CO-1

1. Review of Python Programming

This colab is designed for you to practice and solve the activities that are based on the following concepts:

Python Lists

NumPy Arrays

Activities

Activity 1: Create a 3 X 3 Matrix

To create a 3x3 matrix with values ranging from 2 to 10.

For Example:

[[2 3 4]
 [5 6 7]
 [8 9 10]]
Follow the steps given below to achieve the desired result:

Step 1: Import numpy module.

Step 2: Use arange() function to create array of numbers from 2 to 10 and reshape() function to reshape your array into another array having 3 rows and 3 columns. Store this reshaped array in a variable x.

Step 3: Print variable x to get the output.

```
# Write your code here

import numpy as np
arr=np.arange(2,11).reshape(3,3)
print(arr)
```

OUTPUT:

```
[[ 2  3  4]
 [ 5  6  7]
 [ 8  9 10]]
```

Activity 2: Change Dimension of an Array and Convert the NumPy Array into a List

Write a program to change the dimension of an array (say my_arr =[1, 2, 3, 4, 5, 6, 7, 8, 9]) into a 3 X 3 (3 rows and 3 columns) array and convert this NumPy array into a list.

For Example:

Original array is [1 2 3 4 5 6 7 8 9]
Dimension is (9,)
Change array shape to (3, 3) -> 3 rows and 3 columns
[[1 2 3]
 [4 5 6]
 [7 8 9]]

The data type of the converted variable is list

```
# Write your code here
my_arr =np.arange(1,10).reshape(3,3)
print(arr)
l=list(arr)
print("datatype after covert tom list:\t",type(l))
```

```
[[ 2  3  4]
 [ 5  6  7]
 [ 8  9 10]]
datatype after covert tom list:  <class 'list'>
```

Activity 3: Find Square Root
Write a program to perform following task:

Print the square root of numbers in the list.

For Example:

list1 = [4, 16, 9, 1, 25]
[2 4 3 1 5]
Hint: Use np.sqrt() function.

```
# Write your solution here
list1 = [4, 16, 9, 1, 25]
l2=[]
for i in list1:
  l2.append(np.sqrt(i))
l2
```

```
[2.0, 4.0, 3.0, 1.0, 5.0]
```

Activity 4: Create and Update a Null NumPy Array

Create a null NumPy array of size 10 and update the sixth value to 11.

A null array is basically an array with all elements as 0.

Follow the steps given below to achieve the desired result:

- Step 1: Import the Numpy module as np.
- Step 2: Create a null array by passing the size i.e. 10 inside the np.zeros() function and store it in a variable null_arr.
- Step 3: Print the null array.
- Step 4: Now update the sixth element of the array by using list indexing method. As you need to update the sixth element, the index must be 5.
- Step 5: Print the updated array in the output.

```
# Write a program to create Null array of size 10 and update the
sixth value to 11.
z=np.zeros((10),dtype=int)
print(z)
```

```
z[5]=11
print('Array after update')
print(z)
```

```
[0 0 0 0 0 0 0 0 0 0]
Array after update
[ 0  0  0  0  0 11  0  0  0  0]
```

In the above program we have created a null array by using the np.zeros() function of the numpy module.

We have updated the 6th element to 11 by using a list indexing method.

Activity 5: Populate a Number List

Write a program that populates a list by numbers that lies in the range of 0 - 49 and also divisible by 5. Use List Comprehension method.

Output: [0, 5, 10, 15, 20, 25, 30, 35, 40, 45]

```
# Write a program to populate a number list divisible by 5 in a
range 0 - 49
l=[i for i in range(0,49) if i%5==0]
l
```

```
[0, 5, 10, 15, 20, 25, 30, 35, 40, 45]
```

Here, using list comprehension method we are running a for loop in range 0 - 49 and checking whether the number is divisible by 5 or not, using an if condition.

If number satisfies the condition, then that number is appended to number_list.

Activity 6: Convert List into Array

Write a program to convert a list of numeric values into a one-dimensional NumPy array.

For Example:

Input: mylist = [1.23, 23.32, 300, 16.37]

Data type of mylist = list

Output: numpy_array = [1.23, 23.32, 300, 16.37]

Data type of numpy_array = numpy.ndarray

```
# Program to convert a list into one dimensional NumPy array
mylist = [1.23, 23.32, 300, 16.37]
print("Data type of mylist is")
print(type(mylist))
numpy_array=np.array(mylist)
print("Data type of numpy_array is")
type(numpy_array)
```

```
Data type of mylist is
<class 'list'>
Data type of numpy_array is
numpy.ndarray
```

2.Data visualization using matplotlib vs seaborn

Activity 1: Create Customized Line plots.

Given the dataset of the average annual salary (in dollars) of developers of various programming languages. Create customized line plots to compare the salary variations Age-wise for Python developer with Javascript developer.

Link to the Dataset: https://raw.githubusercontent.com/CoreyMSchafer/code_snippets/master/Python/Matplotlib/10-Subplots/data.csv

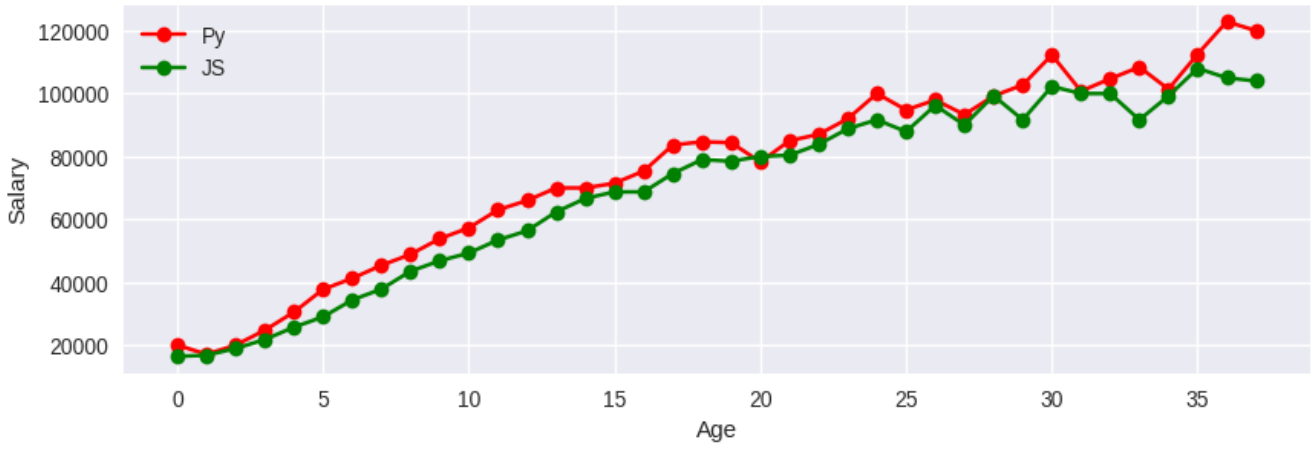
```
import pandas as pd
sal_df=pd.read_csv('https://raw.githubusercontent.com/CoreyMSchafer/
code_snippets/master/Python/Matplotlib/10-Subplots/data.csv')
sal_df.head()
```

	Age	All_Devs	Python	JavaScript
0	18	17784	20046	16446
1	19	16500	17100	16791
2	20	18012	20000	18942
3	21	20628	24744	21780
4	22	25206	30500	25704

```
# Step 3: Create a customised line plot for comparing the Age-wise annual
salary variations for Python developer with JavaScript developer.  Use the
'seaborn-dark' style
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
sal_df=pd.read_csv('https://raw.githubusercontent.com/CoreyMSchafer/
code_snippets/master/Python/Matplotlib/10-Subplots/data.csv')

plt.figure(figsize=(10,3))
plt.style.use('seaborn')
```

```
plt.ylabel('Salary')
plt.xlabel('Age')
plt.plot(sal_df['Python'],'r-o',label='Py')
plt.plot(sal_df['JavaScript'],'g-o',label='JS')
plt.legend()
plt.show()
```



Q: What can you conclude from the above comparison ?

A:Python developers earn more money

Activity 2.1: Create a Pandas DataFrame

Create a Pandas DataFrame by using the below link which has the dataset of Tips taken on the total bill amount in restaurants in the CSV format:

Dataset Link : https://raw.githubusercontent.com/jiss-github123/tips/main/tips.csv

Also, print the first five rows of the dataset.

```
r_df=pd.read_csv('https://raw.githubusercontent.com/jiss-github123/tips/
main/tips.csv')
r_df.head()
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Activity 2.2: Create a Gender wise Count plot

Create a gender wise count plot by using the values in the sex column.

```
# Gender wise count plot for the 'sex' values in the 'tip_df' DataFrame on
the x-axis.
plt.figure(figsize=(5,3))
plt.title("Gender Count")
sns.countplot(x='sex',data= r_df)
plt.show()
```

So according to the above count plot, the number of Female is less than the number of Male in the dataset.

Q : Which gender is recorded more in the dataset ?

A :Male

Activity 3: Histogram using hist() Function

Given a list of random age of 100 individuals in a range between 1 and 91. Write a code to visualize the values in the list using a histogram.

age_list = [1,1,2,3,3,5,7,8,9,10,

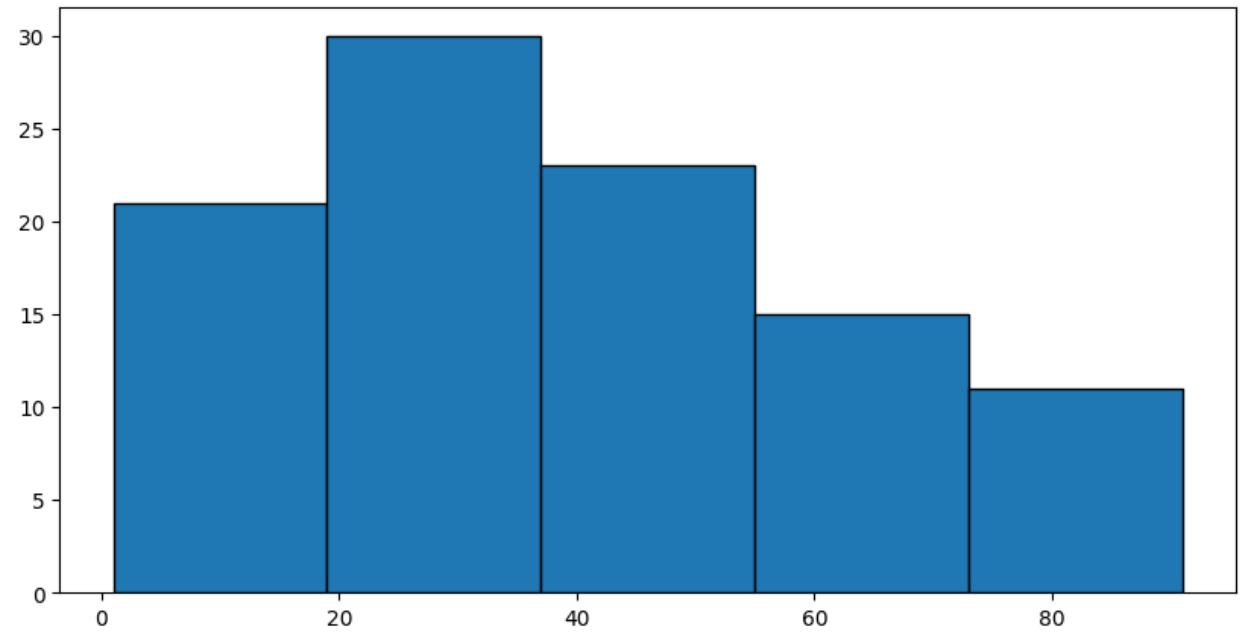
10,11,11,13,13,15,16,17,18,18,
18,19,20,21,21,23,24,24,25,25,
25,25,26,26,26,27,27,27,27,27,
29,30,30,31,33,34,34,34,35,36,
36,37,37,38,38,39,40,41,41,42,
43,44,45,45,46,47,48,48,49,50,
51,52,53,54,55,55,56,57,58,60,
61,63,64,65,66,68,70,71,72,74,
75,77,81,83,84,87,89,90,90,91
]

Steps to Follow:

1. Import the matplotlib.pyplot module.
2. Set the size of the plot using the figsize attribute of the figure() function.
3. Pass the age_list list inside the hist() function and set bins = 10.
4. Display the histogram using the show() function of the matplotlib.pyplot module.

```
# Import the 'matplotlib.pyplot' module.
age_list = [1,1,2,3,3,5,7,8,9,10,
     10,11,11,13,13,15,16,17,18,18,
     18,19,20,21,21,23,24,24,25,25,
     25,25,26,26,26,27,27,27,27,27,
     29,30,30,31,33,34,34,34,35,36,
     36,37,37,38,38,39,40,41,41,42,
     43,44,45,45,46,47,48,48,49,50,
     51,52,53,54,55,55,56,57,58,60,
     61,63,64,65,66,68,70,71,72,74,
     75,77,81,83,84,87,89,90,90,91
     ]
import matplotlib.pyplot as plt
# Set the size of the plot using the 'figsize' attribute of the 'figure()'
function.
plt.figure(figsize=(7,3))
# Pass the 'age_list' list inside the 'hist()' function and set 'bins =
10'.
```

```
plt.hist(age_list,bins=5,edgecolor='black')
# Display the histogram using the 'show()' function of the
'matplotlib.pyplot' module.
plt.show()
```



Activity 4: Lineplot using plot() Function

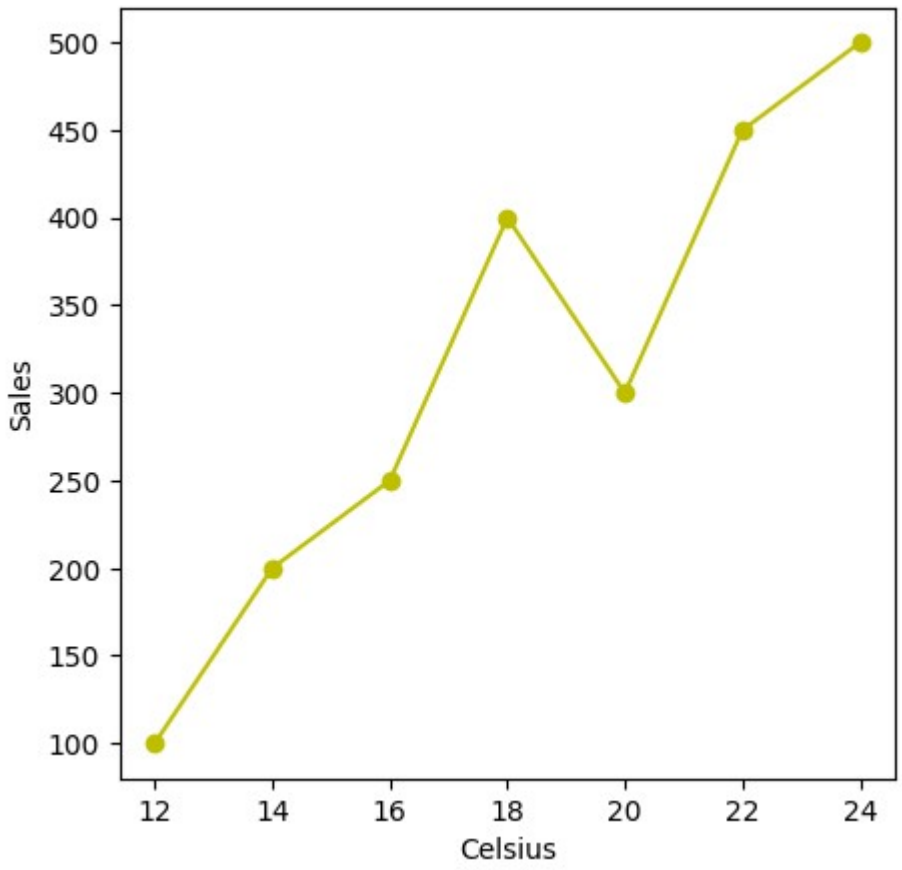
```
"""
Draw a line in a diagram from position (1, 3) to (2, 10) then to (6, 12)
and finally to position
(18, 20). (Mark each point with a beautiful green color and set line color
to red and line style
dotted)
"""
import matplotlib.pyplot as plt
import numpy as np
xpoints=np.array([1,2,6,18])
ypoints=np.array([3,10,12,20])
plt.figure(figsize=(7,3))
plt.plot(xpoints,ypoints,'r-o')
plt.show()
```

```
Draw a plot for the following data:

Temperature in degree Celsius ,  Sales

        12                       100
        14                       200
        16                       250
        18                       400
        20                       300
        22                       450
        24                       500
```

```
c=np.array([12,14,16,18,20,22,24])
s=np.array([100,200,250,400,300,450,500])
plt.figure(figsize=(5,5))
plt.xlabel('Celsius')
plt.ylabel('Sales')
plt.plot(c,s,'y-o')
plt.show()
```



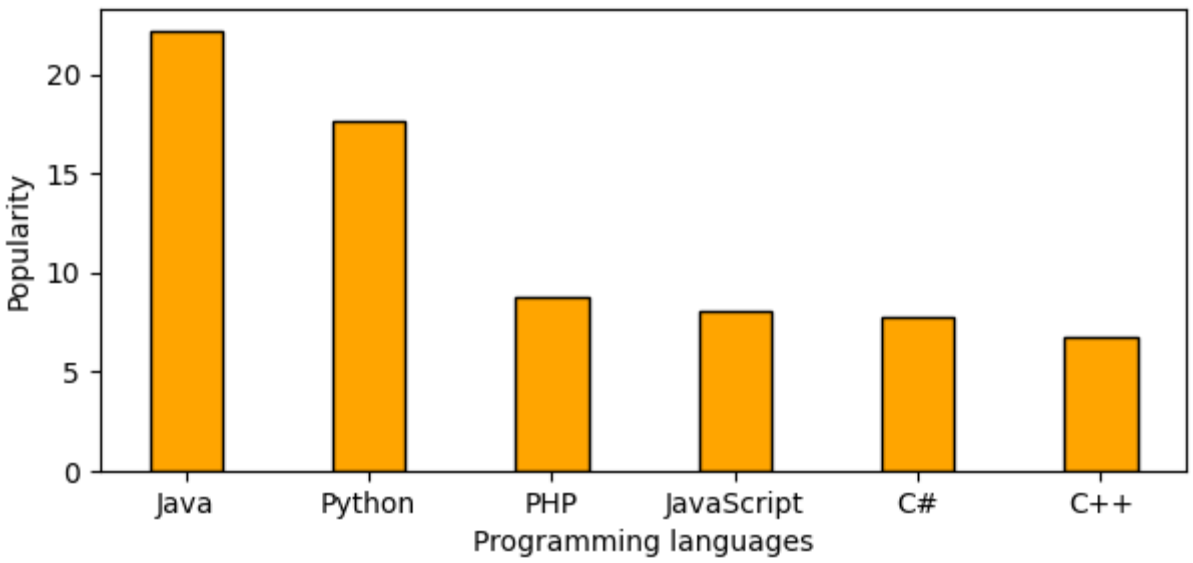
Activity 5: Bar graph using bar() Function

Consider the following data.

Programming languages:	Java	Python	PHP	JavaScript	C#	C++
Popularity	22.2	17.6	8.8	8	7.7	6.7

(i) Write a Python programming to display a bar chart of the popularity of programming Languages.

```
data={'Java':22.2,'Python':17.6,'PHP':8.8,'JavaScript':8,'C#':7.7,'C+
+':6.7}
y=data.values()
x=data.keys()
plt.figure(figsize=(7,3))
plt.ylabel('Popularity')
plt.xlabel('Programming languages')
plt.bar(x,y,width=0.4,color='orange',edgecolor='black')
plt.show()
```



Activity 6: Pie Chart using pie() Function

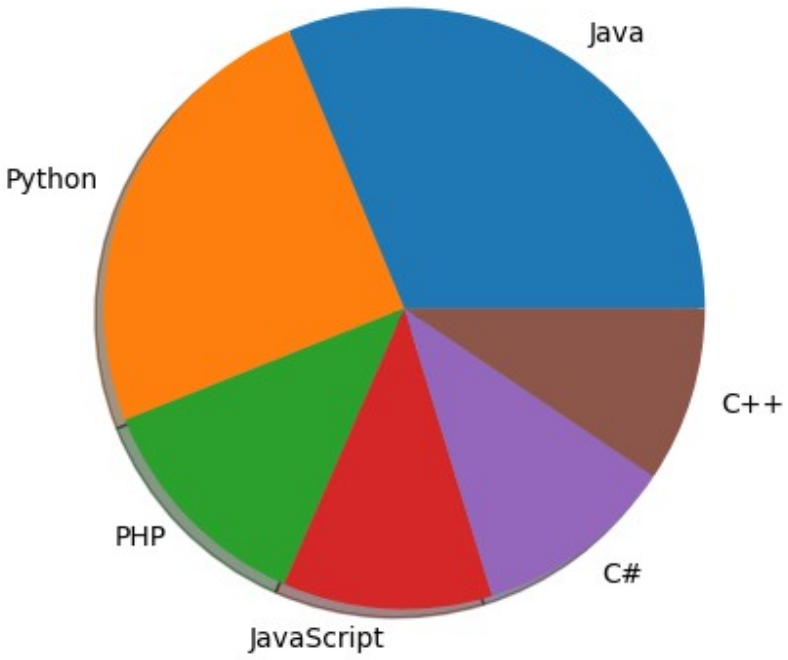
```
"""
Write a Python programming to create a pie chart
```

```
   of the popularity of programming Languages.

Programming languages: Java Python PHP JavaScript C# C++
Popularity           : 22.2 17.6   8.8 8          7.7 6.7

"""
data={'Java':22.2,'Python':17.6,'PHP':8.8,'JavaScript':8,'C#':7.7,'C+
+':6.7}
y=data.values()
x=data.keys()

plt.figure(figsize=(8,5))
plt.pie(y,labels=x,shadow=True)
plt.show
```



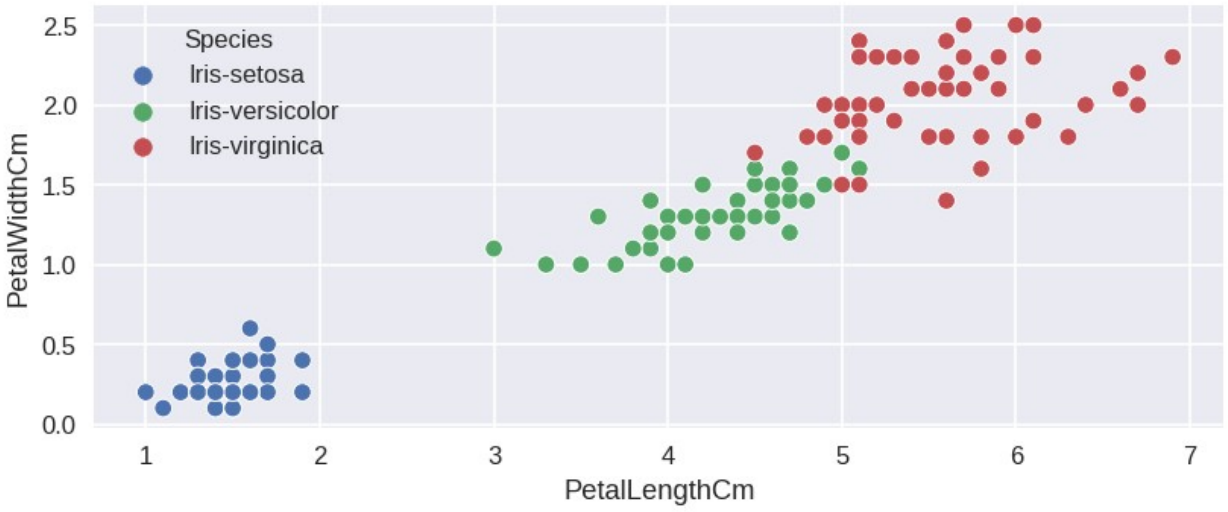
Activity 7: Scatter plot using scatterplot() Function

Create a scatter plot between the 'SepalLengthCm' & 'SepalWidthCm' columns of iris dataset.Differentiate between the data points of different classes using the 'hue' parameter.

```
df=pd.read_csv('https://raw.githubusercontent.com/Rohit-1311/
DataScienceS3/main/iris-dataset.csv')
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(8,3),dpi=120)
sns.scatterplot(data=df,x='PetalLengthCm',y='PetalWidthCm',hue='Species')
plt.show()
```



3.NumpyArray

```
# program to create an array of all the even integers from 20 to
80
import numpy as np
x=np.arange(20,80,2)
x
```

```
array([20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52,
       54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78])
```

```
#Write a program to create a numpy array and perform element-wise
comparison (greater, greater_equal, less than, lesser_equal).
x=np.array([0,2,3,8,5])
y=np.array([1,2,3,7,6])
print(x)
print(y)
print(np.greater(x,y))
print(np.greater_equal(x,y))
print(np.less(x,y))
print(np.less_equal(x,y))
```

```
[0 2 3 8 5]
[1 2 3 7 6]
[False False False  True False]
[False  True  True  True False]
[ True False False False  True]
[ True  True  True False  True]
```

```
#program to multiply two given arrays of same size
a1=np.arange(1,7).reshape(2,3)
a2=np.arange(4,10).reshape(2,3)
print('\na1=>\n',a1)
print('\na2=>\n',a2)
```

```
print('\nProduct\n',np.multiply(a1,a2))
```

```
a1=>
 [[1 2 3]
 [4 5 6]]

a2=>
 [[4 5 6]
 [7 8 9]]

Product
 [[ 4 10 18]
 [28 40 54]]
```

```
#Program to save a given array to a text file and load it
x=np.arange(0,9).reshape(3,3)
print('Array is=>\n',x)
np.savetxt('array.txt',x,fmt="%d")
print('\n After loading the array')
print(np.loadtxt('array.txt'))
```

```
Array is=>
 [[0 1 2]
 [3 4 5]
 [6 7 8]]

 After loading the array
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
```

4.Programs to handle data using pandas

```
#1. Write a python program to implement List-to-Series Conversion
import pandas as pd
name=['adarsh','jissmon','vishnu','dethan']
s=pd.Series(name)
print(s)
```

```
0     adarsh
1    jissmon
2     vishnu
3     dethan
dtype: object
```

```
#2. Write a python program to Generate the series of dates from 1st
September, 2023 to 25th September, 2023 (both inclusive).
import pandas as pd
date=pd.Series(pd.date_range('2023-09-1','2023-09-25',freq='D'))
```

```
0    2023-09-01
1    2023-09-02
2    2023-09-03
3    2023-09-04
4    2023-09-05
5    2023-09-06
6    2023-09-07
7    2023-09-08
8    2023-09-09
9    2023-09-10
10   2023-09-11
11   2023-09-12
12   2023-09-13
13   2023-09-14
14   2023-09-15
15   2023-09-16
16   2023-09-17
17   2023-09-18
18   2023-09-19
19   2023-09-20
20   2023-09-21
21   2023-09-22
22   2023-09-23
23   2023-09-24
24   2023-09-25
dtype: datetime64[ns]
date
```

```
"""
3. Given a dictionary, convert it into corresponding dataframe and display
it.

"""
import pandas as pd
data={'name':['adarsh','vishnu','jissmon','dethan','glodin','rohit'],
       'age':[25,24,27,20,23,22]}
df=pd.DataFrame(data)
df
```

	name	age
0	adarsh	25
1	vishnu	24
2	jissmon	27
3	dethan	20
4	glodin	23
5	rohit	22

```
"""
4. Given a dataframe, select first 2 rows and output them.

"""
df
df[:2]
```

	name	age
0	adarsh	25
1	vishnu	24

```
# 5. Given is a dataframe showing name, occupation,salary of people. Find
the average salary per occupation.
import pandas as pd
emp={'name':['Adarsh','Vishnu','Dethan','Rohit','Jissmon','Glodin'],
      "Occupation":
['Manager','Manager','Programmer','Programmer','Analyst','Analyst'],
      "Salary" : [20000,25000,28000,27000,30000,35000]}
df = pd.DataFrame(emp)
print(df)
avg_sal = df.groupby('Occupation')['Salary'].mean()
print("\nThe average salary per occupation is: ")
print(avg_sal)
```

```
      name  Occupation  Salary
0   Adarsh     Manager   20000
1   Vishnu     Manager   25000
2   Dethan  Programmer   28000
3    Rohit  Programmer   27000
4  Jissmon     Analyst   30000
5   Glodin     Analyst   35000

The average salary per occupation is:
Occupation
Analyst       32500.0
Manager       22500.0
Programmer    27500.0
Name: Salary, dtype: float64
```

CO2

3.1 k-NN classification using any standard dataset

Aim: Program to implement k-NN classification using any standard dataset available in the public domain and find the accuracy of the algorithm

Algorithm:

The class of an unknown instance is computed using the following steps:

1. The distance between the unknown instance and all other training instances is computed.
2. The k nearest neighbors are identified.
3. The class labels of the k nearest neighbors are used to determine the class label of the unknown instance by using techniques like majority voting.

```
# KNN implementation using iris dataset
#import modules
from sklearn.metrics import accuracy_score,classification_report
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd

#Load Dataset
df=pd.read_csv('https://raw.githubusercontent.com/Rohit-1311/
DataScienceS3/main/iris-dataset.csv')
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
#Implement KNN model,check accuracy
x=df[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']]
x
y=df['Species']
y
```

```
0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
             ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 150, dtype: object
```

```
x_trai
n,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=1)
from sklearn import metrics
knn=KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train,y_train)
y_pred=knn.predict(x_test)
a=metrics.accuracy_score(y_test,y_pred)
print('Accuracy=>',a)
```

```
Accuracy=> 1.0
```

```
sample=[[2,2,2,2]]
predicton=knn.predict(sample)
print("The Species is=>",predicton)

print(classification_report(y_test,y_pred))
```

```
The Species is=> ['Iris-setosa']
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        11
Iris-versicolor       1.00      1.00      1.00        13
 Iris-virginica       1.00      1.00      1.00         6

       accuracy                           1.00        30
      macro avg       1.00      1.00      1.00        30
   weighted avg       1.00      1.00      1.00        30
```

3.2 k-NN classification using any standard dataset

Problem Statement

Nowadays, social media advertising is one of the popular forms of advertising. Advertisers can utilise user's demographic information and target their ads accordingly. You are given a dataset having the following attributes:

Field	Description
UserID	Unique ID
Gender	Male or Female
Age	Age of a person
EstimatedSalary	Salary of a person
Purchased	'0' or '1'. '0' means not purchased and '1' means purchased.

Source: https://raw.githubusercontent.com/Rohit-1311/DataScienceS3/main/social-network-ads.csv

Implement kNN Classifier to determine whether a user will purchase a particular product displayed on a social network ad or not.

List of Activities

Activity 1: Import Modules and Read Data

Activity 2: Perform Train-Test Split

Activity 3: Determine the Optimal Value of k

Activity 4: Build kNN Classifier Model

Activity 1: Import Modules and Read Data

Import the necessary Python packages.

Read the data from a CSV file to create a Pandas DataFrame.

Dataset--> social-network-ads.csv

Also, print the first five rows of the dataset. Check for null values and treat them accordingly.

```
# Import all the necessary packages
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd

# Load the dataset
s_df=pd.read_csv('https://raw.githubusercontent.com/Rohit-1311/
DataScienceS3/main/social-network-ads.csv')
# Print first five rows using head() function
s_df.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

```
# Check if there are any null values. If any column has null values, treat
them accordingly
s_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

```
s_df.isnull().sum()
```

```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

: Are there any missing or null values in the dataset?

A: No.

Activity 2: Perform Train-Test Split

In this dataset, Purchased is the target variable and all other columns other than Purchased are feature variables.

Create two separate DataFrames, one containing the feature variables and the other containing the target variable. Also, drop the User ID column from the features DataFrame as it is of no use.

```
# Split the dataset into dependent and independent features
x=s_df[['Gender','Age','EstimatedSalary']]
y=s_df['Purchased']
```

Print the summary of features DataFrame to determine the data type of each feature variable.

```
# Use 'info()' function with the features DataFrame.
x.info()
y.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Gender           400 non-null    object
 1   Age              400 non-null    int64
 2   EstimatedSalary  400 non-null    int64
dtypes: int64(2), object(1)
memory usage: 9.5+ KB
<class 'pandas.core.series.Series'>
RangeIndex: 400 entries, 0 to 399
Series name: Purchased
Non-Null Count  Dtype
--------------  -----
400 non-null    int64
dtypes: int64(1)
memory usage: 3.2 KB
```

Convert categorical Gender feature into numerical by calling the get_dummies() function of pandas module and passing features DataFrame as input.

```
# Use 'get_dummies()' function to convert each categorical column in a
DataFrame to numerical.
x=pd.get_dummies(x)
x
```

	Age	EstimatedSalary	Gender_Female	Gender_Male
0	19	19000	0	1
1	35	20000	0	1
2	26	43000	1	0
3	27	57000	1	0
4	19	76000	0	1
...	...	...	...	...
395	46	41000	1	0
396	51	23000	0	1
397	50	20000	1	0
398	36	33000	0	1
399	49	36000	1	0

400 rows × 4 columns

Split the dataset into train set and test set such that the train set contains 70% of the instances and the remaining instances will become the test set.

```
# Split the DataFrame into the train and test sets.
# Perform train-test split using 'train_test_split' function.
x_trai
n,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=45
)
# Print the shape of the train and test sets.
print('x_train=>',x_train.shape)
print('x_test=>',x_test.shape)
print('y_train=>',y_train.shape)
print('y_test=>',y_test.shape)
```

```
x_train=> (280, 4)
x_test=> (120, 4)
y_train=> (280,)
y_test=> (120,)
```

After this activity, you must obtain train and test sets so that they can be used for training and testing the kNN Classifier.

Activity 4: Build kNN Classifier Model
Deploy the kNN Classifier model for the optimal value of k using the steps given below:

1. Import the KNeighborsClassifier class from the sklearn.neighbors module (if not imported yet).
2. Create an object of KNeighborsClassifier and pass the optimal k value as 5 to its constructor.
3. Call the fit() function using the classifier object and pass the train set as inputs to this function.
4. Perform prediction for train and test sets using the predict() function.
5. Also, determine the accuracy score of the train and test sets using the score() function.

```
# Train kNN Classifier model
s_knn=KNeighborsClassifier(n_neighbors=5)
s_knn.fit(x_train,y_train)
# Perform prediction using 'predict()' function.
pred=s_knn.predict(x_test)
sample=[[19 ,57000,1,0]]
r=s_knn.predict(sample)
print(r)
# Call the 'score()' function to check the accuracy score of the train set
and test set.
accuracy=accuracy_score(y_test,pred)
print("Accuracy=>",accuracy)
```

```
[0]
Accuracy=> 0.7416666666666667
```

Print the classification report to get an in-depth overview of the classifier performance using the classification_report() function of sklearn.metrics module

```
# Display the precision, recall, and f1-score values.
from sklearn.metrics import classification_report
print(classification_report(y_test,pred))
```

```
                  precision    recall  f1-score   support

               0       0.75      0.88      0.81        74
               1       0.73      0.52      0.61        46

        accuracy                           0.74       120
       macro avg       0.74      0.70      0.71       120
    weighted avg       0.74      0.74      0.73       120
```

3.3. k-NN classification using any standard dataset

Goal of the Project

This project is designed for you to practice and solve the activities that are based on the concept: kNN.

Problem Statement

As an owner of a startup, you wish to forecast the sales of your product to plan how much money should be spent on advertisements. This is because the sale of a product is usually proportional to the money spent on advertisements. To analyze this, you are given a dataset having the following attributes:

Attribute	Description
TV	TV advertising budget in thousands of dollars.
Radio	Radio advertising budget in thousands of dollars.
Newspaper	Newspaper advertising budget in thousands of dollars.
Sales	Product Sales in thousands of dollars.

Source: https://raw.githubusercontent.com/Rohit-1311/DataScienceS3/main/advertising.csv

Predict the impact of TV advertising on your product sales by using kNN regression and evaluate the accuracy of the model.

List of Activities

Activity 1: Import Modules and Read Data

Activity 2: Perform Train-Test Split

Activity 3: Build kNN Regressor Model

Activity 1: Import Modules and Read Data

Create a Pandas DataFrame for Advertising-Sales dataset using the below link. This dataset contains information about the money spent on the TV, radio, and newspaper advertisement (in thousand dollars) and their generated sales (in thousand units). The dataset consists of examples that are divided by 1000.

Dataset : advertising.csv

Also, print the first five rows of the dataset. Check for null values and treat them accordingly.

```
# Import modules
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import pandas as pd

# Load the dataset
df=pd.read_csv('https://raw.githubusercontent.com/Rohit-1311/
DataScienceS3/main/advertising.csv')
# Print first five rows using head() function
df.head()
```

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9

```
# Check if there are any null values. If any column has null
values, treat them accordingly.
df.isnull().sum()
```

```
TV          0
Radio       0
Newspaper   0
Sales       0
dtype: int64
```

Q: Are there any missing or null values in the dataset?

A:No

Activity 2: Perform Train-Test Split

In this dataset, Sales is the target variable and all other columns other than Sales are feature variables.

Create two separate DataFrames, one containing the feature variables and the other containing the target variable.

```
# Split the dataset into dependent and independent features
x=df[['TV','Radio','Newspaper']]
y=df['Sales']
```

Normalize all the feature variables using the StandardScaler technique so that all the features have mean 0 and the same variance before applying kNN.

```
# Normalise the feature variables using 'StandardScaler'.
# Import 'StandardScaler' from 'sklearn.preprocessing' module.
from sklearn.preprocessing import StandardScaler
# Create an object of 'StandardScaler' and call 'fit_transform()'
function by passing feature variables.
ob=StandardScaler()
s_x=ob.fit_transform(x)
# Convert the scaled features array obtained from
'fit_transform()' function into a DataFrame.
s_x=pd.DataFrame(s_x)
s_x.columns=x.columns
s_x.head()
```

	TV	Radio	Newspaper
0	0.969852	0.981522	1.778945
1	-1.197376	1.082808	0.669579
2	-1.516155	1.528463	1.783549
3	0.052050	1.217855	1.286405
4	0.394182	-0.841614	1.281802

Split the dataset into a train set and test set such that the train set contains 70% of the instances and the remaining instances will become the test set.

```
# Split the DataFrame into the train and test sets.
# Perform train-test split using 'train_test_split' function.
x_trai
n,x_test,y_train,y_test=train_test_split(s_x,y,train_size=0.7,ran
dom_state=1)
# Print the shape of train and test sets.
print(x_test.shape)
print(x_train.shape)
print(y_test.shape)
print(y_train.shape)
```

```
(60, 3)
(140, 3)
(60,)
(140,)
```

After this activity, you must obtain train and test sets so that they can be used for training and testing the kNN regressor model.

Activity 3: Build kNN Regressor Model

Deploy the kNN regressor model for the optimal value of k using the steps given below:

1. Import the KNeighborsRegressor class from the sklearn.neighbors module (if not imported yet).
2. Create an object of KNeighborsRegressor and pass the optimal k value 2 as input to its constructor.
3. Call the fit() function using the regressor object and pass the train set as inputs to this function.
4. Perform prediction for train and test sets using the predict() function.
5. Also, determine the accuracy score of the train and test sets using the score() function.

```
# Train kNN regressor model
from sklearn.neighbors import KNeighborsRegressor
knn=KNeighborsRegressor(n_neighbors=2)
knn.fit(x_train,y_train)
# Perform prediction using 'predict()' function.
pred=knn.predict(x_test)
# Call the 'score()' function to check the accuracy score of the
train set and test set.
acc=knn.score(x_train,y_train)
print('Accuracy=>',acc)
sample=[[0.969852,1.528463,0.669579]]
r=knn.predict(sample)
print(r)
```

```
Accuracy=> 0.9635754264321423
[24.]
```

```
# Train kNN regressor model
from sklearn.neighbors import KNeighborsRegressor
knn=KNeighborsRegressor(n_neighbors=2)
```

```
knn.fit(x_train,y_train)
# Perform prediction using 'predict()' function.
pred=knn.predict(x_test)
acc=knn.score(x_test,y_test)

sample=[[0.969852,1.528463,0.669579]]
r=knn.predict(sample)
print(r)
print('Test Score=>',acc)
print('Train Score=>',knn.score(x_train,y_train))
# Call the 'score()' function to check the accuracy score of the
train set and test set.
```

```
[24.]
Test Score=> 0.9243088142911436
Train Score=> 0.9635754264321423
```

Q: Write down the train and test set accuracy scores for the kNN regressor?

A:Test Score=> 0.9243088142911436

Train Score=> 0.9635754264321423

After this activity, you must obtain a kNN regressor model using the sklearn module for predicting total sales based on advertising budgets.

4.Naïve Bayes Algorithm

Aim: Program to implement Naïve Bayes Algorithm using any standard dataset available in the public domain and find the accuracy of the algorithm

Short notes: Naive Bayes
Bayes' Theorem provides a way that we can calculate the probability of a piece of data belonging to a given class, given our prior knowledge. Bayes' Theorem is stated as:

P(class|data) = (P(data|class) * P(class)) / P(data)
Where P(class|data) is the probability of class given the provided data.

We are using Iris Dataset. The Iris Flower Dataset involves predicting the flower species given measurements of iris flowers.

It is a multiclass classification problem. The number of observations for each class is balanced. There are 150 observations with 4 input variables and 1 output variable. The variable names are as follows:

Sepal length in cm.

Sepal width in cm.

Petal length in cm.

Petal width in cm.

Class.

Algorithm:

Step 1: Separate By Class.

Step 2: Summarize Dataset.

Step 3: Summarize Data By Class.

Step 4: Gaussian Probability Density Function.

Step 5: Class Probabilities.

```
#Import Modules
from  sklearn.model_selection import train_test_split
from sklearn.metrics import
accuracy_score,confusion_matrix,classification_report
import pandas as pd
#Load iris dataset & do train_test_split
df=pd.read_csv('https://raw.githubusercontent.com/Rohit-1311/
DataScienceS3/main/iris-dataset.csv')
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
X=d
f[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']
]
y=df['Species']
#Feature Scaling
from sklearn.preprocessing import StandardScaler
ob=StandardScaler()
S_X=ob.fit_transform(X)
S_X=pd.DataFrame(S_X)
S_X.columns=X.columns
S_X.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	-0.900681	1.032057	-1.341272	-1.312977
1	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.385353	0.337848	-1.398138	-1.312977
3	-1.506521	0.106445	-1.284407	-1.312977
4	-1.021849	1.263460	-1.341272	-1.312977

```
x_trai
n,x_test,y_train,y_test=train_test_split(S_X,y,stratify=y,train_s
ize=0.7,random_state=10)
x_test.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
9	-1.143017	0.106445	-1.284407	-1.444450
125	1.643844	0.337848	1.274550	0.790591
15	-0.173674	3.114684	-1.284407	-1.050031
117	2.249683	1.726266	1.672610	1.316483
55	-0.173674	-0.587764	0.421564	0.133226

In this step, we introduce the class GaussianNB that is used from the sklearn.naive_bayes library. Here, we have used a Gaussian model, there are several other models such as Bernoulli, Categorical and Multinomial. Here, we assign the GaussianNB class to the variable classifier and fit the X_train and y_train values to it for training purpose.

```
#Implement Naive Bayes
from sklearn.naive_bayes import GaussianNB
classfier=GaussianNB()
classfier.fit(x_train,y_train)
#Predict the values for test data
pred=classfier.predict(x_test)
```

```
# Display accuracy score & display confusion matrix &
classification report
print('Accurracy=>',accuracy_score(y_test,pred))
print('Classifiaction Report=>\
n',classification_report(y_test,pred))
print('Confusion Metrix=>\n',confusion_matrix(y_test,pred))
```

```
Accurracy=> 1.0
Classifiaction Report=>
                  precision    recall  f1-score   support

     Iris-setosa       1.00      1.00      1.00        15
 Iris-versicolor       1.00      1.00      1.00        15
  Iris-virginica       1.00      1.00      1.00        15

        accuracy                           1.00        45
       macro avg       1.00      1.00      1.00        45
    weighted avg       1.00      1.00      1.00        45

Confusion Metrix=>
 [[15  0  0]
 [ 0 15  0]
 [ 0  0 15]]
```

From the above confusion matrix, we infer that, out of 45 test set data, 45 were correctly classified and only 1 was incorrectly classified. This gives us a high accuracy of 100%.

5.1.Forecast Sales Using Simple Linear Regression

Problem Statement

As an owner of a startup, you wish to forecast the sales of your product to plan how much money should be spent on advertisements. This is because the sale of a product is usually proportional to the money spent on advertisements.

Predict the impact of TV advertising on your product sales by performing simple linear regression analysis.

List of Activities

Activity 1: Analysing the dataset

Activity 2: Train-Test split

Activity 3: Model training

Activity 4: Plotting the best fit line

Activity 5: Model prediction

Activity 1: Analysing the Dataset

Create a Pandas DataFrame for Advertising-Sales dataset using the below link. This dataset contains information about the money spent on the TV, radio and newspaper advertisement (in thousand dollars) and their generated sales (in thousand units). The dataset consists of examples that are divided by 1000.

Dataset Link:
https://raw.githubusercontent.com/Rohit-1311/DataScienceS3/main/advertising.csv

Also, print the first five rows of the dataset. Check for null values and treat them accordingly.

```
# Import modules
import numpy as np
import pandas as pd
# Load the dataset
```

```
df=pd.read_csv('https://raw.githubusercontent.com/Rohit-1311/DataScienceS3/main/advertising.csv)
# Print first five rows using head() function
df.head(5)
```

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	12.0
3	151.5	41.3	58.5	16.5
4	180.8	10.8	58.4	17.9

```
# Check if there are any null values. If any column has null values, treat them accordingly
df.isnull().sum()
```

```
TV           0
Radio        0
Newspaper    0
Sales        0
dtype: int64
```

Activity 2: Train-Test Split

For simple linear regression, consider only the effect of TV ads on sales. Thus, TV is the feature variable and Sales is the target variable.

Split the dataset into training set and test set such that the training set contains 67% of the instances and the remaining instances will become the test set.

```
# Split the DataFrame into the training and test sets.
from sklearn.model_selection import train_test_split
x=df['TV']
```

```
y=df['Sales']
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.75,random_state=10)
print('x_train=>',x_train.shape)
print('x_test=>',x_test.shape)
```

```
x_train=> (150,)
x_test=> (50,)
```

Activity 3: Model Training

Train the simple regression model using **training data** to obtain the best fit line

y=mx+c

. For this, perform the following tasks:

1. Create following two functions:
   - A function errors_product() that calculates the errors for the feature and target variables i.e.

$$(x_i - \bar{x})(y_i - \bar{y})$$

   - A function squared_errors() that calculates the squared errors for the feature variable only i.e.

$$(x_i - \bar{x})^2$$

2.Calculate the slope and intercept values for the best fit line by applying the following formulae:

$$slope \Rightarrow m = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} = \frac{errors\_product().sum()}{squared\_errors().sum()}$$

$$intercept \Rightarrow c = \bar{y} - m\bar{x}$$

```
# Create the 'errors_product()' and 'squared_errors()' function.
from scipy import stats
# Calculate the slope and intercept values for the best fit line.
slope,intercept,r,p,std_err=stats.linregress(x,y)
print('\n slope=>',slope)
print('\n Intercept',intercept)
```

```
slope=> 0.05546477046955879

Intercept 6.974821488229903
```

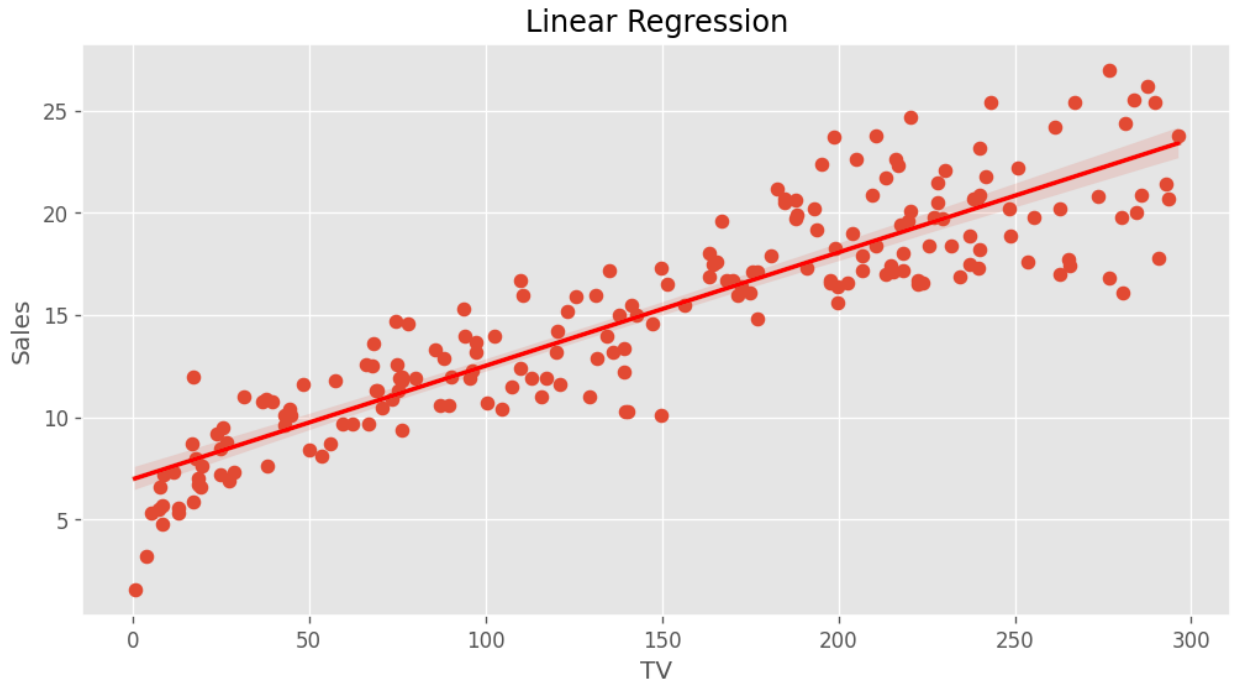
Q: What is the equation obtained for the best fit line of this model?

A:y=slope*x+intercept

Activity 4: Plotting the Best Fit Line

After obtaining the slope and intercept values for the best fit line, plot this line along with the scatter plot to see how well it fits the points.

```
#  Plot the regression line in the scatter plot between Sales and TV advertisment values.
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('ggplot')
plt.figure(figsize=(7,5),dpi=120)
plt.title('Linear Rigression',color='black')
sns.regplot(x='TV',y='Sales',data=df,scatter_kws={"color": "red"}, line_kws={"color": "red"})
plt.show()
```



Activity 5: Model Prediction

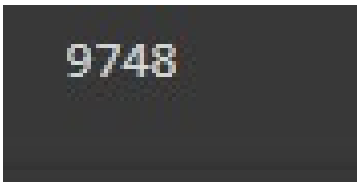
For the TV advertising of $50,000, what is prediction for Sales? In order to predict this value, perform the following task:

- Based on the regression line, create a function sales_predicted() which takes a budget to be used for TV advertising as an input and returns the corresponding units of Sales.
- Call the function sales_predicted() and pass the amount spent on TV advertising.

Note: To predict the sales for TV advertising of $50,000, pass 50 as parameter to sales_predicted() function as the original data of this dataset consists of examples that are divided by 1000. Also, the value obtained after calling sales_predicted(50) must be multiplied by 1000 to obtain the predicted units of sales.

```
#Create a function which takes TV advertisement value as an input and returns the sales.
def sales_predict(tv):
  return slope*tv+intercept

# Calculating sales value against $50,000 spent in TV ads
sales=sales_predict(50)
round(sales*1000)
```



Q: If you are planning to invest $50,000 dollars in TV advertising, how many unit of sales can be predicted according to this simple linear regression model?

A:9748

5.2.Simple Linear Regression

Problem Statement

The most important factor for an Insurance Company is to determine what premium charges must be paid by an individual. The charges depend on various factors like age, gender, income, etc.

Build a model that is capable of predicting the insurance charges a person has to pay depending on his/her age using simple linear regression. Also, evaluate the accuracy of your model by calculating the value of error metrics such as R-squared, MSE, RMSE, and MAE.

List of Activities

Activity 1: Analyzing the Dataset

Activity 2: Train-Test Split

Activity 3: Model Training

Activity 4: Model Prediction and Evaluation

Activity 1: Analysing the Dataset

- Create a Pandas DataFrame for Insurance dataset using the below link. This dataset consists of following columns:

Field	Description
age	Age of primary beneficiary
sex	Insurance contractor gender, female or male
bmi	Body mass index
children	Number of children covered by health insurance/number of dependents
region	Beneficiary's residential area in the US, northeast, southeast, southwest, northwest
charges	Individual medical costs billed by health insurance

Source: https://www.kaggle.com/bmarco/health-insurance-data

Dataset Link: insurance_dataset.csv

- Print the first five rows of the dataset. Check for null values and treat them accordingly.

- Create a regression plot with age on X-axis and charges on Y-axis to identify the relationship between these two attributes.

```
# Import modules
import pandas as pd
import numpy as np

# Load the dataset
df=pd.read_csv('https://raw.githubusercontent.com/Rohit-1311/DataScienceS3/main/
insurance_dataset.csv')
# Print first five rows using head() function
df.head(5)
```

	age	sex	bmi	children	region	charges
0	18	male	33.770	1	southeast	1725.55230
1	28	male	33.000	3	southeast	4449.46200
2	33	male	22.705	0	northwest	21984.47061
3	32	male	28.880	0	northwest	3866.85520
4	31	female	25.740	0	southeast	3756.62160

```
# Check if there are any null values. If any column has null values, treat them accordingly
df.isnull().sum()
```

```
age         0
sex         0
bmi         0
children    0
region      0
charges     0
dtype: int64
```

```
# Create a regression plot between 'age' and 'charges'
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('ggplot')
plt.figure(figsize=(7,3),dpi=120)
plt.title('Regression Line')
sns.regplot(x=df['age'],y=df['charges'],scatter_kws={'color':'black'},line_kws={'color':'red'})
plt.show()
```

Activity 2: Train-Test Split

We have to determine the effect of age on insurance charges. Thus, age is the feature variable and charges is the target variable.

Split the dataset into training set and test set such that the training set contains 67% of the instances and the remaining instances will become the test set.


```
# Split the DataFrame into the training and test sets.
from sklearn.model_selection import train_test_split
x=df['age']
y=df['charges']
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.67,random_state=15)
```

Activity 3: Model Training

Implement simple linear regression using sklearn module in the following way:

1. Reshape the feature and the target variable arrays into two-dimensional arrays by using reshape(-1, 1) function of numpy module.
2. Deploy the model by importing the LinearRegression class and create an object of this class.
3. Call the fit() function on the LinearRegression object and print the slope and intercept values of the best fit line.

```
# 1. Create two-dimensional NumPy arrays for the feature and target variables.
# Print the shape or dimensions of these reshaped arrays
x_train_reshaped=x_train.values.reshape(-1,1)
x_test_reshaped=x_test.values.reshape(-1,1)
y_train_reshaped=y_train.values.reshape(-1,1)
```

```
y_test_reshaped=y_test.values.reshape(-1,1)
```

```
# 2. Deploy linear regression model using the 'sklearn.linear_model' module.
from sklearn.linear_model import LinearRegression
# Create an object of the 'LinearRegression' class.
lin_reg=LinearRegression()
# 3. Call the 'fit()' function
lin_reg.fit(x_train_reshaped,y_train_reshaped)
# Print the slope and intercept values
print("slope=>",lin_reg.coef_)
print("Intercept=>",lin_reg.intercept_)
```

```
slope=> [[264.32416958]]
Intercept=> [-2003.06824466]
```

Activity 4: Model Prediction and Evaluation

Predict the values for both training and test sets by calling the predict() function on the LinearRegression object. Also, calculate the R2 , MSE, RMSE and MAE values to evaluate the accuracy of your model.

```
# Predict the target variable values for both training set and test set
pred=lin_reg.predict(x_test_reshaped)
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
# Call 'r2_score', 'mean_squared_error' & 'mean_absolute_error' functions of the 'sklearn'
module. Calculate RMSE value by taking the square root of MSE.
# Print these values for both training set and test set
print('\nTraining Set\n')
print('\nR2_score=>',r2_score(x_test_reshaped,pred))
print('\nMSE=>',mean_squared_error(x_test_reshaped,pred))
print('\nRMSE=>',np.sqrt(mean_squared_error(x_test_reshaped,pred)))
print('\nMAE=>',mean_absolute_error(x_test_reshaped,pred))
```

```
print('-'*50)
print('\nTraining Set\n')
print('\nR2_score=>',r2_score(y_test_reshaped,pred))
print('\nMSE=>',mean_squared_error(y_test_reshaped,pred))
print('\nRMSE=>',np.sqrt(mean_squared_error(y_test_reshaped,pred)))
print('\nMAE=>',mean_absolute_error(y_test_reshaped,pred))
```

```
Training Set


R2_score=> -380461.9274752225

MSE=> 81581735.99840507

RMSE=> 9032.26084645506

MAE=> 8167.8278054044895
--------------------------------------------------

Training Set


R2_score=> 0.43366144225426495

MSE=> 20808428.221594278

RMSE=> 4561.625611730348

MAE=> 2483.8728245953957
```

6.Multiple Linear Regression

Goal of the Project

This project is designed for you to practice and solve the activities that are based on the concepts covered in the following lessons:

1. Multiple linear regression - Introduction

Problem Statement

A real estate company wishes to analyse the prices of properties based on various factors such as area, number of rooms, bathrooms, bedrooms, etc. Create a multiple linear regression model which is capable of predicting the sale price of houses based on multiple factors and evaluate the accuracy of this model.

List of Activities

Activity 1: Analysing the Dataset

Activity 2: Data Preparation

Activity 3: Train-Test Split

Activity 4: Model Training

Activity 5: Model Prediction and Evaluation

Activity 1: Analysing the Dataset

Create a Pandas DataFrame for Housing dataset using the below link. This dataset consists of following columns:

Field	Description
price	Sale price of a house in INR
area	Total size of a property in square feet
bedrooms	Number of bedrooms
bathrooms	Number of bathrooms
storeys	Number of storeys excluding basement
mainroad	yes, if the house faces a main road
livingroom	yes, if the house has a separate living room or a drawing room for guests
basement	yes, if the house has a basement
hotwaterheating	yes, if the house uses gas for hot water heating
airconditioning	yes, if there is central air conditioning
parking	number of cars that can be parked
prefarea	yes, if the house is located in the preferred neighbourhood of the city

Dataset Link: house-prices.csv

- Print the first five rows of the dataset. Check for null values and treat them accordingly.

```
# Import modules

import pandas as pd

import numpy as np

# Load the dataset

df=pd.read_csv('https://raw.githubusercontent.com/Rohit-1311/DataScienceS3/main/house-prices.csv')

# Print first five rows using head() function

df.head(5)
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	yes	no	no	no	yes	2	yes	furnished
1	12250000	8960	4	4	4	yes	no	no	no	yes	3	no	furnished
2	12250000	9960	3	2	2	yes	no	yes	no	no	2	yes	semi-furnished
3	12215000	7500	4	2	2	yes	no	yes	no	yes	3	yes	furnished
4	11410000	7420	4	1	2	yes	yes	yes	no	yes	2	no	furnished

```
# Check if there are any null values. If any column has null values, treat them accordingly

df.isnull().sum()
```

```
price             0
area              0
bedrooms          0
bathrooms         0
stories           0
mainroad          0
guestroom         0
basement          0
hotwaterheating   0
airconditioning   0
parking           0
prefarea          0
furnishingstatus  0
dtype: int64
```

Activity 2: Data Preparation

This dataset contains many columns having categorical data i.e. values 'Yes' or 'No'. However for linear regression, we need numerical data. So you need to convert all 'Yes' and 'No' values to 1s and 0s, where

- 1 means 'Yes'
- 0 means 'No'

Similarly, replace

- unfurnished with 0
- semi-furnished with 1
- furnished with 2

Hint: To replace all 'Yes' values with 1 and 'No' values with 0, use replace() function of the DataFrame object.

For ex: df.replace(to_replace="yes", value=1, inplace=True) ⇒ replaces the "yes" values in all columns with 1. If you need to make changes inplace, use inplace boolean argument.

```
# Replace all non-numeric values with numeric values.
df.replace(to_replace="no", value=0, inplace=True)
df.replace(to_replace="yes", value=1, inplace=True)
df.replace(to_replace="unfurnished", value=0, inplace=True)
df.replace(to_replace="semi-furnished", value=1, inplace=True)
df.replace(to_replace="furnished", value=2, inplace=True)
```

```
x=df.drop('price',axis=1)
y=df['price']
df.head(10)
```

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwaterheating	airconditioning	parking	prefarea	furnishingstatus
0	13300000	7420	4	2	3	1	0	0	0	1	2	1	2
1	12250000	8960	4	4	4	1	0	0	0	1	3	0	2
2	12250000	9960	3	2	2	1	0	1	0	0	2	1	1
3	12215000	7500	4	2	2	1	0	1	0	1	3	1	2
4	11410000	7420	4	1	2	1	1	1	0	1	2	0	2
5	10850000	7500	3	3	1	1	0	1	0	1	2	1	1
6	10150000	8580	4	3	4	1	0	0	0	1	2	1	1
7	10150000	16200	5	3	2	1	0	0	0	0	0	0	0
8	9870000	8100	4	1	2	1	1	1	0	1	2	1	2
9	9800000	5750	3	2	4	1	1	0	0	1	1	1	0

Activity 3: Train-Test Split

You need to predict the house prices based on several factors. Thus, price is the target variable and other columns except price will be feature variables.

Split the dataset into training set and test set such that the training set contains 67% of the instances and the remaining instances will become the test set.

```
# Split the DataFrame into the training and test sets.
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.67,random_state=15)
```

Activity 4: Model Training

Implement multiple linear regression using sklearn module in the following way:

Reshape the target variable array into two-dimensional arrays by using reshape(-1, 1) function of the numpy module.

Deploy the model by importing the LinearRegression class and create an object of this class.

Call the fit() function on the LinearRegression object.

```
# Create two-dimensional NumPy arrays for the target variable

y_train_reshaped=y_train.values.reshape(-1,1)

y_test_reshaped=y_test.values.reshape(-1,1)

# Build linear regression model

from sklearn.linear_model import LinearRegression

lin_reg=LinearRegression()

lin_reg.fit(x_train,y_train_reshaped)

# Print the value of the intercept

print("slope=>",lin_reg.coef_)

print("Intercept=>",lin_reg.intercept_)

# Print the names of the features along with the values of their corresponding coefficients.
```

```
slope=> [[ 2.73999269e+02  1.58958213e+05  8.57329988e+05  5.03766556e+05
   4.40315675e+05 -5.41799185e+04  4.55791051e+05  7.91742312e+05
   9.25609640e+05  1.99217311e+05  6.56373097e+05  2.25960925e+05]]
Intercept=> [-484639.47636302]
```

Activity 5: Model Prediction and Evaluation

Predict the values for both training and test sets by calling the predict() function on the LinearRegression object.

Also, calculate the R2 , MSE, RMSE and MAE values to evaluate the accuracy of your model.

```
# Predict the target variable values for training and test set

# Predict the target variable values for training and test set

train_pred=lin_reg.predict(x_train)

test_pred=lin_reg.predict(x_test)
```

```python
# Evaluate the linear regression model using the 'r2_score', 'mean_squared_error' & 'mean_absolute_error'
functions of the 'sklearn' module.
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error
print(f"\n\n Test \n{'-'*50}")
print("R2 score",r2_score(y_train_reshaped,train_pred))
print("MSE",mean_squared_error(y_train_reshaped,train_pred))
print("RMSE",mean_squared_error(y_train_reshaped,train_pred))
print("MAC",mean_absolute_error(y_train_reshaped,train_pred))

print(f"\n\n Train \n{'-'*50}")
print(f"r2_sqaured:{r2_score(y_test_reshaped,test_pred):3f}")
print(f"mean_sqaured_error:{mean_squared_error(y_test_reshaped,test_pred):3f}")
print(f"root_mean_sqaured_error:{np.sqrt(mean_squared_error(y_test_reshaped,test_pred)):3f}")
print(f"mean_absolute_error:{mean_absolute_error(y_test_reshaped,test_pred):3f}")
```

```
 Test
--------------------------------------------------
R2 score 0.6821516463495294
MSE 1052523255066.7065
RMSE 1052523255066.7065
MAC 756184.1236636976


 Train
--------------------------------------------------
r2_sqaured:0.655388
mean_sqaured_error:1327866999550.741943
root_mean_sqaured_error:1152331.115414
mean_absolute_error:846583.878282
```

CO-3

9.Support Vector Machines

Loading Data
Let's load both the training and the test datasets.

Train
Dataset:https://raw.githubusercontent.com/akshayr89/MNSIST_Handwritten_Digit_Recognition-SVM/master/train.csv

Test
Dataset:https://raw.githubusercontent.com/akshayr89/MNSIST_Handwritten_Digit_Recognition-SVM/master/test.csv

Dataset credits: http://yann.lecun.com/exdb/mnist/

Now, get the information on both data frames.

```
#Load train & test dataset
import numpy as np
import pandas as pd
train_df=pd.read_csv('https://raw.githubusercontent.com/akshayr89/MNSIST_Handwritten_Digit_Recognition-SVM/master/train.csv')
test_df=pd.read_csv('https://raw.githubusercontent.com/akshayr89/MNSIST_Handwritten_Digit_Recognition-SVM/master/test.csv')
# Get the information on the train dataset.
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

There are 42000 rows and 785 columns in the training dataset.

```
# Get the information on the test dataset.
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28000 entries, 0 to 27999
Columns: 784 entries, pixel0 to pixel783
dtypes: int64(784)
memory usage: 167.5 MB
```

```
# Print the first and last five columns of both the test and train datasets.
train_df.head()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows x 785 columns

```
test_df.head()
```

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows x 784 columns

As you can see, the train set has the label column but the test set doesn't.

Now, let's print the first ten rows of the data frame containing the train set.

```
# Print the first ten rows of the data frame containing the train set.
train_df.head(10)
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
6	7	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
7	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
8	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

10 rows × 785 columns

As you can see:

- The first row contains the pixel values of the image of the handwritten digit 1 .
- Similarly, the second row contains the pixel values of the image of the handwritten digit 0 .
- Similarly, the third row contains the pixel values of the image of the handwritten digit 1 .

The 10th row contains the pixel values of the image of the handwritten digit 3 .
Let's print the image of the digit 4 .

The matplotlib.pyplot.imshow() Function

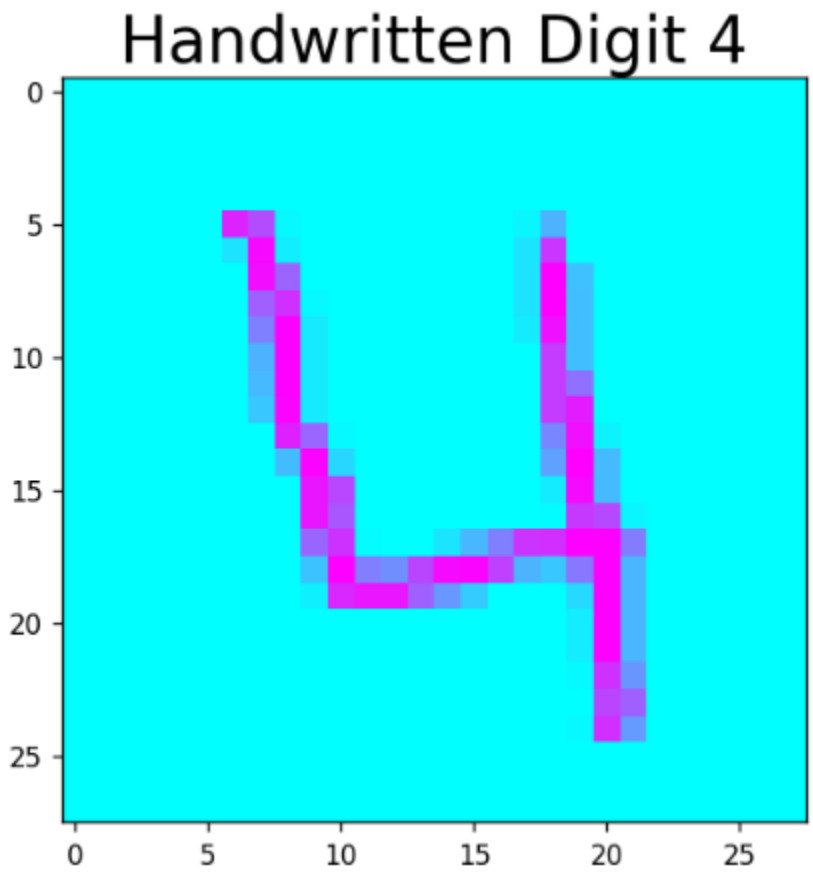
To display an image from its pixel values, you can use the imshow() function of the matplotlib.pyplot module. So, to create the image of the digit 4 from its pixel values, we will follow the steps given below:

1. Create a 1D array containing the pixel values from the training data frame for the image and store it in a variable.
2. Then reshape the above array into a 2D array having 28 rows and 28 columns.
3. Use the imshow() function of the matplotlib.pyplot module and pass the following inputs to the function:
    a. The 28×28 array containing the pixel values of an image
    b. The colour mapping value for the image. We will create a grayscale image hence, we will set the colormapping value using the parameters cmap = 'gray', vmin = 0, vmax = 255.

Note: There are other parameters that can be passed to imshow() function as inputs. But for now, we will pass the above parameters only.

4. Provide the title to the image.

```
# Display the image of the handwritten digit 4 from the train data frame.
import matplotlib.pyplot as plt
fourpixel=train_df.iloc[3,1:]
fourpixel=fourpixel.values.reshape(28,28)
plt.figure(figsize=(5,10),dpi=120)
plt.title("Handwritten Digit 4",size=24)
plt.imshow(fourpixel,cmap='cool')
plt.show()
```



In the above code:

- four_pixels = train_df.iloc[3, 1:] part gets the pixel values of the image of the digit 4 that are stored in the 4th row of the data frame.
- four_pixels = four_pixels.values.reshape(28, 28) part first gets the pixel values from the Pandas series in the form of a NumPy array and then reshapes the 1D array into a 2D array having 28 rows and 28 columns.

- plt.figure(figsize = (5, 5), dpi = 81) part sets the figure size.
- plt.title("Handwritten Digit 4", fontsize = 16) part sets the title of the plot.
- plt.imshow(four_pixels, cmap = 'gray', vmin = 0, vmax = 255) part creates a 2D image in gray colour.

If you look at the axes of the above image, you can see that nearly the first four and last three rows are blank. Similarly, the first five and last five columns are blank which is denoted by the black colour. So let's print the rows 5 to 26 and columns 5 to 25 of the four_pixel NumPy array to see the pixel values of the image of the handwritten digit 4.

```
# Print the rows 5 to 26 and columns 5 to 23 of the 'four_pixel' NumPy array to see the pixel values of the image of the handwritten digit 4.
print(fourpixel[5:26,5:23])
```

```
[[  0 220 179   6   0   0   0   0   0   0   0   0   9  77   0   0   0   0]
 [  0  28 247  17   0   0   0   0   0   0   0   0  27 202   0   0   0   0]
 [  0   0 242 155   0   0   0   0   0   0   0   0  27 254  63   0   0   0]
 [  0   0 160 207   6   0   0   0   0   0   0   0  27 254  65   0   0   0]
 [  0   0 127 254  21   0   0   0   0   0   0   0  20 239  65   0   0   0]
 [  0   0  77 254  21   0   0   0   0   0   0   0   0 195  65   0   0   0]
 [  0   0  70 254  21   0   0   0   0   0   0   0   0 195 142   0   0   0]
 [  0   0  56 251  21   0   0   0   0   0   0   0   0 195 227   0   0   0]
 [  0   0   0 222 153   5   0   0   0   0   0   0   0 120 240  13   0   0]
 [  0   0   0  67 251  40   0   0   0   0   0   0   0  94 255  69   0   0]
 [  0   0   0   0 234 184   0   0   0   0   0   0   0  19 245  69   0   0]
 [  0   0   0   0 234 169   0   0   0   0   0   0   0   3 199 182  10   0]
 [  0   0   0   0 154 205   4   0   0  26  72 128 203 208 254 254 131   0]
 [  0   0   0   0  61 254 129 113 186 245 251 189  75  56 136 254  73   0]
 [  0   0   0   0  15 216 233 233 159 104  52   0   0   0  38 254  73   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  18 254  73   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  18 254  73   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   5 206 106   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 186 159   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   6 209 101   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

From the above output, you can see the non-zero pixel values arranged in the pattern of digit 4.

It is to be noted that the pixel values for a grayscale image range from 0 to 255.

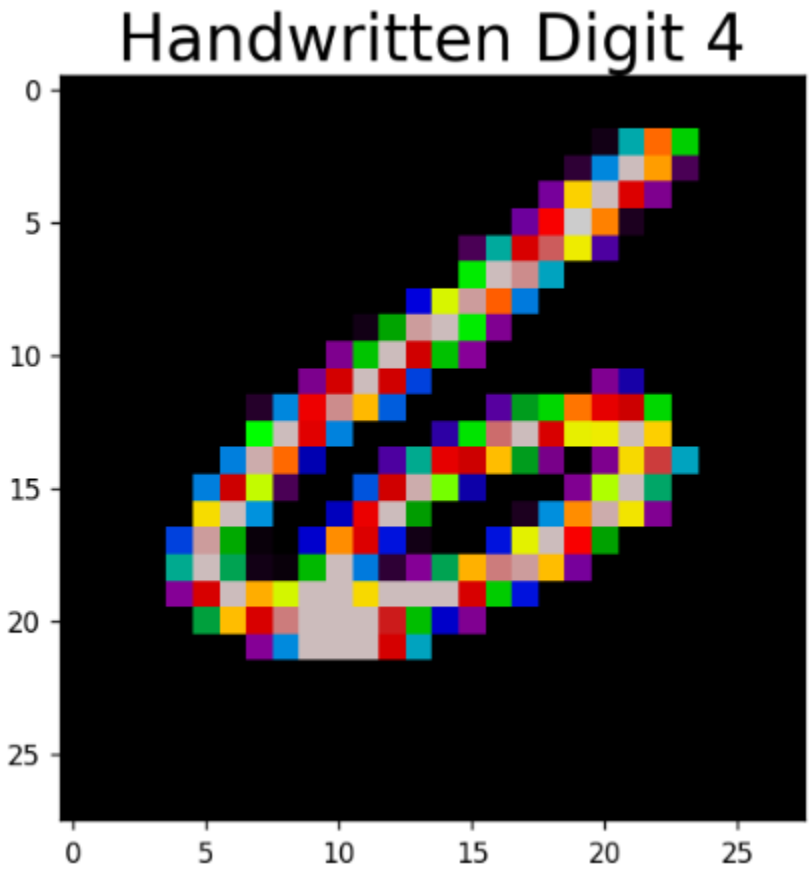
You can also look at the descriptive statistics for the first 10 images in the train data frame.

```
# Create a data frame from the training data frame that contain the pixel values of the images of the digit 6.
```

```
sixpixel=train_df.iloc[21,1:]
sixpixel=sixpixel.values.reshape(28,28)
```

Now, from the above data frame, let's create an image of the first instance of the image of digit 6 . Its index is 21.

```
# Create an image from the pixel values of the image of the digit 6 that are stored in row 21.
plt.figure(figsize=(5,10),dpi=120)
plt.title("Handwritten Digit 4",size=24)
plt.imshow(sixpixel,cmap='nipy_spectral')
plt.show()
```



Now, let's print the part of the array containing the pixel values of the above image such that their arrangement resembles the digit 6.

```
# S3.8: Print the rows 2 to 22 and columns 5 to 21 of the 'six_pixels' array.
print(sixpixel[2:22,5:21])
```

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   5  70]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  27 189 254]
 [  0   0   0   0   0   0   0   0   0   0   0   0  28 219 255 206]
 [  0   0   0   0   0   0   0   0   0   0   8  94 233 248 179  31]
 [  0   0   0   0   0   0   0   0   0   0 146 254 251  84   0   0]
 [  0   0   0   0   0   0   0   0  51 173 252 209  65   0   0   0]
 [  0   0   0   0   0   0   2 119 252 254 146  20   0   0   0   0]
 [  0   0   0   0   0  18 131 254 239 130  25   0   0   0   0   0]
 [  0   0   0   0  17 237 254 239  58   0   0   0   0   0   0  20]
 [  0   0   4  70 223 251 196  61   0   0   0  30 112 138 207 226]
 [  0   0 153 254 228  68   0   0   0  34 143 249 254 233 177 179]
 [  0  67 253 208  40   0   0  31  99 226 241 195 112  14   0  18]
 [ 67 241 168   8   0   0  60 239 253 161  37   0   0   0  20 165]
 [185 254  74   0   0  43 224 254 116   0   0   0   3  73 205 253]
 [252 121   1   0  47 205 230  53   2   0   0  53 176 254 219 118]
 [254 107   2   1 127 254  65   5  24 107 198 250 252 195  27   0]
 [234 254 199 172 254 254 186 254 254 254 234 134  53   0   0   0]
 [109 195 233 250 254 254 254 244 129  46  20   0   0   0   0   0]
 [  0   0  24  71 254 254 254 235  84   0   0   0   0   0   0   0]]
```

Now, for a machine learning algorithm (in this case, SVM), to correctly identify an image for a digit, it has to figure out the arrangement of pixel values for a digit on a 2D grid (in this case, 28×28 grid).

Knowing this, we can now build a machine learning model (in this case, SVM) to classify the images of different handwritten digits.

Check for Data Imbalance

Before building a classification model, let's check whether the training dataset is imbalanced or not.

```
# Find out the counts of records for each digit in the training dataset.
train_df['label'].value_counts()
```

```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```

Note:

1. The dropna = False parameter counts the number of NA or null values if they are present in a Pandas series.

2. The normalize = True parameter calculates the count of a value as the fraction of the total number of records.

From the count of labels, we can see that the training dataset is balanced. Hence, we can now proceed to build a classification model.

Activity 1: Feature Scaling or Normalisation

Now that we have ensured that there is no data imbalance, let's scale down the pixel values of each image because the support vector machines is sensitive to the numeric data. Also, in the case of large values, the time taken to train an SVM model will be high.

So let's divide each pixel value for each image by 255 (the greatest pixel value for a grayscale image) to reduce the values between 0 and 1.

```
# Create features and target data frames and divide each pixel for each image by 255.0
x_train=train_df.iloc[:,1:]/255.0
y_train=train_df['label']
```

Activity 2: Model Building

Let's build a preliminary SVM classification model to classify the images of digits.

Note: Since there are 42000 training samples (or image samples or rows), the SVC model will take some time (about 4 to 6 minutes) to train.

```
#Build an SVC model with the linear kernel.
from sklearn.svm import SVC
svc_linear=SVC(kernel='linear')
svc_linear.fit(x_train,y_train)
```

Now that we have built a classification model using support vector machines, let's get the predicted digts and them compare the predicted values with the actual values.

Note: The code below may take 3 to 5 minutes to execute.

```
#  Predict the target values for the training set.
y_train_pred=svc_linear.predict(x_train)
print(y_train_pred)
```

```
[1 0 1 ... 7 6 9]
```

\
Now let's create a confusion matrix to check for misclassification.

```
# Create a confusion matrix to check for misclassification.
from sklearn.metrics import confusion_matrix,classification_report
print("Confusion Matrix:\n",confusion_matrix(y_train,y_train_pred))
```

```
Confusion Metrix:
 [[4130    0    0    0    0    1    1    0    0    0]
 [   0 4674    2    1    0    0    0    0    6    1]
 [   2    7 4092   16   13    3    6    9   27    2]
 [   6    3   48 4188    1   49    0    5   38   13]
 [   2    6    3    1 3999    0    1    3    0   57]
 [   4    8   12   67    4 3649   19    0   29    3]
 [   1    0    2    1    4   11 4116    0    2    0]
 [   2    3   22    4   10    1    0 4308    2   49]
 [  11   30   19   60    2   49    3    2 3880    7]
 [   4    8    2   12   61    6    0   76   11 4008]]
```

```
#Print the precision, recall and f1-score values to further evaluate the efficacy of the model.
print("classifictaion report=>\n",classification_report(y_train,y_train_pred))
```

```
classifictaion report=>
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      4132
           1       0.99      1.00      0.99      4684
           2       0.97      0.98      0.98      4177
           3       0.96      0.96      0.96      4351
           4       0.98      0.98      0.98      4072
           5       0.97      0.96      0.96      3795
           6       0.99      0.99      0.99      4137
           7       0.98      0.98      0.98      4401
           8       0.97      0.95      0.96      4063
           9       0.97      0.96      0.96      4188

    accuracy                           0.98     42000
   macro avg       0.98      0.98      0.98     42000
weighted avg       0.98      0.98      0.98     42000
```

The f1-scores for all the labels (or digits) are almost equal to 1. This implies that the SVC model built to classify digits is very accurate. So now let's predict the digits on the test set.

Activity 3: Prediction on Test Set

We already know that the test set does not have a label column. So don't need to separate the features and target variables. But we do need to normalise the features in the test set as well with the same technique used for the train set. Hence, we will divide each pixel value in the test set by 255.

```
# Divide each pixel value in the test set by 255. Also, for each image pixels, print the minimum and maximum pixel values.
x_test=test_df.iloc[:,:]/255.0
print(x_test.max())
print(x_test.min())
```

```
pixel0       0.0
pixel1       0.0
pixel2       0.0
pixel3       0.0
pixel4       0.0
            ...
pixel779     0.0
pixel780     0.0
pixel781     0.0
pixel782     0.0
pixel783     0.0
Length: 784, dtype: float64
pixel0       0.0
pixel1       0.0
pixel2       0.0
pixel3       0.0
pixel4       0.0
            ...
pixel779     0.0
pixel780     0.0
pixel781     0.0
pixel782     0.0
pixel783     0.0
Length: 784, dtype: float64
```

Now let's predict the digits for the test set using the SVC model that we just built.

```
# Predict the digits for the test set using the SVC model built above.

y_test_pred=svc_linear.predict(x_test)
print(y_test_pred)
```

```
[2 0 5 ... 3 9 2]
```

Now let's get the count of the predicted labels (or handwritten digits) to see their distribution.

```
# Get the count of the predicted labels (or handwritten digits) to see their distribution.
predicted_counts = pd.Series(y_test_pred).value_counts()

print(predicted_counts)
```

```
1    3288
2    2882
7    2868
3    2818
0    2810
4    2808
6    2729
9    2677
8    2609
5    2511
dtype: int64
```

It seems that the handwritten digits in the test set are quite uniformly distributed.

Activity 4: Visualising Digits

Let's now visualize at least one-one samples from each digit. But first, let's add a new column called label to the test_df data frame so that its values are the predicted labels (or digits). Make sure that the column is added to the column index = 0 location.

```
#  Add 'label' at column index = 0 to the 'test_df' data frame so that its values are the predicted labels (or digits).
test_df.insert(0, 'label', y_test_pred)
```

Lets's display the first 5 rows of the modified test_df data frame.

```
# Display the first 5 rows of the modified 'test_df' data frame.
test_df.head(5)
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

Now let's group all the rows of the test_df data frame by the label column so that pixel values of images of a digit can be clubbed together and a sample of a digit can be retrieved easily later.

Eg., you can easily retrieve one of the sample images of digit 0 from a data frame containing pixel values of all the image samples of the digit 0 only.

```
#  Group all the rows of the 'test_df' data frame by the 'label' column. Also, get a data frame containing pixel values of images of digit 0.
grouped_test_df = test_df.groupby(by="label")
zeros_test_df = grouped_test_df.get_group(0)
zeros_test_df
```

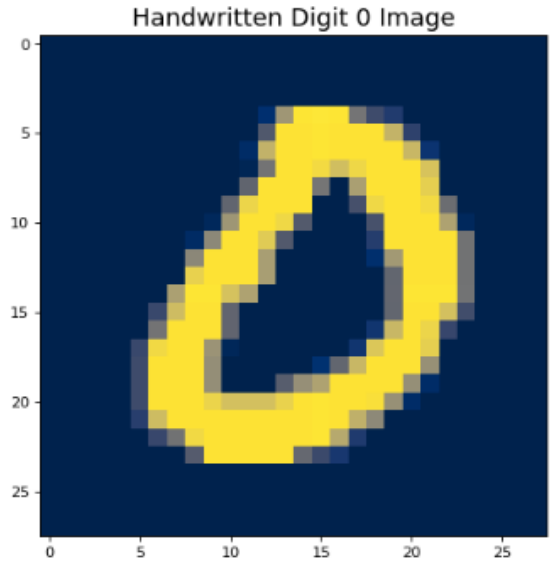
	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
27967	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
27971	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
27974	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
27977	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
27983	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

2810 rows × 785 columns

Now, let's create an image from the pixel values of one of the samples of digit 0.

```
#  Create an image from the pixel values of one of the samples of digit 0.
zero_pixels = test_df.iloc[6, 1:].values.reshape(28, 28)

plt.figure(figsize = (6, 6), dpi = 81)
plt.title("Handwritten Digit 0 Image", fontsize = 16)
plt.imshow(zero_pixels, cmap = "cividis", )
plt.show()
```



Indeed the predicted image is 0. Let's create an image of one of the sample images of digit three.

```
# Get a data frame containing pixel values of all images of digit 3 from 'grouped_test_df' data frame.
grouped_test_df = test_df.groupby(by="label")
threes_test_df = grouped_test_df.get_group(3)
threes_test_df
```

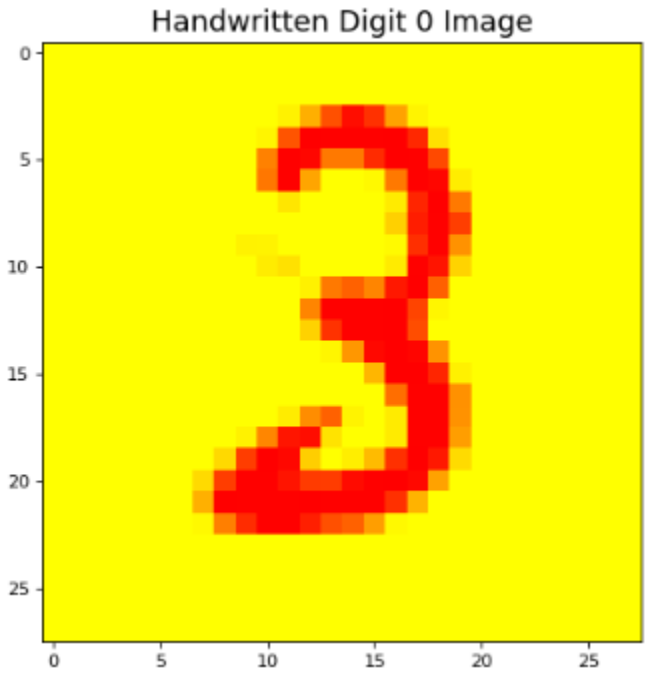
	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
7	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
16	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
53	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
27975	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
27980	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
27985	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
27992	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
27997	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

2818 rows × 785 columns

Now, let's create an image of one of the sample images of digit 3.

```
# Create an image of one of the sample images of digit 3.
three_pixels = test_df.iloc[4, 1:].values.reshape(28, 28)
```

```
plt.figure(figsize = (6, 6), dpi = 81)
plt.title("Handwritten Digit 0 Image", fontsize = 16)
plt.imshow(three_pixels, cmap = "autumn_r")
plt.show()
```



10.Decision trees

Aim: Program to implement decision trees using any standard dataset available in the public domain and find the accuracy of the algorithm.

Short notes

Decision tree is a type of supervised learning algorithm (having a predefined target variable) that is mostly used in classification problems. It works for both categorical and continuous input and output variables. In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.

Used Python Packages:

sklearn : In python, sklearn is a machine learning package which include a lot of ML algorithms. Here, we are using some of its modules like train_test_split, DecisionTreeClassifier and accuracy_score.

NumPy : It is a numeric python module which provides fast maths functions for calculations. It is used to read data in numpy arrays and for manipulation purpose.

Pandas : Used to read and write different files. Data manipulation can be done easily with dataframes.

Important Terminology related to Tree based Algorithms

Root Node: It represents entire population or sample and this further gets divided into two or more homogeneous sets.

Splitting: It is a process of dividing a node into two or more sub-nodes.

Decision Node: When a sub-node splits into further sub-nodes, then it is called decision node.

Leaf/ Terminal Node: Nodes do not split is called Leaf or Terminal node.

Pruning: When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.

Branch / Sub-Tree: A sub section of entire tree is called branch or sub-tree.

Parent and Child Node: A node, which is divided into sub-nodes is called parent node of sub-nodes where as sub-nodes are the child of parent node.

```
#import the modules
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

#load iris data
df=pd.read_csv('https://raw.githubusercontent.com/Adarsh-Soman/DataScience/main/iris-species
.csv')
#Display the number of rows & columns in dataframe
df.shape
df.columns
```

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

```
#Perform Train Test Split
x=df[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
y=df['Species']
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.75,random_state=15)
#Construct decission tree classifier with criterion='entropy' with min_samples_split to 50.
Default value is 2
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier(criterion='entropy')
dtc.fit(x_train,y_train)
pred=dtc.predict(x_test)
#Display Accuracy on test data
from sklearn.metrics import accuracy_score
acc=accuracy_score(y_test,pred)
print("the accuracy is =>",acc)
```

```
the accuracy is => 0.9736842105263158
```

```
#display confusioon metrix for the decision tree
from sklearn.metrics import confusion_matrix
print("Confusion Metrix=>\n",confusion_matrix(y_test,pred))
```

```
Confusion Metrix=>
 [[12  0  0]
 [ 0 14  0]
 [ 0  1 11]]
```

11.KMean

Problem Statement
Program to implement k-means clustering technique using any standard dataset available in the public domain

Dataset Description
In this project, we will be using the dataset holding the information of carbon dioxide emission from different car models.

The dataset includes 36 instances with 5 columns which can be briefed as:

Column	Description
Car	Brand of the car
Model	Model of the car
Volume	Total space available inside the car (in $litres$)
Weight	Total weightof the car (in kg)
CO_2	Total emission of carbon dioxide from the car

Note: (This is a manually created custom dataset for this project.)

List of Activities

Activity 1: Import Modules and Read Data

Activity 2: Data Cleaning

Activity 3: Find Optimal Value of **K**

Activity 4: Plot Silhouette Scores

Activity 1: Import Modules and Read Data

Import the necessary Python modules along with the following modules:

- KMeans - For clustering using K-means.
- re - To remove unwanted rows using regex.

Read the data from a CSV file to create a Pandas DataFrame and go through the necessary data-cleaning process (if required).

Dataset link: https://raw.githubusercontent.com/Adarsh-Soman/DataScience/main/cars.csv

```
# Import the modules and Read the data.
import pandas as pd
df=pd.read_csv('https://raw.githubusercontent.com/Adarsh-Soman/DataScience/main/cars.csv')
# Print the first five records
df
```

	Car	Model	Volume	Weight	CO2
0	Mitsubishi	Space Star	1200	1160	95
1	Skoda	Citigo	1000	929	95
2	Fiat	500	900	865	90
3	Mini	Cooper	1500	1140	105
4	VW	Up!	1000	929	105

```
# Get the total number of rows and columns, data types of columns and missing values (if exist) in the dataset.
print(df.shape)
print(df.dtypes)
print(df.isnull().sum())
```

```
(32, 5)
Car       object
Model     object
Volume     int64
Weight     int64
CO2        int64
dtype: object
Car       0
Model     0
Volume    0
Weight    0
CO2       0
dtype: int64
```

Activity 3: Find Optimal value of K

In this activity, you need to find the optimal value of K using the silhouette score.

1. Create a subset of the dataset consisting of three columns i.e **Volume**, **Weight**, and **CO2**.

```
# Create a new DataFrame consisting of three columns 'Volume', 'Weight', 'CO2'.
new_df=df[["Volume","Weight","CO2"]]
# Print the first 5 rows of this new DataFrame.
new_df.head()
```

	Volume	Weight	CO2
0	1200	1160	95
1	1000	929	95
2	900	865	90
3	1500	1140	105
4	1000	929	105

2. Compute K-Means clustering for the 3D dataset data_3d by varying K from 2 to 10 clusters. Also, for each K, calculate silhouette score using silhouette_score function.

Steps to Follow

- Create an empty list to store silhouette scores obtained for each K (let's say sil_scores).

- Initiate a for loop that ranges from 2 to 10.
- Perform K-means clustering for the current value of K inside for loop.
- Use fit() and predict() to create clusters.
- Calculate silhouette score for current K value using silhouette_score() function and append it to the empty list sil_scores.
- Create a DataFrame with two columns. The first column must contain K values from 2 to 10 and the second column must contain silhouette values obtained after the for loop.

```
# Calculate inertia for different values of 'K'.
from pandas.core.common import random_state
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

sil_score=[]
clusters=range(2,11)
for k in clusters:
  kmeans_k=KMeans(n_clusters=k,random_state=1,n_init=10)
  kmeans_k.fit(new_df)
  cluster_labels=kmeans_k.predict(new_df)
  s=silhouette_score(new_df,cluster_labels)
  sil_score.append(s)

# Create an empty list to store silhouette scores obtained for each 'K'
sil_data=pd.DataFrame({'Clusters':clusters,"Silhouette Score":sil_score})
sil_data
```

	Clusters	Silhouette Score
0	2	0.466982
1	3	0.569304
2	4	0.506027
3	5	0.537547
4	6	0.535720
5	7	0.525962
6	8	0.460244
7	9	0.478152
8	10	0.466768

Q: What are the maximum silhouette score and the corresponding cluster value?

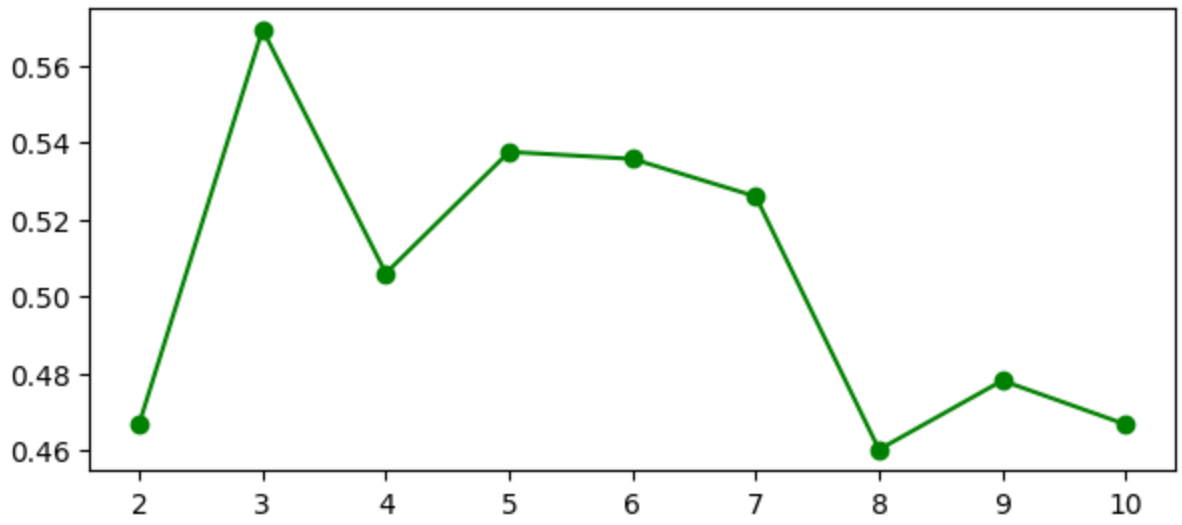
A:

Maximum silhouette score=0.569304
Corresponding cluster value=3

Activity 4: Plot silhouette Scores find optimal value for K

Create a line plot with K ranging from 2 to 10 on the x -axis and the silhouette scores stored in sil_scores list on the y -axis.

```
# Plot silhouette scores vs number of clusters.
import matplotlib.pyplot as plt
plt.figure(figsize=(7,3))
plt.plot(sil_data['Clusters'],sil_data['Silhouette Score'],'g-o')
plt.show()
```



Q: Write your observations of the graph.

A: From the graph, we can conclude that the optimal value of K is 3.

```
# Clustering the dataset for K = 3
# Perform K-Means clustering with n_clusters = 3 and random_state = 10
kmeans_k=KMeans(n_clusters=3,random_state=1,n_init=10)
# Fit the model to the scaled_df
kmeans_k.fit(new_df)
cluster_labels=kmeans_k.predict(new_df)

# Make a pandas series of the predictions done by K-Means
cluster_labels=pd.Series(cluster_labels)
cluster_labels.value_counts()
```

```
1    16
2     9
0     7
dtype: int64
```

```
# Create a DataFrame with cluster labels for cluster visualisation
final_df=pd.concat([df,cluster_labels],axis=1)
final_df.columns=list(df.columns)+["Assigned Cluster"]
final_df.head(5)
```

	Car	Model	Volume	Weight	CO2	Assigned Cluster
0	Mitsubishi	Space Star	1200	1160	95	0
1	Skoda	Citigo	1000	929	95	0
2	Fiat	500	900	865	90	0
3	Mini	Cooper	1500	1140	105	1
4	VW	Up!	1000	929	105	0

12.Feedforward Network

Problem Statement

Program on feedforward network to classify any standard dataset available in the public domain.

```
#Programs on feedforward network to classify any standard dataset available in the public domain.
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import pandas as pd


# Load the Iris dataset
iris=load_iris()
X = iris.data
Y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
# Standardize the feature values
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the feedforward neural network model
 # Output layer with 3 neurons for the 3 classes
model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(3, activation='softmax')])
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```python
# Train the model
model.fit(X_train, Y_train, epochs=50, batch_size=32, validation_split=0.1)

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, Y_test)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 1/50
4/4 [==============================] - 2s 159ms/step - loss: 0.9839 - accuracy: 0.4722 - val_loss: 0.9394 - val_accuracy: 0.6667
Epoch 2/50
4/4 [==============================] - 0s 33ms/step - loss: 0.9229 - accuracy: 0.6759 - val_loss: 0.9026 - val_accuracy: 0.6667
Epoch 3/50
4/4 [==============================] - 0s 35ms/step - loss: 0.8672 - accuracy: 0.7407 - val_loss: 0.8692 - val_accuracy: 0.6667
Epoch 4/50
4/4 [==============================] - 0s 23ms/step - loss: 0.8175 - accuracy: 0.7593 - val_loss: 0.8382 - val_accuracy: 0.7500
Epoch 5/50
4/4 [==============================] - 0s 26ms/step - loss: 0.7729 - accuracy: 0.7870 - val_loss: 0.8083 - val_accuracy: 0.8333
Epoch 6/50
4/4 [==============================] - 0s 28ms/step - loss: 0.7328 - accuracy: 0.7870 - val_loss: 0.7810 - val_accuracy: 0.7500
Epoch 7/50
4/4 [==============================] - 0s 26ms/step - loss: 0.6953 - accuracy: 0.7963 - val_loss: 0.7564 - val_accuracy: 0.8333
Epoch 8/50
4/4 [==============================] - 0s 25ms/step - loss: 0.6634 - accuracy: 0.7963 - val_loss: 0.7333 - val_accuracy: 0.7500
Epoch 9/50
4/4 [==============================] - 0s 24ms/step - loss: 0.6336 - accuracy: 0.7963 - val_loss: 0.7115 - val_accuracy: 0.8333
Epoch 10/50
4/4 [==============================] - 0s 27ms/step - loss: 0.6076 - accuracy: 0.7963 - val_loss: 0.6921 - val_accuracy: 0.8333
Epoch 11/50
4/4 [==============================] - 0s 24ms/step - loss: 0.5831 - accuracy: 0.7963 - val_loss: 0.6747 - val_accuracy: 0.8333
Epoch 12/50
4/4 [==============================] - 0s 23ms/step - loss: 0.5620 - accuracy: 0.7963 - val_loss: 0.6583 - val_accuracy: 0.8333
Epoch 13/50
4/4 [==============================] - 0s 29ms/step - loss: 0.5431 - accuracy: 0.7870 - val_loss: 0.6431 - val_accuracy: 0.8333
Epoch 14/50
4/4 [==============================] - 0s 26ms/step - loss: 0.5248 - accuracy: 0.7870 - val_loss: 0.6290 - val_accuracy: 0.8333
Epoch 15/50
4/4 [==============================] - 0s 40ms/step - loss: 0.5089 - accuracy: 0.8056 - val_loss: 0.6153 - val_accuracy: 0.8333
Epoch 16/50
4/4 [==============================] - 0s 41ms/step - loss: 0.4940 - accuracy: 0.8056 - val_loss: 0.6021 - val_accuracy: 0.8333
Epoch 17/50
4/4 [==============================] - 0s 28ms/step - loss: 0.4804 - accuracy: 0.8148 - val_loss: 0.5895 - val_accuracy: 0.8333
Epoch 18/50
```

```
Epoch 45/50
4/4 [==============================] - 0s 29ms/step - loss: 0.2985 - accuracy: 0.8611 - val_loss: 0.4158 - val_accuracy: 0.9167
Epoch 46/50
4/4 [==============================] - 0s 23ms/step - loss: 0.2950 - accuracy: 0.8704 - val_loss: 0.4124 - val_accuracy: 0.9167
Epoch 47/50
4/4 [==============================] - 0s 25ms/step - loss: 0.2912 - accuracy: 0.8704 - val_loss: 0.4075 - val_accuracy: 0.9167
Epoch 48/50
4/4 [==============================] - 0s 25ms/step - loss: 0.2875 - accuracy: 0.8704 - val_loss: 0.4028 - val_accuracy: 0.9167
Epoch 49/50
4/4 [==============================] - 0s 40ms/step - loss: 0.2839 - accuracy: 0.8704 - val_loss: 0.3979 - val_accuracy: 0.9167
Epoch 50/50
4/4 [==============================] - 0s 38ms/step - loss: 0.2808 - accuracy: 0.8704 - val_loss: 0.3917 - val_accuracy: 0.9167
1/1 [==============================] - 0s 68ms/step - loss: 0.2321 - accuracy: 0.9000
Test accuracy: 0.8999999761581421
```