WEEK-1

Aim: Extract data from different file formats and display the summary statistics.

Description: Sometimes work with some datasets must have mostly worked with .csv(Comma Separated Value) files only. They are really a great starting point in applying Data Science techniques and algorithms. But many of us will land up in Data Science firms or take up real-world projects in Data Science sooner or later. Unfortunately in real-world projects, the data won't be available to us in a neat .csv file. There we have to extract data from different sources like images, pdf files, doc files, image files, etc. In this article, we will see the perfect start to tackle those situations.

Program:

1. csv file

File: sample-csv.csv

	A	B	C	D
1	roll-num	s1	s2	s3
2	101	25	22	26
3	102	24	15	24
4	103	23	13	25
5	104	15	24	23
6	105	26	30	21
7	106	34	26	28
8	107	21	23	27
9	108	30	12	29
10				

Code:

```
import pandas as pd
df=pd.read_csv("sample-csv.csv")
print(df)
print()
print(df.loc[4])
print(df.loc[4,'s1'])
print()
print(df.describe())
print()
```

Output:

```
data :
   roll-num  s1  s2  s3
0       101  25  22  26
1       102  24  15  24
2       103  23  13  25
3       104  15  24  23
4       105  26  30  21
5       106  34  26  28
6       107  21  23  27
7       108  30  12  29

df.loc[4] :
roll-num    105
s1           26
s2           30
s3           21
Name: 4, dtype: int64
df.loc[4,'s1'] :
26

Stats :
        roll-num         s1         s2        s3
count    8.00000   8.000000   8.000000   8.00000
mean   104.50000  24.750000  20.625000  25.37500
std      2.44949   5.700877   6.545173   2.66927
min    101.00000  15.000000  12.000000  21.00000
25%    102.75000  22.500000  14.500000  23.75000
50%    104.50000  24.500000  22.500000  25.50000
75%    106.25000  27.000000  24.500000  27.25000
max    108.00000  34.000000  30.000000  29.00000
```

2. zip file

File: sample-zip.zip

	A	B	C	D
1	roll-num	s1	s2	s3
2	101	25	22	26
3	102	24	15	24
4	103	23	13	25
5	104	15	24	23
6	105	26	30	21
7	106	34	26	28
8	107	21	23	27
9	108	30	12	29
10				

Code:

```
import zipfile
archive = zipfile.ZipFile('sample-zip.zip', 'r')
df = archive.read('sample-csv.csv')
print(df)
```

Output:

```
Out[2]: b'roll-num,s1,s2,s3\r\n101,25,22,26\r\n102,24,15,24\r\n103,23,13,25
\r\n104,15,24,23\r\n105,26,30,21\r\n106,34,26,28\r\n107,21,23,27\r
\n108,30,12,29\r\n'
```

3. excel file

File: sample-excel.xlsx

	A	B	C	D
1	roll-num	s1	s2	s3
2	101	25	22	26
3	102	24	15	24
4	103	23	13	25
5	104	15	24	23
6	105	26	30	21
7	106	34	26	28
8	107	21	23	27
9	108	30	12	29
10				

Code:

```
import pandas as pd
df=pd.read_excel("sample-excel.xlsx")
print(df)
print()
print(df.loc[4])
print(df.loc[4,'s1'])
print()
print(df.describe())
```

Output:

```
data :
   roll-num  s1  s2  s3
0       101  25  22  26
1       102  24  15  24
2       103  23  13  25
3       104  15  24  23
4       105  26  30  21
5       106  34  26  28
6       107  21  23  27
7       108  30  12  29

df.loc[4] :
roll-num    105
s1           26
s2           30
s3           21
Name: 4, dtype: int64
df.loc[4,'s1'] :
26

Stats :
        roll-num         s1         s2         s3
count    8.00000   8.000000   8.000000    8.00000
mean   104.50000  24.750000  20.625000   25.37500
std      2.44949   5.700877   6.545173    2.66927
min    101.00000  15.000000  12.000000   21.00000
25%    102.75000  22.500000  14.500000   23.75000
50%    104.50000  24.500000  22.500000   25.50000
75%    106.25000  27.000000  24.500000   27.25000
max    108.00000  34.000000  30.000000   29.00000
```

4. txt file

File: sample-txt.txt

sample-txt - Notepad

File Edit View

I'm Anjali because I'm anjali so I can also be anju. That means anjali is anjj ans=d anju is anjali.

Code:

```
text=open("sample-txt.txt")
data=text.read()
print(data)
```

Output:

```
I'm Anjali because I'm anjali so I can also be anju. That means anj
ali is anjj ans=d anju is anjali.
```

5. json file

File: sample-json.json

```json
{
    "student": [
    {
        "id":"1",
        "name":"anju"
    },
    {
        "id":"2",
        "name":"ani"
    },
    {
        "id":"3",
        "name":"janu"
    }
    ]
}
```

Code:

```python
import pandas as pd
d=pd.read_json("sample-json.json")
print(d)
print()
print(d.loc[1])
print()
print(d.describe())
```

Output:

```
data :
                           student
0  {'id': '1', 'name': 'anju'}
1   {'id': '2', 'name': 'ani'}
2  {'id': '3', 'name': 'janu'}

d.loc[1] :
student    {'id': '2', 'name': 'ani'}
Name: 1, dtype: object

stats :
                           student
count                            3
unique                           3
top     {'id': '2', 'name': 'ani'}
freq                             1
```

6. xml file

File: sample-xml.xml

```xml
<catalog>

    <book id="bk101">
        <author>Gambardella, Matthew</author>
        <title>XML Developer's Guide</title>
    </book>

    <book id="bk102">
        <author>Ralls, Kim</author>
        <title>Midnight Rain</title>
    </book>

    <book id="bk103">
        <author>Corets, Eva</author>
        <title>Maeve Ascendant</title>
    </book>

</catalog>
```

Code:

```
#retreive data manually

import xml.etree.ElementTree as et
tree = et.parse("sample-xml.xml")
root = tree.getroot()
print (root)
print(root.tag)
print()
print(root[0].tag)
print(root[0].attrib)
print(root[0][0].tag)
print(root[0][0].text)
print(root[0][1].tag)
print(root[0][1].text)
print()
print(root[1].tag)
print(root[1].attrib)
print(root[1][0].tag)
print(root[1][0].text)
print(root[1][1].tag)
print(root[1][1].text)
print()
print(root[2].tag)
print(root[2].attrib)
print(root[2][0].tag)
print(root[2][0].text)
print(root[2][1].tag)
print(root[2][1].text)
```

(OR)

```
#retreive data using loops
import xml.etree.ElementTree as et
tree = et.parse("sample-xml.xml")
root = tree.getroot()
print (root)
print(root.tag)
```

```
print(len(root))
print(len(root[0]))
print()
for i in range(len(root)):
    print(root[i].tag)
    print(root[i].attrib)
    for j in range(len(root[i])):
        print(root[i][j].tag)
        print(root[i][j].text)
    print()
```

Output:

```
<Element 'catalog' at 0x0000023A14D65EF0>
catalog
3

book
{'id': 'bk101'}
author
Gambardella, Matthew
title
XML Developer's Guide

book
{'id': 'bk102'}
author
Ralls, Kim
title
Midnight Rain

book
{'id': 'bk103'}
author
Corets, Eva
title
Maeve Ascendant
```

7. Finding covariance and correlation

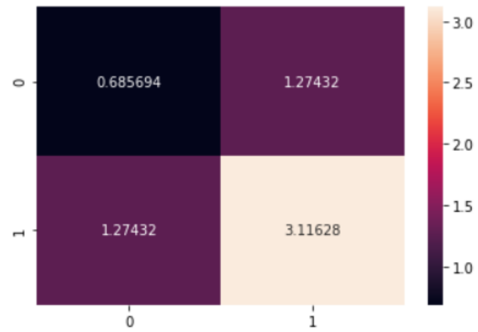
File: iris.csv

Code:

```
# iris dataset
import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt

# summary stats
df=pd.read_csv("iris.csv")
print("summary stats")
print(df.describe())
print()

# covariance matrix
sl=df['slength'].values
pl=df['plength'].values
mat=np.stack((sl,pl),axis=0)
covmat=np.cov(mat)
print("covariance matrix")
print(covmat)
print()

# covariance matrix (with heat-map)
sn.heatmap(covmat, annot=True, fmt='g')
print("covariance matrix (with heat-map)")
plt.show()
print()

# correlation matrix
print("correlation matrix")
corrmat = df.corr()
print(corrmat)
f, ax = plt.subplots(figsize =(9, 8))
sn.heatmap(corrmat, ax = ax, cmap ="YlGnBu", linewidths = 0.1)
```

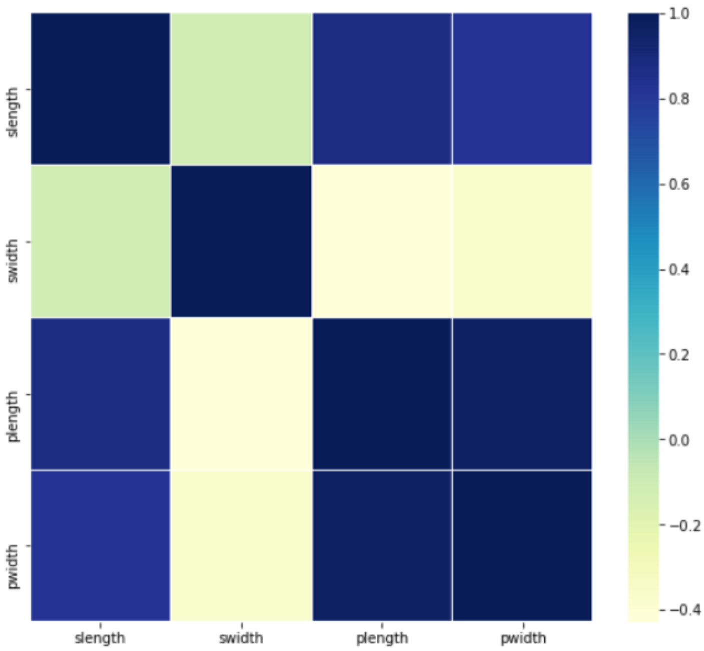
Output:

```
summary stats
          slength      swidth     plength      pwidth
count  150.000000  150.000000  150.000000  150.000000
mean     5.843333    3.057333    3.758000    1.199333
std      0.828066    0.435866    1.765298    0.762238
min      4.300000    2.000000    1.000000    0.100000
25%      5.100000    2.800000    1.600000    0.300000
50%      5.800000    3.000000    4.350000    1.300000
75%      6.400000    3.300000    5.100000    1.800000
max      7.900000    4.400000    6.900000    2.500000


covariance matrix
[[0.68569351 1.27431544]
 [1.27431544 3.11627785]]


covariance matrix (with heat-map)
```




```
 correlation matrix
          slength    swidth   plength    pwidth
slength  1.000000 -0.117570  0.871754  0.817941
swidth  -0.117570  1.000000 -0.428440 -0.366126
plength  0.871754 -0.428440  1.000000  0.962865
pwidth   0.817941 -0.366126  0.962865  1.000000
```

```
Out[22]: <AxesSubplot:>
```



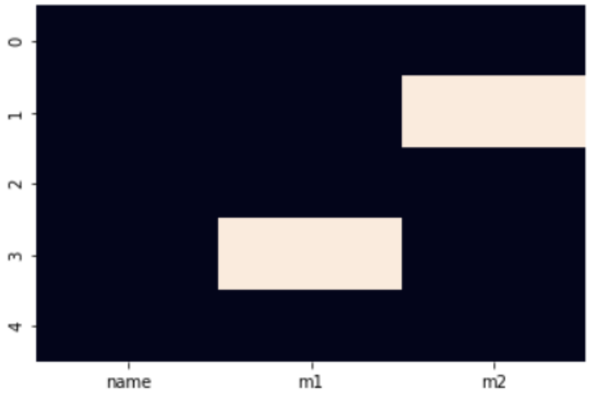
8. Dealing with missing values and finding distance-matrix

Code:

```
# csv consisting missing values
import pandas as pd
import seaborn as sns
import numpy as np
from scipy.spatial import distance_matrix

df=pd.read_csv("csv-with-nan.csv")
print(df)
sns.heatmap(df.isnull(), cbar=False)

# replace missing values
df=df.fillna(df.mean())
print(df)

# distance matrix
print("distance matrix")
x = np.array([[1,2],[2,1],[2,2]])
y = np.array([[5,0],[1,2],[2,0]])
distmat = distance_matrix(x, y, p=2)
print(distmat)
```

Output:

```
with nan values
  name    m1    m2
0  abc  30.0  30.0
1  xyz  23.0   NaN
2  lmn  25.0  29.0
3  pqr   NaN  27.0
4  def  26.0  28.0
without nan values
  name    m1    m2
0  abc  30.0  30.0
1  xyz  23.0  28.5
2  lmn  25.0  29.0
3  pqr  26.0  27.0
4  def  26.0  28.0
distance matrix
[[4.47213595 0.         2.23606798]
 [3.16227766 1.41421356 1.        ]
 [3.60555128 1.         2.        ]]
```



WEEK-2

Aim: Write a program that extracts the words (features) used in a sentence.

Description: First, we have to extract all words from a String, as a string may contain many sentences with punctuation marks.For extracting words from a String first replace all the puncuation marks with spaces and then find the unique words in a sentence.

Program:

Code:

```
import pandas as pd
import numpy as np
import re #regular exp
import nltk #natural language tool kit
nltk.download('stopwords')

corpus = ['The sky is blue and beautiful.',
        'The quick brown fox jumps over the lazy dog.',
        'The sky is very blue and the sky is very beautiful today'
       ]
labels = ['weather',  'animals', 'weather']
# corpus = np.array(corpus)
corpus_df = pd.DataFrame({'Document': corpus,
                  'Category': labels})
# corpus_df = corpus_df[['Document', 'Category']]
print(corpus_df)


wpt = nltk.WordPunctTokenizer()
print('wpt = ',wpt)
stop_words = nltk.corpus.stopwords.words('english')

def normalize_document(doc):
   # lower case and remove special characters\whitespaces
   doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc)
   doc = doc.lower()
```

```
    doc = doc.strip()    #white spaces removing
    # tokenize document
    tokens = wpt.tokenize(doc)
    print('vf ',tokens)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    # re-create document from filtered tokens
    doc = ' '.join(filtered_tokens)
    return doc

normalize_corpus = np.vectorize(normalize_document)

norm_corpus = normalize_corpus(corpus)
print(norm_corpus)


from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer()
print('cv = ', cv)
cv_matrix = cv.fit_transform(norm_corpus)
cv_matrix = cv_matrix.toarray()
print(cv_matrix)

vocab = cv.get_feature_names()
print(pd.DataFrame(cv_matrix, columns=vocab))

bv = CountVectorizer(ngram_range=(2,2))
bv_matrix = bv.fit_transform(norm_corpus)
bv_matrix = bv_matrix.toarray()
vocab = bv.get_feature_names()
print(pd.DataFrame(bv_matrix, columns=vocab))
```

Output:

```
                                             Document Category
0                      The sky is blue and beautiful.  weather
1        The quick brown fox jumps over the lazy dog.  animals
2  The sky is very blue and the sky is very beaut...  weather
wpt =  WordPunctTokenizer(pattern='\\w+|[^\\w\\s]+', gaps=False, discard_empty=True, flags=re.UNICODE|re.MULTILINE|re.DOTALL)
vf  ['the', 'sky', 'is', 'blue', 'and', 'beautiful']
vf  ['the', 'sky', 'is', 'blue', 'and', 'beautiful']
vf  ['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
vf  ['the', 'sky', 'is', 'very', 'blue', 'and', 'the', 'sky', 'is', 'very', 'beautiful', 'today']
['sky blue beautiful' 'quick brown fox jumps lazy dog'
 'sky blue sky beautiful today']
cv =  CountVectorizer()
[[1 1 0 0 0 0 0 0 1 0]
 [0 0 1 1 1 1 1 1 0 0]
 [1 1 0 0 0 0 0 0 2 1]]
   beautiful  blue  brown  dog  fox  jumps  lazy  quick  sky  today
0          1     1      0    0    0      0     0      0    1      0
1          0     0      1    1    1      1     1      1    0      0
2          1     1      0    0    0      0     0      0    2      1
   beautiful today  blue beautiful  blue sky  brown fox  fox jumps  \
0                0               1         0          0          0
1                0               0         0          1          1
2                1               0         1          0          0

   jumps lazy  lazy dog  quick brown  sky beautiful  sky blue
0           0         0            0              0         1
1           1         1            1              0         0
2           0         0            0              1         1
```

WEEK-3

Aim: Write a program for edge detection to extract edge based features from a sample image.

Description: Edge detection is an image processing technique for finding the boundaries of an object in the given image. The edges are the part of the image that represents the boundary or the shape of the object in the image. Also, the pixel values around the edge show a significant difference or a sudden change in the pixel values. Based on this fact we can identify which pixels represent the edge or which pixel lie on the edge.

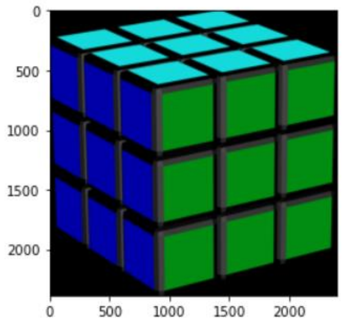
Program:

Code:

(i)using cv2:

```
!pip install opencv-python
import cv2 #Computer Vision
import matplotlib.pyplot as plt

#read the image
img=cv2.imread("abc.png") #cv2- BGR
plt.imshow(img)
```

```
Requirement already satisfied: opencv-python in c:\users\
Requirement already satisfied: numpy>=1.17.3 in c:\users\
Out[2]: <matplotlib.image.AxesImage at 0x1cdd1e80d30>
```



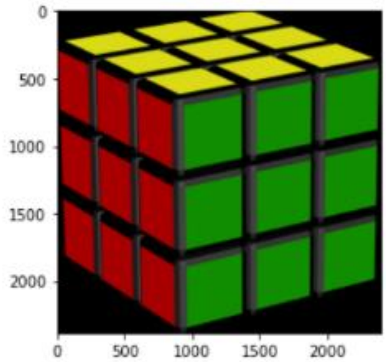
#converting BGR to RGB format

img=cv2.imread("abc.png")

img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.imshow(img)

Out[3]: <matplotlib.image.AxesImage at 0x1cdd2fd5d30>



#converting original picture to gray scale

img=cv2.imread("abc.png",0)

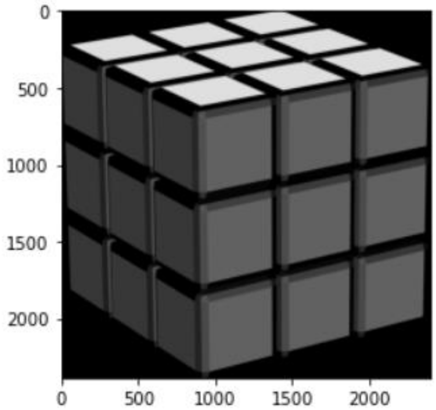
plt.imshow(img,cmap="gray")

(or)

img=cv2.imread("abc.png") #cv2- BGR

img=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

plt.imshow(img,cmap="gray")



#we can use many number of kernals for edge detection

#sobel, laplacin, canny

img=cv2.imread("abc.png",0)

img1=cv2.Sobel(img, cv2.CV_64F,1,0,7) #horizontal 1,0 is x, y coordinate, 5 is kernal

img2=cv2.Sobel(img, cv2.CV_64F,0,1,7) #vertical 1,0 is x, y coordinate, 5 is kernal

img3=cv2.Laplacian(img,cv2.CV_64F)

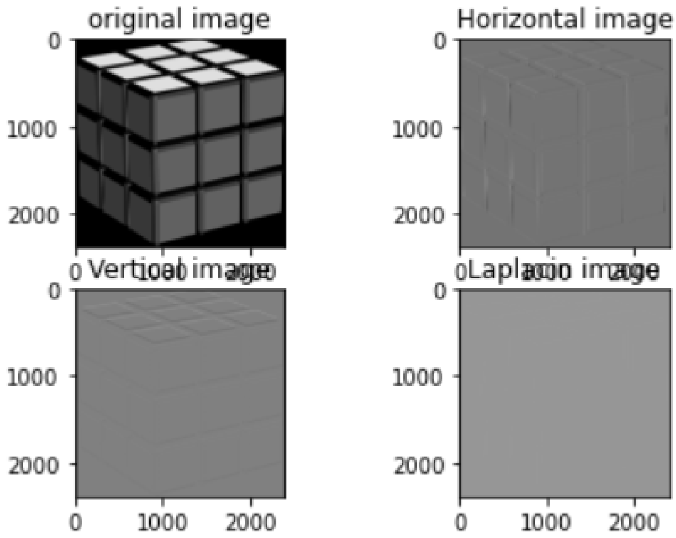
plt.subplot(2,2,1),plt.imshow(img,cmap='gray'),plt.title("original image")

plt.subplot(2,2,2),plt.imshow(img1,cmap='gray'),plt.title("Horizontal image")

plt.subplot(2,2,3),plt.imshow(img2,cmap='gray'),plt.title("Vertical image")

plt.subplot(2,2,4),plt.imshow(img3,cmap='gray'),plt.title("Laplacin image")

```
Out[32]: (<AxesSubplot:title={'center':'Laplacin image'}>,
          <matplotlib.image.AxesImage at 0x1cd80999c40>,
          Text(0.5, 1.0, 'Laplacin image'))
```

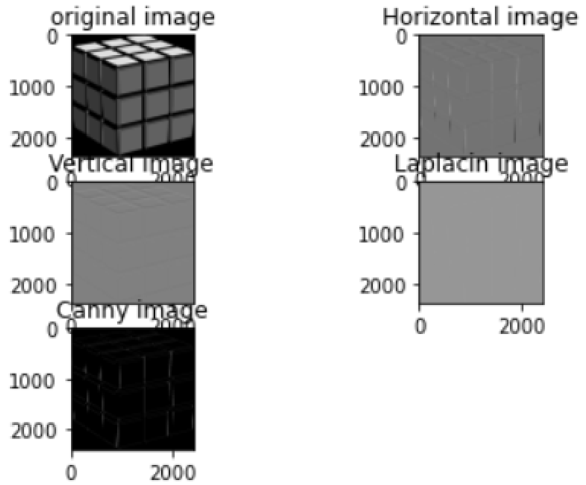


img4=cv2.Canny(img, 50,150)

plt.subplot(3,2,1),plt.imshow(img,cmap='gray'),plt.title("original image")

```
plt.subplot(3,2,2),plt.imshow(img1,cmap='gray'),plt.title("Horizontal image")
plt.subplot(3,2,3),plt.imshow(img2,cmap='gray'),plt.title("Vertical image")
plt.subplot(3,2,4),plt.imshow(img3,cmap='gray'),plt.title("Laplacin image")
plt.subplot(3,2,5),plt.imshow(img4,cmap='gray'),plt.title("Canny image")
```

```
Out[35]: (<AxesSubplot:title={'center':'Canny image'}>,
          <matplotlib.image.AxesImage at 0x1cd8560bc70>,
          Text(0.5, 1.0, 'Canny image'))
```

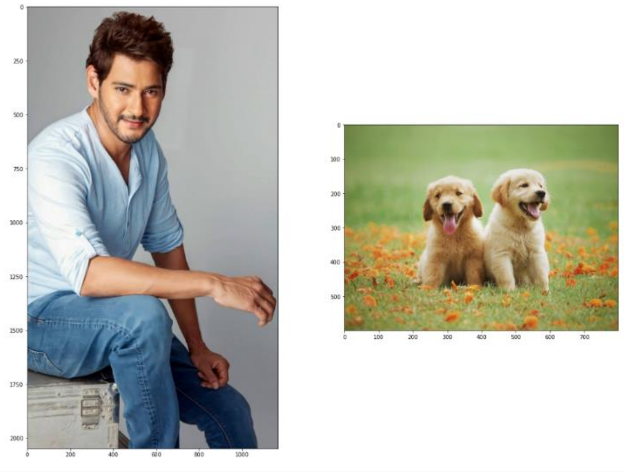


(ii)using skimage:

```
# import skimage
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from skimage.io import imread, imshow
get_ipython().magic('matplotlib inline')
man = imread('man.jpg')  #read in pixcels format
dog = imread('dog.jpg')
df = pd.DataFrame(['Man', 'Dog'], columns=['Image'])
print(man.shape, dog.shape)
```

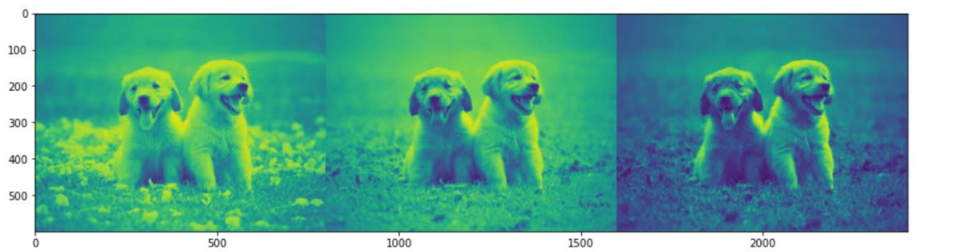
```
op: (2048, 1167, 3) (600, 800, 3)
```

```
fig = plt.figure(figsize = (20,15))
ax1 = fig.add_subplot(1,2, 1)
# print(ax1)
ax1.imshow(man)
ax2 = fig.add_subplot(1,2, 2)
ax2.imshow(dog)
```



```
dog_r = dog[:,:,0]
dog_g = dog[:,:,1]
dog_b = dog[:,:,2]
plot_image = np.concatenate((dog_r, dog_g, dog_b), axis=1)
plt.figure(figsize = (15,7))
plt.imshow(plot_image)
```

```
Out[47]: <matplotlib.image.AxesImage at 0x2204273ad30>
```



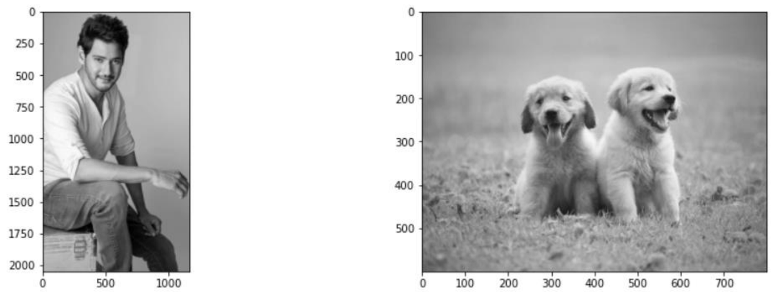
```
from skimage.color import rgb2gray
cgs = rgb2gray(man)
dgs = rgb2gray(dog)
print('Image shape:', cgs.shape, '\n')
# 2D pixel map
print('2D image pixel map')
print(np.round(cgs, 2), '\n')
# flattened pixel feature vector
print('Flattened pixel map:', (np.round(cgs.flatten(), 2)))
```

op:

```
Image shape: (2048, 1167)

2D image pixel map
[[0.58 0.58 0.58 ... 0.74 0.74 0.74]
 [0.58 0.58 0.58 ... 0.74 0.74 0.74]
 [0.58 0.58 0.58 ... 0.74 0.74 0.74]
 ...
 [0.69 0.69 0.69 ... 0.53 0.53 0.53]
 [0.69 0.69 0.69 ... 0.53 0.53 0.53]
 [0.69 0.69 0.69 ... 0.53 0.53 0.53]]

Flattened pixel map: [0.58 0.58 0.58 ... 0.53 0.53 0.53]
```

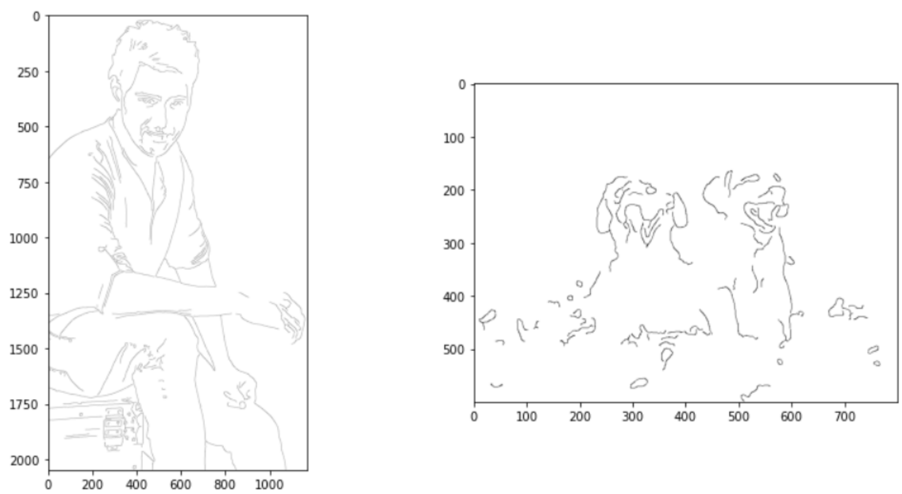
```
fig = plt.figure(figsize = (15,10))
ax1 = fig.add_subplot(2,2, 1)
ax1.imshow(cgs, cmap="gray")
ax2 = fig.add_subplot(2,2, 2)
ax2.imshow(dgs, cmap='gray')
```

```
Out[49]: <matplotlib.image.AxesImage at 0x220409b2bb0>
```



```
from skimage.feature import canny
cat_edges = canny(cgs, sigma=3)
dog_edges = canny(dgs, sigma=3)
fig = plt.figure(figsize = (14,7))
ax1 = fig.add_subplot(1,2, 1)
ax1.imshow(cat_edges, cmap='binary')
ax2 = fig.add_subplot(1,2, 2)
ax2.imshow(dog_edges, cmap='binary')
```

```
Out[53]: <matplotlib.image.AxesImage at 0x2204756d0a0>
```



WEEK-4

Aim: Write a program to extract SURF/SIFT feature descriptors from a sample image.

Description: Feature detection is the process of computing the abstraction of the image information and making a local decision at every image point to see if there is an image feature of the given type existing in that point. SIFT, or Scale Invariant Feature Transform, is a feature detection algorithm in Computer Vision. The major advantage of SIFT features, over edge features or hog features, is that they are not affected by the size or orientation of the image. SURF approximates the DoG with box filters.

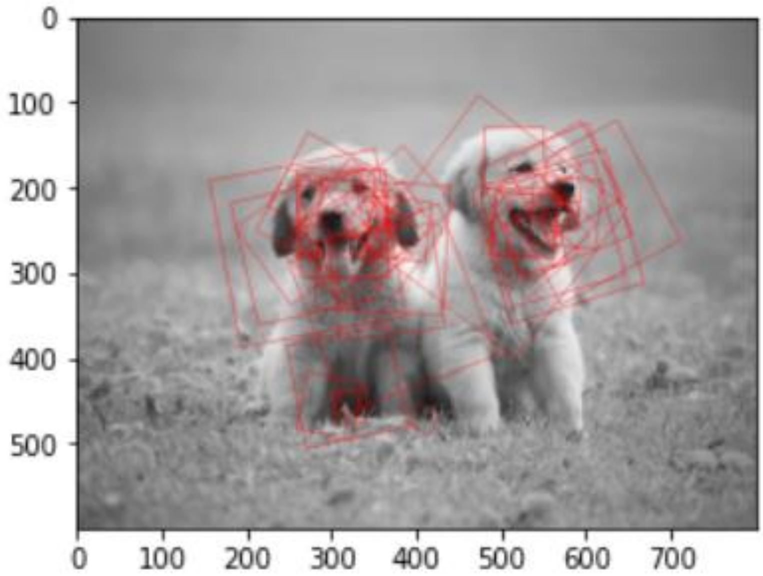
Program:

Code:

(i)SURF:

```
#feature extraction using surf
!pip install mahotas
from mahotas.features import surf
import mahotas as mh
from skimage.io import imread, imshow
from skimage.color import rgb2gray
import matplotlib.pyplot as plt
man = imread('dog.jpg')
man_mh = mh.colors.rgb2gray(man)
man_surf = surf.surf(man_mh, nr_octaves=8, nr_scales=16, initial_step_size=1,
 threshold=0.1, max_points=50)
fig = plt.figure(figsize = (10,4))
ax1 = fig.add_subplot(1,2, 1)
ax1.imshow(surf.show_surf(man_mh, man_surf))
```

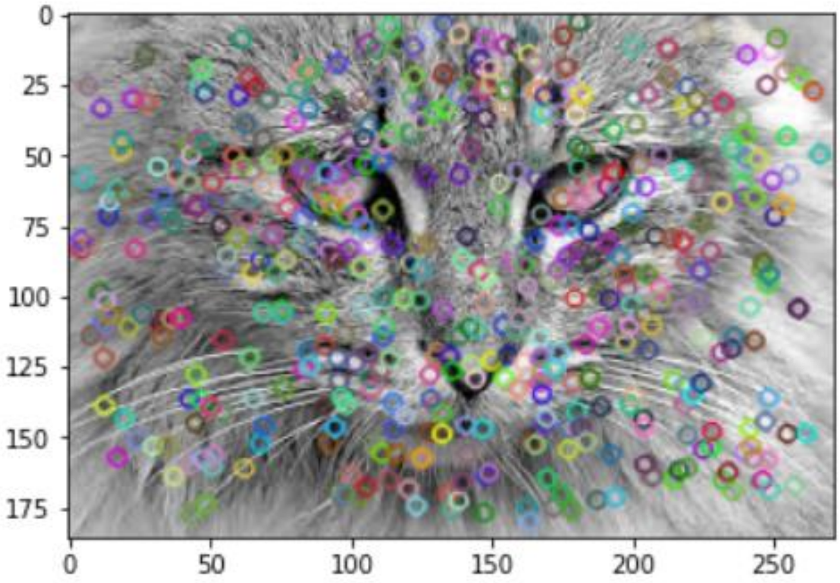
output:



(i)SIFT:

```
#feature extraction with sift
!pip install opencv-python
!pip install opencv-contrib-python
import matplotlib.pyplot as plt
import cv2
# reading the image
# img = cv2.imread('man.jpg')
img = cv2.imread('cat.jpg')
# convert to greyscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# create SIFT feature extractor
sift = cv2.xfeatures2d.SIFT_create()
# detect features from the image
keypoints, descriptors = sift.detectAndCompute(img, None)
# draw the detected key points
sift_image = cv2.drawKeypoints(gray, keypoints, img)
# show the image
plt.imshow(sift_image)
```

output:



WEEK-6

Aim: Write a program to perform Dimensionality Reduction using Principle Component Analysis techniques on real time datasets.

Description: Principal component analysis (PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance.

Program:

Code:

```
# dim reduction using pca for brest cancer dataset
import pandas as pd
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_val_score
from sklearn.decomposition import PCA

pca = PCA(n_components=4)
bc_data = load_breast_cancer()
bc_features=pd.DataFrame(bc_data.data, columns=bc_data.feature_names)
bc_classes = pd.DataFrame(bc_data.target, columns=['IsMalignant'])
bc_X = np.array(bc_features)
bc_y = np.array(bc_classes).T[0]

print('data set')
print(bc_data)
print()
print('features')
print(bc_features)
print()
print('class')
print(bc_classes)
print()
print('features array')
print(bc_X)
print()
print('class array')
print(bc_y)
```

```
pca.fit(bc_X)
PCA(copy=True, iterated_power='auto', n_components=3,
random_state=None, svd_solver='auto', tol=0.0, whiten=False)

print()
print('variance ratio : ')
print(pca.explained_variance_ratio_)

bc_pca = pca.transform(bc_X)
print()
print('transformed values : ')
print(np.round(bc_pca, 2))

# to calc accuracy
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
print()
print('accuracy : ')
print(np.average(cross_val_score(lr, bc_pca, bc_y, scoring='accuracy', cv=5)))
```

Output:

```
features
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0          17.99         10.38          122.80     1001.0          0.11840
1          20.57         17.77          132.90     1326.0          0.08474
2          19.69         21.25          130.00     1203.0          0.10960
3          11.42         20.38           77.58      386.1          0.14250
4          20.29         14.34          135.10     1297.0          0.10030
..           ...           ...             ...        ...              ...
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0             0.27760         0.30010              0.14710         0.2419
1             0.07864         0.08690              0.07017         0.1812
2             0.15990         0.19740              0.12790         0.2069
```

```
3            0.28390         0.24140              0.10520         0.2597
4            0.13280         0.19800              0.10430         0.1809
..               ...             ...                  ...            ...
564          0.11590         0.24390              0.13890         0.1726
565          0.10340         0.14400              0.09791         0.1752
566          0.10230         0.09251              0.05302         0.1590
567          0.27700         0.35140              0.15200         0.2397
568          0.04362         0.00000              0.00000         0.1587

     mean fractal dimension  ...  worst radius  worst texture  \
0                   0.07871  ...        25.380          17.33
1                   0.05667  ...        24.990          23.41
2                   0.05999  ...        23.570          25.53
3                   0.09744  ...        14.910          26.50
4                   0.05883  ...        22.540          16.67
..                      ...  ...           ...            ...
564                 0.05623  ...        25.450          26.40
565                 0.05533  ...        23.690          38.25
566                 0.05648  ...        18.980          34.12
567                 0.07016  ...        25.740          39.42
568                 0.05884  ...         9.456          30.37

     worst perimeter  worst area  worst smoothness  worst compactness  \
0             184.60      2019.0           0.16220            0.66560
1             158.80      1956.0           0.12380            0.18660
2             152.50      1709.0           0.14440            0.42450
3              98.87       567.7           0.20980            0.86630
4             152.20      1575.0           0.13740            0.20500
..               ...         ...               ...                ...
564           166.10      2027.0           0.14100            0.21130
565           155.00      1731.0           0.11660            0.19220
566           126.70      1124.0           0.11390            0.30940
567           184.60      1821.0           0.16500            0.86810
568            59.16       268.6           0.08996            0.06444

     worst concavity  worst concave points  worst symmetry  \
0             0.7119                0.2654          0.4601
1             0.2416                0.1860          0.2750
2             0.4504                0.2430          0.3613
3             0.6869                0.2575          0.6638
4             0.4000                0.1625          0.2364
..               ...                   ...             ...
564           0.4107                0.2216          0.2060
565           0.3215                0.1628          0.2572
566           0.3403                0.1418          0.2218
567           0.9387                0.2650          0.4087
568           0.0000                0.0000          0.2871
```

```
     worst fractal dimension
0                    0.11890
1                    0.08902
2                    0.08758
3                    0.17300
4                    0.07678
..                       ...
564                  0.07115
565                  0.06637
566                  0.07820
567                  0.12400
568                  0.07039

[569 rows x 30 columns]

class
     IsMalignant
0              0
1              0
2              0
3              0
4              0
..           ...
564            0
565            0
566            0
567            0
568            1

[569 rows x 1 columns]

features array
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]

class array
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
```

```
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]

variance ratio :
[9.82044672e-01 1.61764899e-02 1.55751075e-03 1.20931964e-04]

transformed values :
[[1160.14 -293.92   48.58   -8.71]
 [1269.12   15.63  -35.39   17.86]
 [ 995.79   39.16   -1.71    4.2 ]
 ...
 [ 314.5    47.55  -10.44   -9.77]
 [1124.86   34.13  -19.74  -23.66]
 [-771.53  -88.64   23.89    2.55]]

accuracy :
0.9455364073901569
```