Programming for AI Lab 09

Course: Programming for AI (AI2001) Semester: Fall 2023

Instructor: Sameer Faisal T.A: N/A

Note:

- Maintain discipline during the lab.
- Listen and follow the instructions as they are given.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

Line Graphs

➢ Basic Line Plot

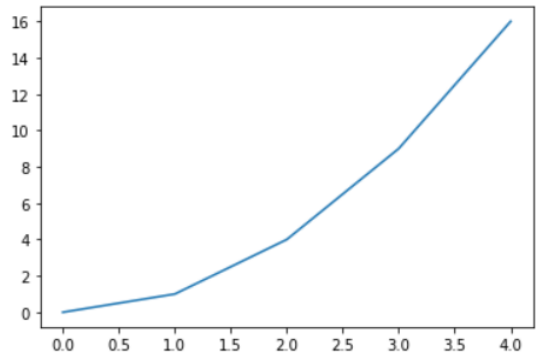
Line graphs are helpful for visualizing how a variable changes over time.

Some possible data that would be displayed with a line graph:

- Average prices of gasoline over the past decade.
- Weight of an individual over the past couple of months.
- Average temperature along a line of longitude over different latitudes

Using Matplotlib methods, the following code will create a simple line graph using .plot() and display it using .show():

```
from matplotlib import pyplot as plt

x_values=[0,1,2,3,4]
y_values=[0,1,4,9,16]

plt.plot(x_values,y_values)
plt.show()
```



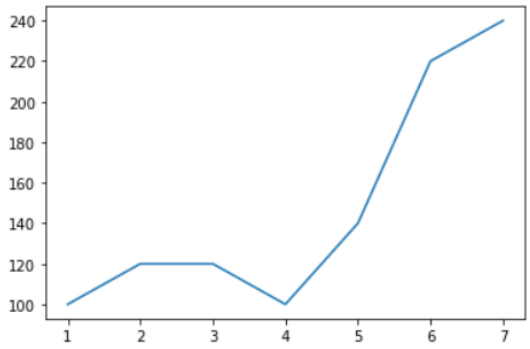
➢ Example - 1

We are going to make a simple graph representing someone's spending on lunch over the past week.

First, define two lists, days and money spent, that contain the following integers:

Days	Money Spent
1	100
2	120
3	120
4	100
5	140
6	220
7	240

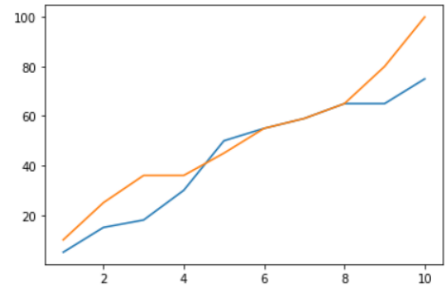
```
from matplotlib import pyplot as plt

days = range(1,8) # days = [1, 2, 3, 4, 5, 6, 7]

money_spent = [100, 120, 120, 100, 140, 220, 240]

plt.plot(days, money_spent)

plt.show()
```



➢ Multiple Line Graphs

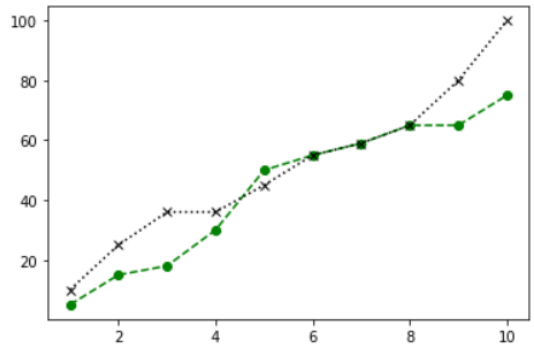
We can also have multiple line plots displayed on the same set of axes. This can be very useful if we want to compare two datasets with the same scale and axis categories
Matplotlib will automatically place the two lines on the same axes and give them different colors if you call plt.plot() twice.

```
from matplotlib import pyplot as plt

overs=[1,2,3,4,5,6,7,8,9,10]
Pakistan=[5,15,18,30,50,55,59,65,65,75]
Australia=[10,25,36,36,45,55,59,65,80,100]
plt.plot(overs, Pakistan)
plt.plot(overs, Australia)

plt.show()
```



➢ Changing Line Styles

```
from matplotlib import pyplot as plt

overs=[1,2,3,4,5,6,7,8,9,10]
Pakistan=[5,15,18,30,50,55,59,65,65,75]
NewZealand=[10,25,36,36,45,55,59,65,80,100]
plt.plot(overs, Pakistan,color='g',linestyle='--',marker='o')
plt.plot(overs, NewZealand ,color='black',linestyle=':',marker='x')

plt.show()
```



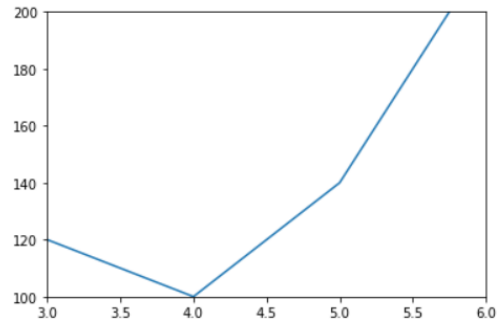
➢ Axis and Labels

Sometimes, it can be helpful to zoom in or out of the plot, especially if there is some detail we want to address. To zoom, we can use plt.axis().

We use plt.axis() by feeding it a list as input. This list should contain:

- The minimum x-value displayed
- The maximum x-value displayed
- The minimum y-value displayed
- The maximum y-value displayed

```
from matplotlib import pyplot as plt
days = range(1,8) # days = [1, 2, 3, 4, 5, 6, 7]
money_spent = [100, 120, 120, 100, 140, 220, 240]
plt.plot(days, money_spent)
plt.axis([3,6,100,200]) #x-min,x-max,y-min,y-max
plt.show()
```



➢ Example - 2

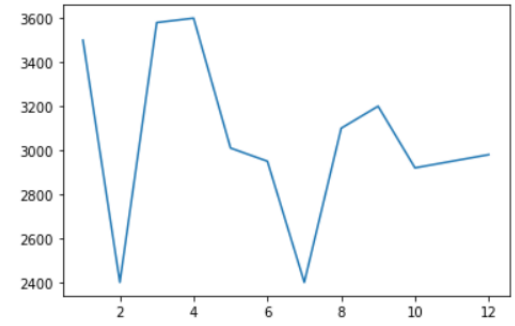
We have plotted a line representing someone's spending on coffee over the past 12 years.

Years	Spending
1	3500
2	2400
3	3580
4	3600
5	3010
6	2950
7	2400
8	3100
9	3200
10	2920
11	2950
12	2980

Use plt.axis() to modify the axes so that the x-axis goes from 1 to 12, and the y-axis goes from 2900 to 3100.

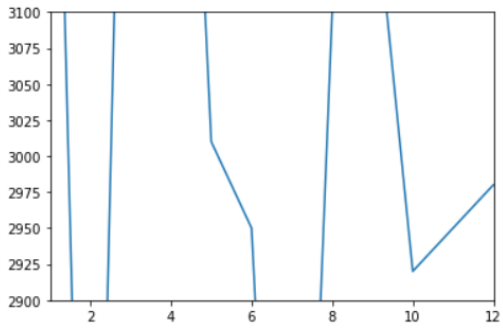
Original Graph:

```
from matplotlib import pyplot as plt
x = range(1,13)
y = [3500, 2400, 3580, 3600, 3010, 2950, 2400, 3100, 3200,
2920, 2950, 2980]
plt.plot(x, y)
plt.show()
```

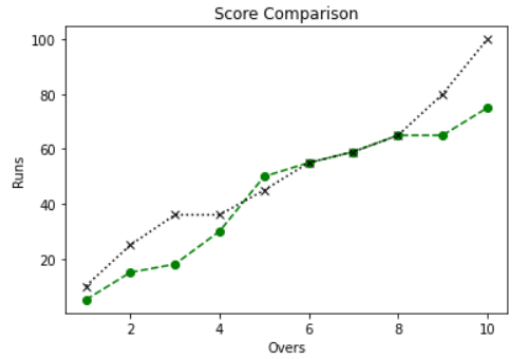


Modified Graph:

```
from matplotlib import pyplot as plt
x = range(1,13)
y = [3500, 2400, 3580, 3600, 3010, 2950, 2400, 3100, 3200
2920, 2950, 2980]
plt.plot(x, y)
plt.axis([1,12,2900,3100])
plt.show()
```

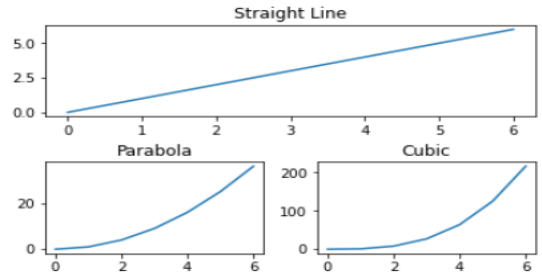


```
from matplotlib import pyplot as plt
overs=[1,2,3,4,5,6,7,8,9,10]
Pakistan=[5,15,18,30,50,55,59,65,65,75]
NewZealand=[10,25,36,36,45,55,59,65,80,100]
plt.plot(overs, Pakistan,color='g',linestyle='--',marker='o')
plt.plot(overs, NewZealand ,color='black',linestyle=':',marker='x')
plt.title("Score Comparison")
plt.xlabel("Overs")
plt.ylabel("Runs")
plt.show()
```



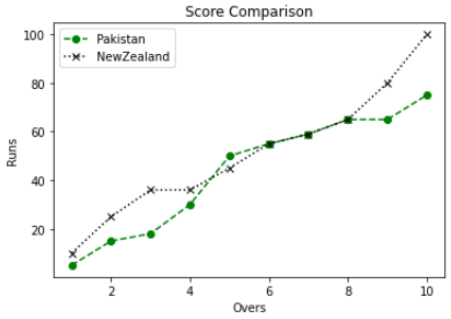
➢ Example – 3 (Straight Line, Cubic and Parabola)

```
from matplotlib import pyplot as plt
x = range(7)
straight_line = [0, 1, 2, 3, 4, 5, 6]
parabola = [0, 1, 4, 9, 16, 25, 36]
cubic = [0, 1, 8, 27, 64, 125, 216]
# Subplot 1
plt.subplot(2, 1, 1)
plt.title("Straight Line")
plt.plot(x, straight_line)
# Subplot 2
plt.subplot(2, 2, 3)
plt.title("Parabola")
plt.plot(x, parabola)
# Subplot 3
plt.subplot(2, 2, 4)
plt.plot(x, cubic)
plt.title("Cubic")
plt.subplots_adjust(hspace=0.5,wspace=0.25, bottom=0.2)
plt.show()
```



➢ Legends

```
from matplotlib import pyplot as plt
overs=[1,2,3,4,5,6,7,8,9,10]
Pakistan=[5,15,18,30,50,55,59,65,65,75]
NewZealand=[10,25,36,36,45,55,59,65,80,100]
plt.plot(overs, Pakistan,color='g',linestyle='--',marker='o')
plt.plot(overs, NewZealand ,color='black',linestyle=':',marker='x')
plt.legend(['Pakistan', 'NewZealand'])
plt.title("Score Comparison")
plt.xlabel("Overs")
plt.ylabel("Runs")
plt.show()
```



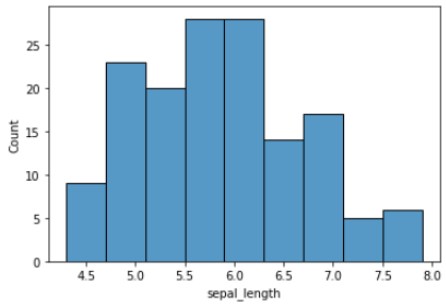
Sometimes, it's easier to label each line as we create it.
If we want, we can use the keyword label inside of plt.plot().
If we choose to do this, we don't pass any labels into plt.legend().

Seaborn Library for Visualization (Based on Matplotlib)

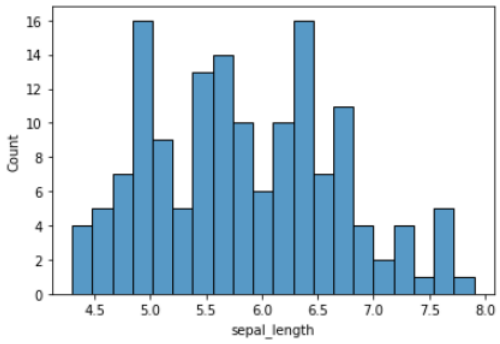
This library is also famous for visualization especially for visualization of pandas dataframes as syntax is easier.

```
#Histogram of dataframe using seaborn
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset("iris")    #Loading built in iris flower dataset
print(df)
sns
sns.histplot(data=df, x="sepal_length")   #creating histogram of column sepal_length
plt.show()
```

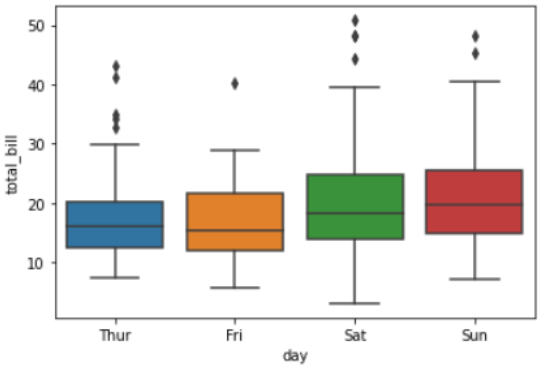
```
     sepal_length  sepal_width  petal_length  petal_width    species
0             5.1          3.5           1.4          0.2     setosa
1             4.9          3.0           1.4          0.2     setosa
2             4.7          3.2           1.3          0.2     setosa
3             4.6          3.1           1.5          0.2     setosa
4             5.0          3.6           1.4          0.2     setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

[150 rows x 5 columns]
```



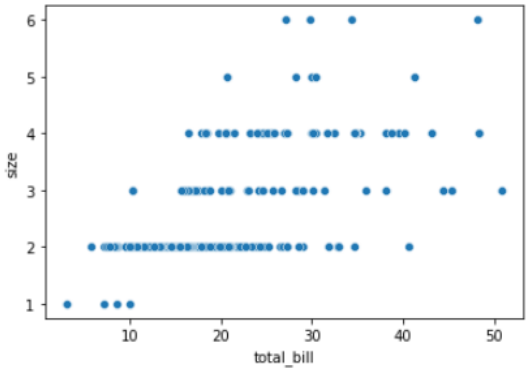
```
#histogram using seaborn with customized number of bins. Here bins=20
sns.histplot(data=df, x="sepal_length", bins=20)
plt.show()
```




```
# boxplot syntax using seaborn is also very easy
tips = sns.load_dataset('tips')   #loading built in dataset tips.csv
sns.boxplot(x = 'day', y = 'total_bill', data=tips)
plt.show()
```




```
# scatterplot syntax using seaborn is also very easy
sns.scatterplot(data=tips, x="total_bill", y="size")
plt.show()
```



Scikit Learn Library (sklearn)

Scikit-learn (sklearn) is a library in Python that provides many unsupervised and supervised learning algorithms. It's built upon some of the technology you might already be familiar with, like NumPy, pandas, and Matplotlib.

The functionality that scikit-learn provides include:

- Regression, including Linear and Logistic Regression algorithms etc
- Classification, including K-Nearest Neighbors algorithms etc
- Clustering, including K-Means and K-Means++ etc
- Model selection
- Preprocessing, including Min-Max Normalization

Machine learning has three types:

- Supervised machine learning (class/label/target is given)
- Un-supervised machine learning (class/label/target is not given)
- Reinforcement Learning (reward based)

➢ Supervised Algorithms

Classification algorithms (like KNN, Decision Tree, Naive Bayes etc.)
Regression (like linear regression, logistic regression etc.)

➢ Unsupervised Algorithms

Clustering algorithms (like kmeans, kmeans++ etc.)

Supervised Algorithms

K-Nearest Neighbour Algorithm (KNN)

The K-nearest neighbors (KNN) algorithm is a type of supervised machine learning algorithms. KNN is extremely easy to implement in its most basic form, and yet performs quite complex classification tasks. It is a lazy learning algorithm since it doesn't have a specialized training phase. Rather, it uses all of the data for training while classifying a new data point or instance. KNN is a non-parametric learning algorithm, which means that it doesn't assume anything about the underlying data.

This is an extremely useful feature since most of the real-world data doesn't really follow any theoretical assumption e.g. linear-separability, uniform distribution, etc.

The intuition behind the KNN algorithm is one of the simplest of all the supervised machine learning algorithms.

It simply calculates the distance of a new data point to all other training data points. The distance can be of any type e.g. Euclidean or Manhattan etc. It then selects the K-nearest data points, where K can be any integer.

Finally, it assigns the data point to the class to which the majority of the K data points belong.

Algorithm - Pseudocode description of the nearest neighbor algorithm:

Require: a set of training instances

Require: a query instance

1: Iterate across the instances in memory to find the nearest neighbor—this is the instance with the shortest distance across the feature space to the query instance.

2: Make a prediction for the query instance that is equal to the value of the target feature of the nearest neighbor.

```
# Load libraries
import pandas
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

```
# Lets apply K-Nearest Neighbour Algorithm on Iris Flower dataset
# Iris flower dataset has four attributes (first four columns) based on which it tells
#which type of flower it is out of total 3 types

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pandas.read_csv(r"iris.data", names=names)

dataset
```

	sepal-length	sepal-width	petal-length	petal-width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

```
dataset.shape #finding dimensions
```

```
(150, 5)
```

```
dataset.groupby('class').size()
```

```
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

```
# In X, we are having first four columns of every row i-e sepal_length, sepal_width, petal_length, petal_width
# In Y, we are having 4th column of every row which is basically class/label i-e Iris satosa, Iris verginica or Iris versicolor

array = dataset.values
X = array[:,0:4]
Y = array[:,4]

t_size = 0.20   # training size is 0.2 (20 percent), testing size will be automatically set to 1-0.2 = 0.8 (80 percent)
seed = 7        #if this seed is set to any constant number then it will generate same random numbers every time

#dividing into training and testing
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=t_size,random_state=seed)

#check by printing what data came into these four variables

print("X_train: ", X_train)
print("Y_train : ", Y_train)
print("X_test : ", X_test)
print("Y_test : ", Y_test)
```

Output (Some output is truncated intentionally):

```
X_train:  [[6.2 2.8 4.8 1.8]
 [5.7 2.6 3.5 1.0]
 [4.6 3.6 1.0 0.2]
 [6.9 3.1 5.4 2.1]
 [6.4 2.9 4.3 1.3]
 [4.8 3.0 1.4 0.3]
 [5.5 3.5 1.3 0.2]
 [5.4 3.9 1.7 0.4]
 [5.1 3.5 1.4 0.3]
 [7.1 3.0 5.9 2.1]
 [6.7 3.3 5.7 2.1]
 [6.8 2.8 4.8 1.4]
 [6.4 2.8 5.6 2.2]
 [6.5 3.0 5.5 1.8]
 [5.7 3.0 4.2 1.2]
 [5.0 3.3 1.4 0.2]
 [6.7 3.1 4.4 1.4]
 [6.0 2.2 4.0 1.0]
 [6.4 2.7 5.3 1.9]
```

```
Y_train :  ['Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'
 'Iris-versicolor' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa'
 'Iris-virginica' 'Iris-virginica' 'Iris-versicolor' 'Iris-virginica'
 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-virginica' 'Iris-setosa' 'Iris-setosa'
 'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-virginica'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa'
 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-virginica'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'
 'Iris-setosa' 'Iris-setosa' 'Iris-virginica' 'Iris-virginica'
 'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor'
 'Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-versicolor'
 'Iris-setosa' 'Iris-virginica' 'Iris-virginica' 'Iris-versicolor'
 'Iris-setosa' 'Iris-setosa' 'Iris-versicolor' 'Iris-virginica'
 'Iris-setosa' 'Iris-virginica' 'Iris-virginica' 'Iris-versicolor'
 'Iris-setosa' 'Iris-versicolor' 'Iris-setosa' 'Iris-virginica'
 'Iris-virginica' 'Iris-setosa' 'Iris-setosa' 'Iris-virginica'
 'Iris-versicolor' 'Iris-virginica' 'Iris-virginica' 'Iris-versicolor'
 'Iris-setosa' 'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-setosa'
```

```
X_test :  [[5.9 3.0 5.1 1.8]
 [5.4 3.0 4.5 1.5]
 [5.0 3.5 1.3 0.3]
 [5.6 3.0 4.5 1.5]
 [4.9 2.5 4.5 1.7]
 [4.5 2.3 1.3 0.3]
 [6.9 3.1 4.9 1.5]
 [5.6 2.7 4.2 1.3]
 [4.8 3.4 1.6 0.2]
 [6.4 3.2 4.5 1.5]
 [6.7 3.0 5.0 1.7]
 [6.0 3.4 4.5 1.6]
 [5.2 4.1 1.5 0.1]
 [7.2 3.6 6.1 2.5]
 [5.2 3.4 1.4 0.2]
 [5.9 3.2 4.8 1.8]
 [6.7 2.5 5.8 1.8]
 [6.4 3.1 5.5 1.8]
 [5.1 3.8 1.6 0.2]
 [4.9 3.1 1.5 0.1]
```

```
Y_test :  ['Iris-virginica' 'Iris-versicolor' 'Iris-setosa' 'Iris-versicolor'
 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-setosa' 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-setosa' 'Iris-virginica' 'Iris-setosa' 'Iris-versicolor'
 'Iris-virginica' 'Iris-virginica' 'Iris-setosa' 'Iris-setosa'
 'Iris-versicolor' 'Iris-virginica' 'Iris-versicolor' 'Iris-virginica'
 'Iris-virginica' 'Iris-virginica' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-virginica' 'Iris-virginica']
```

```
knn = KNeighborsClassifier(n_neighbors=2) # two nearest neighbors i-e K=2
knn.fit(X_train, Y_train)      # training the model
predictions = knn.predict(X_test)  # now testing on test data to get class of test data
print((accuracy_score(Y_test, predictions))) # comparing results predicted by model with actual to get accuracy score
```

```
0.9333333333333333
```

```
# evaluation metrics other than accuracy
print((classification_report(Y_test, predictions)))
print(confusion_matrix(Y_test, predictions))
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00         7
Iris-versicolor       0.86      1.00      0.92        12
 Iris-virginica       1.00      0.82      0.90        11

       accuracy                           0.93        30
      macro avg       0.95      0.94      0.94        30
   weighted avg       0.94      0.93      0.93        30

[[ 7  0  0]
 [ 0 12  0]
 [ 0  2  9]]
```

Precision: Precision is a measure that assesses the accuracy of the positive predictions made by the algorithm. It quantifies the ratio of true positive predictions to all positive predictions, indicating how well KNN correctly identifies instances of the target class.

Recall: Recall is a measure that evaluates the ability of the algorithm to identify all instances of the target class. It quantifies the ratio of true positive predictions to all actual instances of the target class, emphasizing the completeness of the algorithm's predictions.

F1-score: F1-score is a single metric that combines both precision and recall to provide a balanced measure of the algorithm's performance. It is particularly useful when there is an uneven class distribution and helps assess the trade-off between precision and recall.

Support: Support represents the number of instances in the dataset that belong to a specific class. It is a useful indicator of the distribution of each class and helps in understanding the relative prevalence of different classes in the dataset.

Lab Tasks

1. Define three lists, x, y1, and y2 and fill them with integers. These numbers can be anything you want, but it would be neat to have them be actual metrics that you want to compare. Plot y1 vs x and display the plot. On the same graph, plot y2 vs x (after the line where you plot y1 vs x). Make the y1 line a pink line and the y2 line a gray line. Give both lines round markers. Give your graph a title of your choice for example “Two Lines on One Graph”, and label the x-axis” Amazing X-axis” and y-axis “Incredible Y-axis”. Give the graph a legend and put it in the lower right.
2. Implement K-Nearest Neighbor classifier on the iris dataset and analyze the performance using accuracy, while varying the number of neighbors (e.g. 1 – 250). Also print the neighbor(s) having the highest accuracy and those with the lowest accuracy.
3. Run the above provided KNN Algorithm for different random seed values from 1 to 10. Print all accuracies and then print the highest and the lowest.
4. Download the Dermatology data set from the UCI depositary, apply KNN and analyze the performance. Display the confusion matrix and discuss it. Use 10-Fold Cross Validation and random train/test split (70%, 30%).

(source: https://archive.ics.uci.edu/ml/datasets/Dermatology)

Note: There might be some missing/null values in the dataset. First clean the data and then work on the algorithm.