(x_test)
*SVR
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
From sklearn import datasets
diabetes = datasets.load_diabetes()
x_train = diabetes.data[:-20]
y_train = diabetes.target[:-20]
x_test = diabetes.data[-20:]
y_test = diabetes.target[-20:]
x0_test = x_test[:,2]
x0_train = x_train[:,2]
x0_test = x0_test[:, np.newaxis]
x0_train = x0_train[:, np.newaxis]
x0_test.sort(axis=0)
x0_test = x0_test*100
x0_train = x0_train*100
SVR = svm.SVR(kernel='linear', C=1000)
```

```
svm2 = svm.SVR(kernel='poly', C=1000, degree=2)
svm3 = svm.SVR(kernel='poly', C=1000, degree=3)
svm.fit(x0_train, y_train)
svm2.fit(x0_train, y_train)
svm3.fit(x0_train, y_train)
y = svm.predict(x0_test)
y2 = svm2.predict(x0_test)
y3 = svm3.predict(x0_test)
plt.scatter(x0_test, y_test, color='k')
plt.plot(x0_test, y2, c='r')
plt.plot(x0_test, y3, c='g')
svm.score(x0_train, y_train)*100
```

```
#PCA
from sklearn.decomposition import PCA
x_reduced = PCA(n_components=3).fit_transform(iris.data)
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA
x_reduced = PCA(n_components=3).fit_transform(diabetes.data)
species = diabetes.target
#SCATTER PLOT 3D
fig = plt.figure()
ax = Axes3D(fig)
ax.set_title('Iris Dataset by PCA', size=14)
ax.scatter(x_reduced[:,0], x_reduced[:,1], x_reduced[:,2],
           c=species)
ax.set_xlabel('First Eigenvector')
ax.set_ylabel('second Eigenvector')
ax.set_zlabel('Third Eigenvector')
ax.w_xaxis.set_ticklabels(())
ax.w_yaxis.set_ticklabels(())
ax.w_zaxis.set_ticklabels(())
```

-NN

```
mport numpy as np
rom sklearn import datasets
np.random.seed(0)
x = diabetes.data
y = diabetes.target
i = np.random.permutation(len(diabetes.data))
x_train = x[i[:-10]]
y_train = y[i[:-10]]
x_test = x[i[-10:]]
y_test = y[i[-10:]]
From sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(x_train, y_train)
knn.score(x_test, y_test)*100
```

Aim: Draw a horizontal bar chart with Matplotlib.

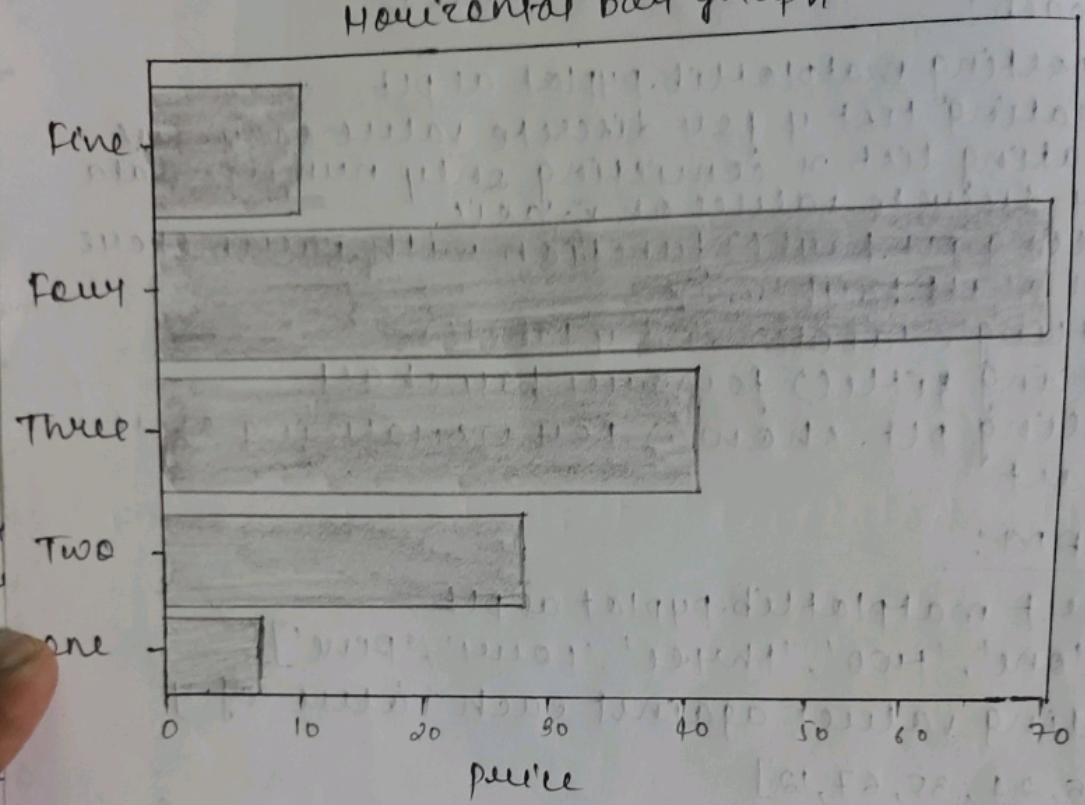
Creating a horizontal bar chart:

Approach:

- Importing matplotlib.pyplot as plt
- Creating list y for discrete value on y-axis
- Creating list x containing only numeric data for discrete values on x-axis.
- Calling plt.barh() function with parameters y, x as plt.barh(y, x)
- Setting x label() and y label()
- Setting title() for our bar chart
- Calling plt.show() for visualizing our chart

PROGRAM:

```
import matplotlib.pyplot as plt
y=['one','two','three','four','five']
# getting values against each value of y
x=[5,24,35,67,12]
plt.barh(y,x)
# setting label of y-axis
plt.ylabel("pen sold")
# setting label of x-axis
plt.xlabel("price")
plt.title("Horizontal bar graph")
plt.show()
```

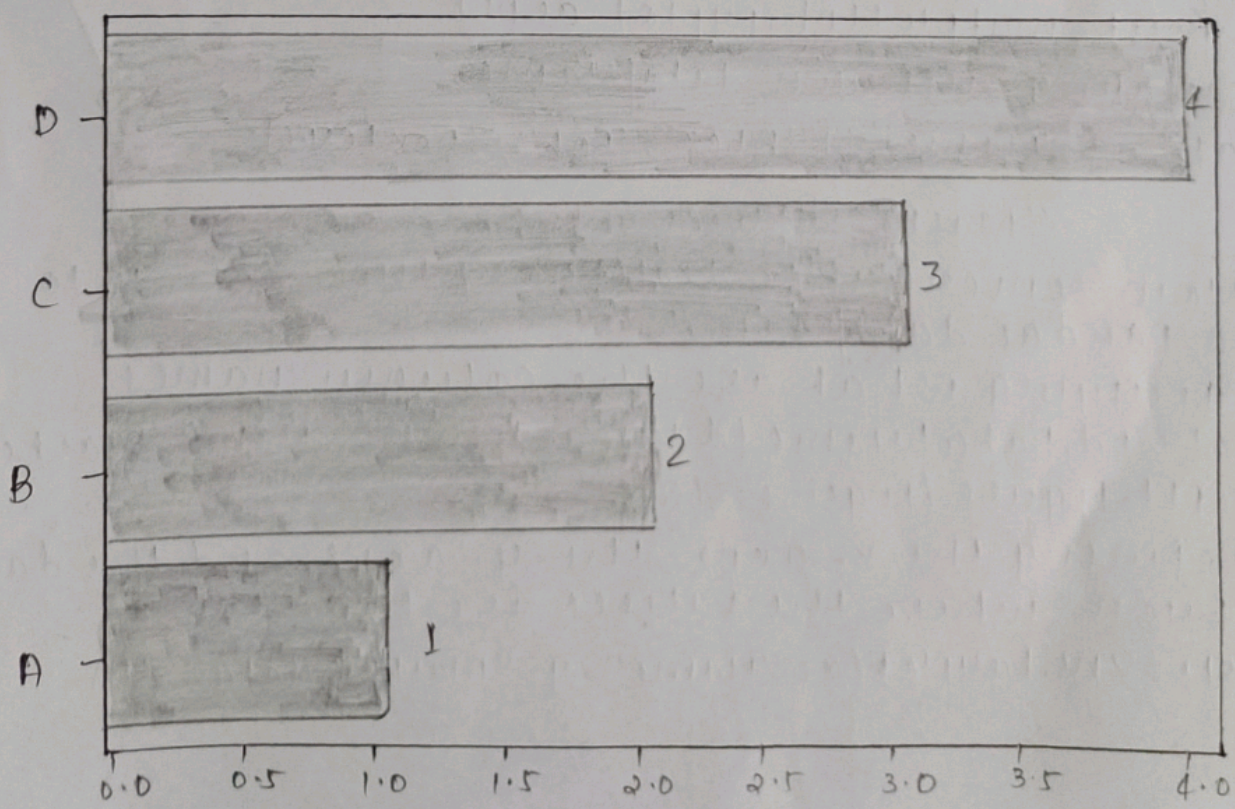

Horizontal bar graph
Four
Three
Two
0
10
20
30
40
50
60
70
price

ADITYA PANDEY - RA2111047010153

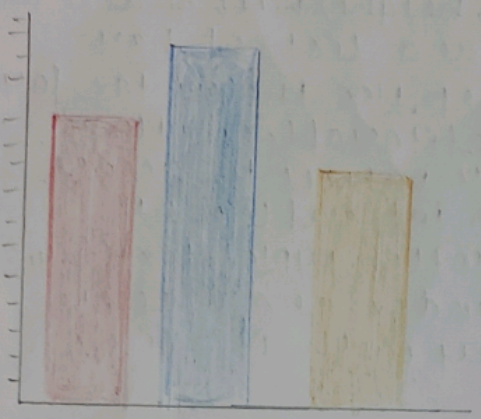
How to display the value of each bar in a bar chart using Matplotlib?

Call matplotlib.pyplot.barh(x, height) with x as a list of bars names and height as a list of bar values to create a bar chart. Use the syntax for index, value in enumerate(iterable) with iterable as the list of bar values to access each index, values pair in iterables. At each iteration call matplotlib.pyplot.text(x,y,s) with x as value, y as index and s as str(value) to label each bar with its size.

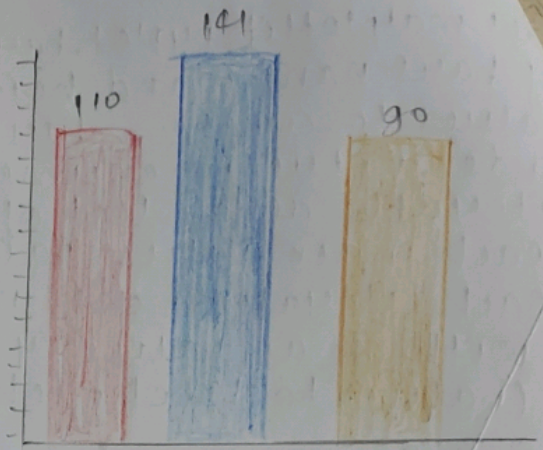
```
x=["A","B","C","D"]
y=[1,2,3,4]
plt.barh(x,y)
for index,value in enumerate(y):
  plt.text(value,index,str(value))
```



5- How to annotate the following barplot?



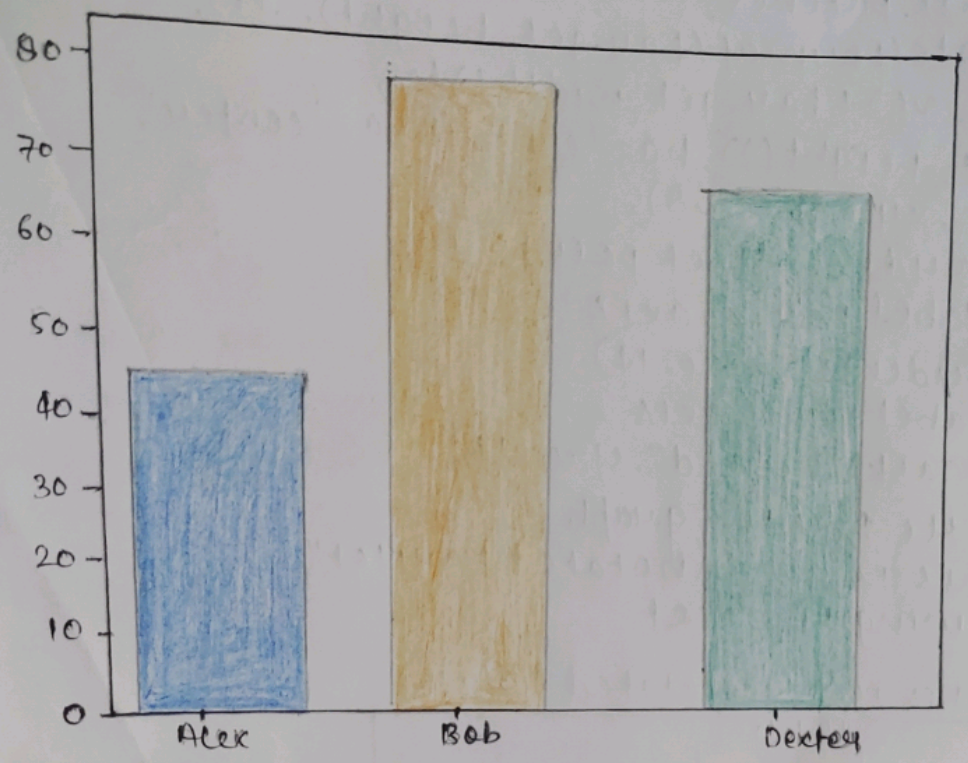
Non-annotated subplot



Annotated subplot

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# creating our own dataframe
data = {"Name": ["Alex", "Bob", "Dexter"],
        "Marks": [45, 78, 65]}
# Now convert this dictionary type data into
  a pandas dataframe
# Specifying what are the column names
df = pd.DataFrame(data, column = ['Name', 'Marks'])
plt.figure(figsize = (8,6))
# Defining the x-axis, the y-axis and the data
# from where the values are to be taken
plots = sns.barplot(x = "Name", y = "Marks", data = df)
```

* setting the x-axis label and its size
plt.xlabel("Students", size=15)
* setting the y-axis label and its size
plt.ylabel("Marks secured", size=15)
* final plotting the graph
plt.show()




```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
* creating our own dataframe
  data = {"Name":["Alex","Bob","Dexter"],
          "Marks":[110,141,90]}
* Now convert this dictionary type data into a
* pandas dataframe.
* specifying what are the columns name
  df = pd.DataFrame(data, columns=['Name','Marks'])
plt.figure(figsize=(8,8))
```

* Defining the values for x-axis, y-axis
* and from which dataframe the values are picked

```
plots = sns.barplot(x="Name", y="Marks", data=df)
```

* Annotating over the bars one by one

```
for bar in plots.patches:
    plots.annotate(format(bar.get_height(), '.2f'),
        (bar.get_x() + bar.get_width()/2,
        bar.get_height()), ha='center', va='center',
        size=15, xytext=(0,8),
        textcoords='offset points')
```

* setting the label for x-axis

```
plt.xlabel("Students", size=14)
```

* setting the label for y-axis

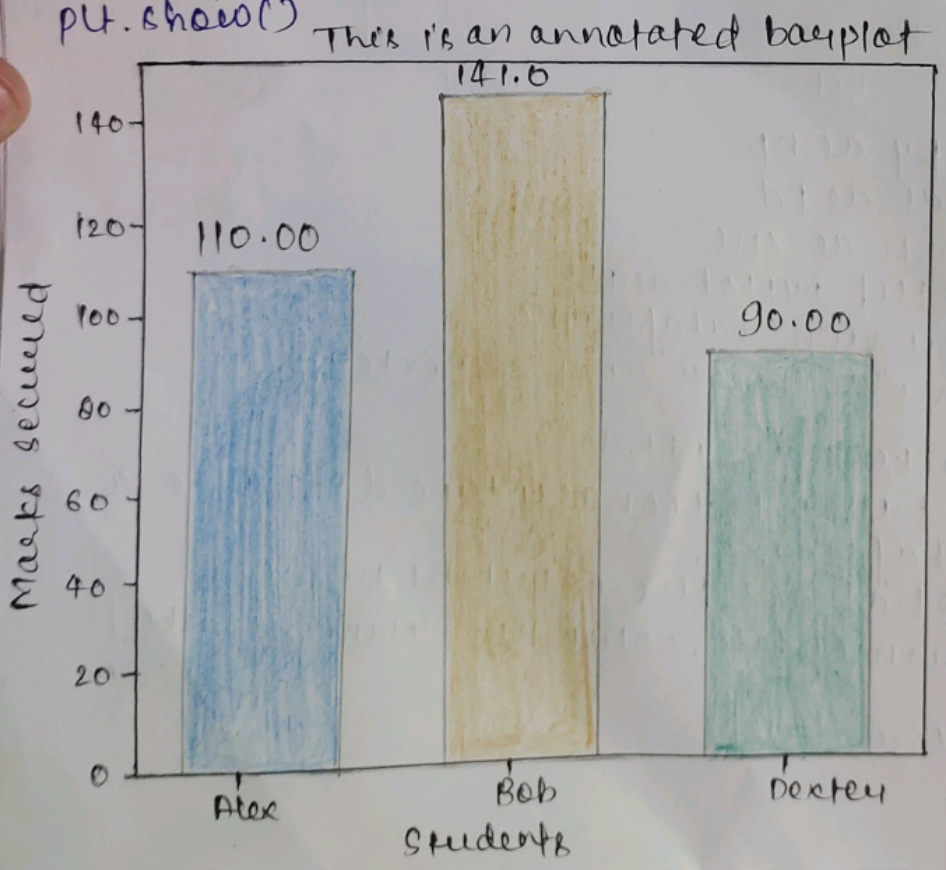
```
plt.ylabel("Marks secured", size=14)
```

* setting the title for the graph

```
plt.title("This is an annotated barplot")
```

* finally showing the plot

```
plt.show()
```



How to add a legend to a scatter plot in Matplotlib?

We can add a legend to the plot using the Matplotlib module. We use the matplotlib.pyplot.legend() method to mark out and label the element of the graph.

The syntax to add a legend to the plot:

```
matplotlib.pyplot.legend(["Title"], ncol=1, loc="upper left", bbox_to_anchor=(1,1))
```

The parameter used above are described below:

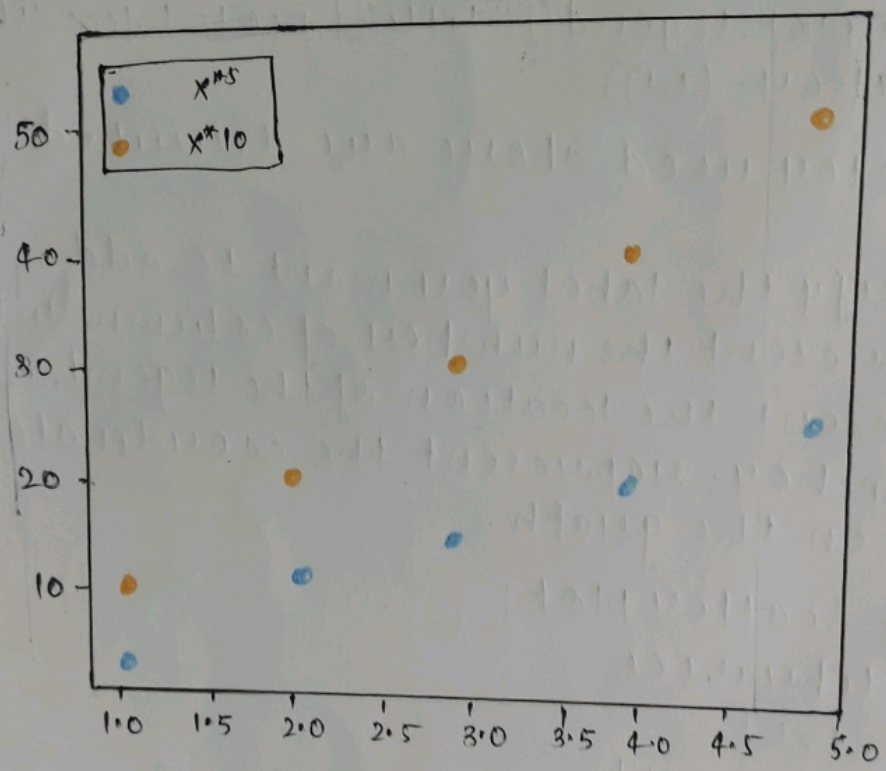
- title: specify the label you want to add
- ncol: represent the number of columns in legend.
- loc: represent the location of the legend.
- bbox_to_anchor: represent the coordinate of legend on the graph.

Example of scatter plot:

```
# Import libraries
x = [1, 2, 3, 4, 5]
y1 = [5, 10, 15, 20, 25]
y2 = [10, 20, 30, 40, 50]
# scatter plot
plt.scatter(x, y1)
plt.scatter(x, y2)
# Add legend
plt.legend(["x*5", "x*10"])
# Display
plt.show()
```

- In the above example, we import the pyplot module of matplotlib.
- After this, we define data coordinates

- plt.scatter() method is used to plot scatter
- plt.legend() method used to add legend to pl
- plt.show() method is used to visualize the pl on the user's screen.



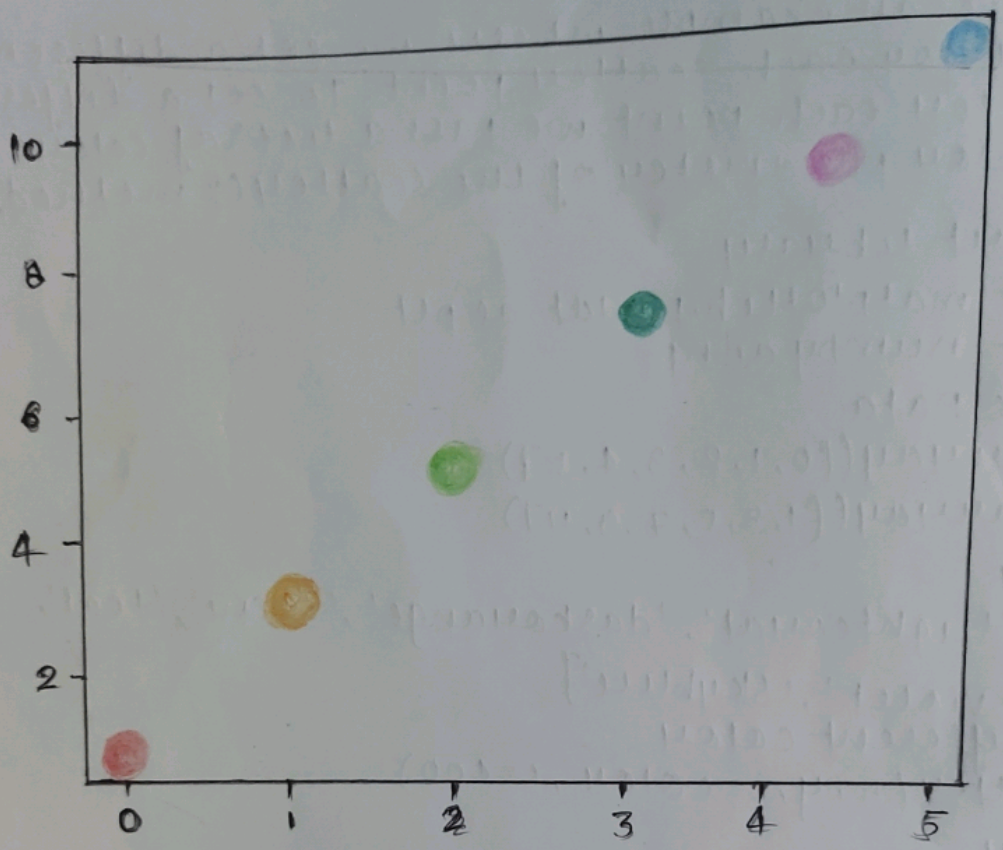
How to create a scatter plot with several colors in Matplotlib?

We'll see an example, where we set a different color for each scatter point. To set a different color for each point we pass a list of colors to the color parameter of the scatter() method.

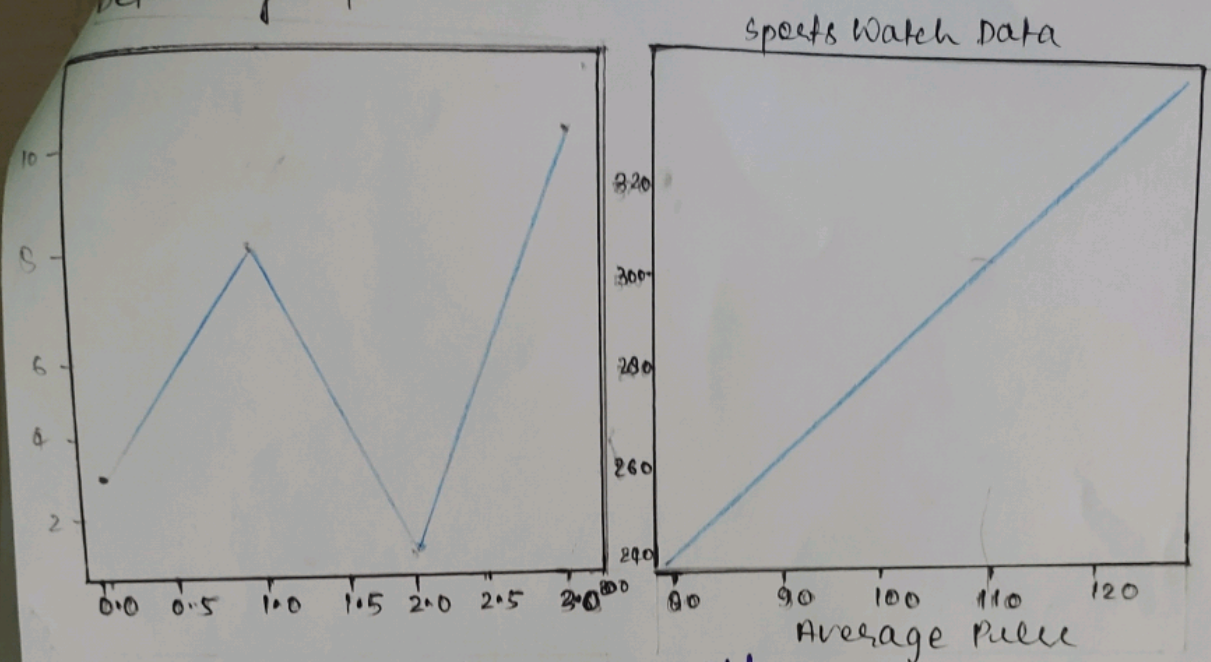
```
# Import library
import matplotlib.pyplot as plt
import numpy as np
# Define Data
x = np.array([0, 1, 2, 3, 4, 5])
y = np.array([1, 3, 5, 7, 9, 11])
# color
color = ['lightcoral', 'darkorange', 'olive', 'teal',
         'violet', 'skyblue']
# set different color
plt.scatter(x, y, c=color, s=400)
# Display
plt.show()
```

The following are the steps:

- Import library matplotlib.pyplot and numpy for data visualization and creation.
- Next, define data axes using array() method of numpy.
- Then create list of colors.
- To plot a scatter graph, use scatter() function.
- To set the different color for each scatter maker pass color parameter and set it's value to given list of colors.


10
8
6
4
2
0
1
2
3
4
5

...w to include markers, label, grid and subplots to below graph.

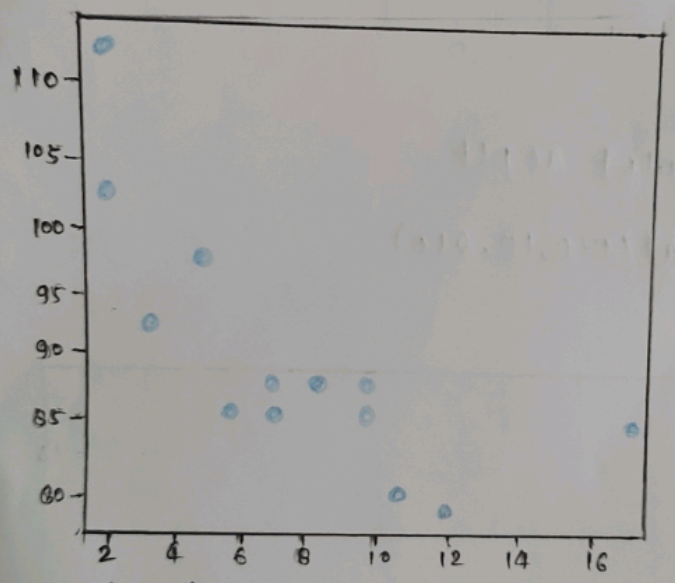



```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
df= pd.DataFrame({0:[3,8,2,10], 1:[240,260,300,330]})
plt.figure(figsize=(10,10))
plt.subplot(2,2,1)
plt.plot([0,1,2,3], df[0])
plt.xticks([0.0,0.5,1.0,1.5,2.0,2.5,3.0],[0.0,0.5,1.0,1.5,2.0,2.5,3.0])
plt.yticks([2,4,6,8,10])
plt.subplot(2,2,2)
plt.plot([80,90,110,125], df[1])
plt.xticks([80,90,100,110,120],[80,90,100,110,120])
plt.yticks([240,260,280,300,320])
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.title("Sport Watch Data")
```

histogram, bar charts, scatter plots, and a charts on any data.

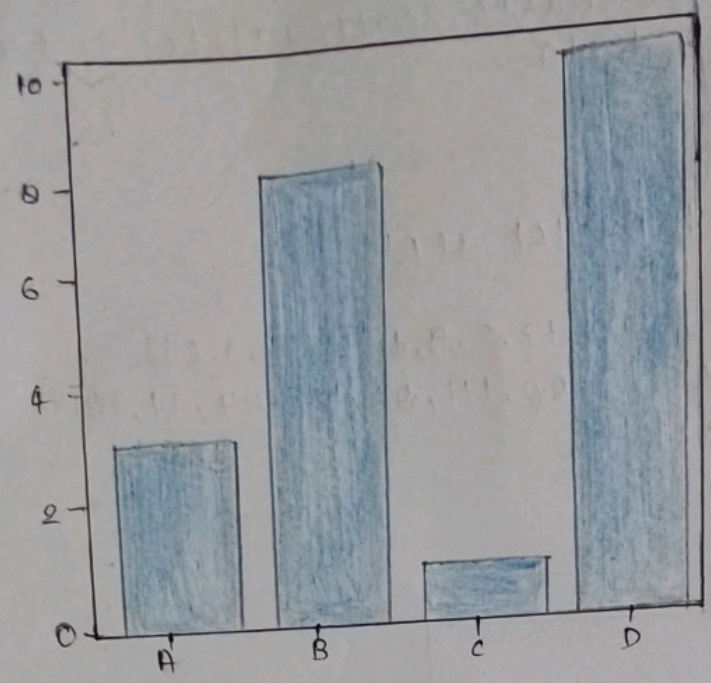
*scatter plot

```
import matplotlib.pyplot as plt
import numpy as np
x=np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y=np.array([99,86,87,88,111,86,103,87,94,78,77,
            85,86])
plt.scatter(x,y)
plt.show()
```



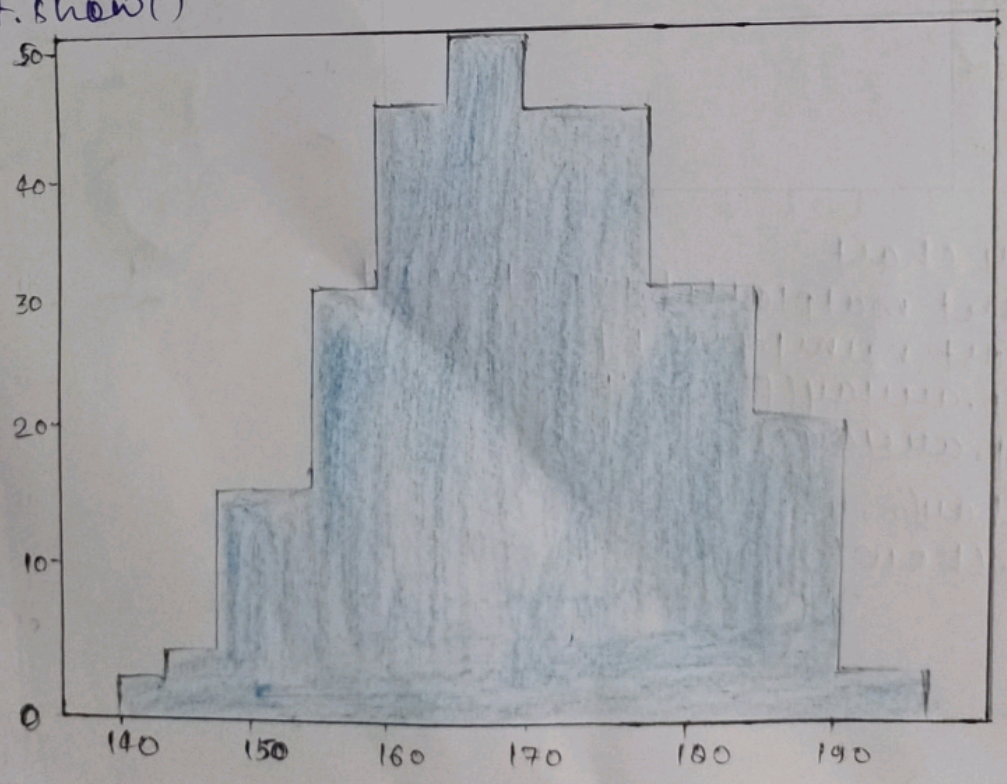
*Bar chart

```
import matplotlib.pyplot as plt
import numpy as np
x=np.array(["A","B","c","D"])
y=np.array([3,8,1,10])
plt.bar(x,y)
plt.show()
```



*Histogram chart

```
import matplotlib.pyplot as plt
import numpy as np
x= np.random.normal(170,10,250)
plt.hist(x)
plt.show()
```



e chart

```
mport matplotlib.pyplot as plt
mport numpy as np
y= np.array([35,25,25,15])
plt.pie(y)
plt.show()
```