Anil Neerukonda Institute of Technology and Sciences

(Affiliated to Andhra University)

Sangivalasa,Visakhapatnam -531162

2021-2025

DATA ANALYTICS LAB

This is to certify that V. Vijay Vamsi is a student studying IV/IV B.Tech CSE bearing Register No. A21126510063 has done 9 experiments during the year 2024-2025 in the subject Data Analytics Lab.

Dr. A Rohini
Associate professor

Prof. Dr. M Rekha Sundari
Head of the Department
Computer Science and Engineering

INDEX

Sl.No	Date	Name of the Experiment	CO	Grade	Signature
1	02/07/2024	Python Numpy (Recap)	1		
2	16/07/2024	Pandas a) Numpy b) Analysis using Pandas	1,2		
3	30/07/2024	Statistical Analysis	1,2		
4	13/08/2024	Data Visualization using Box Plot, Correlogram, and Heatmap	2		
5	20/08/2024	Visualizing Geospatial Data using choropleth map	2		
6	27/08/2024	Data Preprocessing	1,2		
7	17/09/2024	Linear Regression Analysis	3		
8	15/10/2024	Dimensionality Reduction using PCA	4		
9	22/10/2024	K-Means Clustering	4		

List of Experiments

1) Python Numpy (Recap):- Getting familiarity with Python IDE, Notebooks, Data Structures & Numpy.
2) Pandas
    a) Create a Series object from a list, a NumPy array, or a Python dictionary. Apply most of the NumPy functions on the Series object. Create a DataFrame object and Apply arithmetic operations.
    b) Create a dataset of sales data for different products and analyze the total sales and average sales for each product.
3) Perform data pre-processing operations on a Dataset.
4) Perform Statistical analysis (Mean, Median, Mode and Standard deviation) on a Dataset.
5) Perform Visualization using Box Plot, Correlogram, and Heatmap.
6) Visualize geospatial data using choropleth map.
7) Perform Simple Linear Regression and Multiple Linear Regression.
8) Perform dimensionality reduction operation using PCA on a Dataset.
9) Perform K-Means clustering operation and visualize the clusters.

Dt: 02/07/2024 **Experiment-01**

Python Numpy (Recap)

Aim: To get familiarity with Python IDE, Notebooks, Data Structures & Numpy.

1) Python Version Check

```
import sys
print(sys.version)
```

```
3.10.12 (main, Sep 11 2024, 15:47:36) [GCC 11.4.0]
```

MAGIC COMMANDS :

Magic commands, indicated by a % or %% prefix, offer quick access to various Jupyter and IPython functionalities, likefile management, timing, and debugging.

2) %pwd returns current working directory

```
%pwd
```

```
'/home/anits'
```

3) Use the %pastebin magic function to select a range of cells

```
%pastebin 1-2
```

```
'https://dpaste.com/BH9TTKRY3'
```

4) To have a list of defined variables, use %whos or %who_ls

```
x,y="Hello","world"
%whos
%who_ls
```

```
Variable   Type    Data/Info
----------------------------
x          str     Hello
y          str     world
['x', 'y']
```

5) %system ➞ to use shell (mostly used to get current directory, date, etc)

```
%system date
```

```
['Sat Oct 26 08:33:52 AM UTC 2024']
```

6) %timeit measures the execution time of the code in the current cell to help evaluate performance

```
%timeit x = range(1000)
```

```
279 ns ± 18.9 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

7) Autosave every 120 seconds

```
%autosave 120
```

```
Autosaving every 120 seconds
```

8) %%HTML to execute HTML code

```
%%HTML
This is <i>HTML</i> code!
<br><br>
```

This is *HTML* code

9) %history displays a list of all previously run commands in the session

```
%history
```

```
%history
%%HTML
This is <i>HTML</i> code!
%autosave 120
%%HTML
This is <i>HTML</i> code!
%timeit x = range(1000)
%system date
x,y="Hello","world"
%whos
%who_ls
%pastebin
1-3 import
sys
print(sys.version)
!conda list
!conda list
%pwd
%history
```

10) %lsmagic list currently available magic functions

```
%lsmagic
```

```
Available line magics:
%alias  %alias_magic  %autoawait  %autocall  %automagic  %autosave  %bookmark  %cat  %cd  %clear
%code_wrap  %colors  %conda
%config  %connect_info  %cp  %debug  %dhist  %dirs  %doctest_mode  %ed  %edit  %env  %gui  %hist
%history  %killbgscripts  %ldir
```

```
%less  %lf  %lk  %ll  %load  %load_ext  %loadpy  %logoff  %logon  %logstart  %logstate  %logstop  %ls
%lsmagic  %lx  %macro
%magic  %mamba  %man  %matplotlib  %micromamba  %mkdir  %more  %mv  %notebook  %page  %pastebin  %pdb
%pdef  %pdoc  %pfile  %pinfo
%pinfo2  %pip  %popd  %pprint  %precision  %prun  %psearch  %psource  %pushd  %pwd  %pycat  %pylab
%qtconsole  %quickref  %recall
%rehashx  %reload_ext  %rep  %rerun  %reset  %reset_selective  %rm  %rmdir  %run  %save  %sc  %set_env
%store  %sx  %system  %tb
%time  %timeit  %unalias  %unload_ext  %who  %who_ls  %whos  %xdel  %xmode
```

11) setting up matplotlib

```
%matplotlib
```

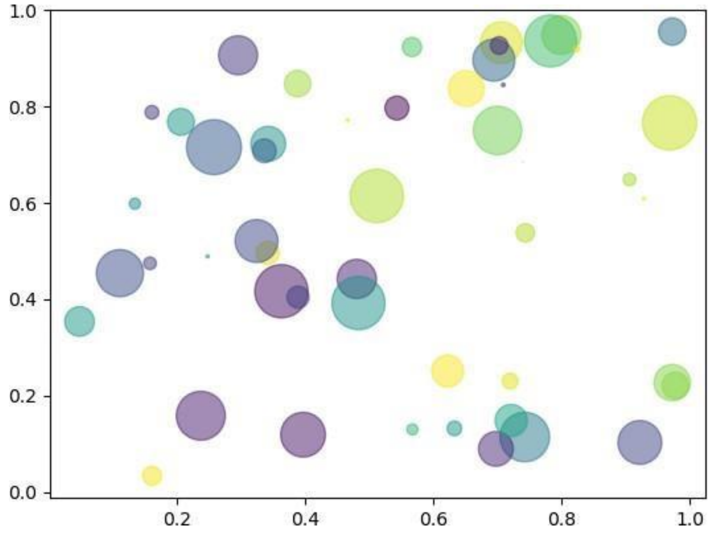
```
Using matplotlib backend: <object object at 0x7f69c8fab2b0>
```

```
>> import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

12) **Random Scatter Plot Generation**:

Generates a scatter plot with 50 random points, where x and y coordinates are randomly distributed. The size of each point (area) is scaled by arandom value, and colors are randomly assigned, resulting in a visually varied plot

```
>> np.random.seed(19680801)
N =50
x =np.random.rand(N)
y =np.random.rand(N)
colors=np.random.rand(N)
area =(30*np.random.rand(N))**2  # 0 to 15 point radii
plt.scatter(x, y, s=area, c=colors, alpha=0.5)
plt.show()
```



Dt: 16/07/2024

Experiment-02

Pandas

a) Aim:- To create a Series object from a list, a NumPy array, or a Python dictionary and apply most of the NumPy functions on the Series object and create a DataFrame object and apply arithmetic operations.

1) Creating Numpy Arrays

```
>> import numpy as np
ar=np.array([1,2,3])
print(ar)
a2=np.arange(2,78,5)
print(a2)
```

```
[1 2 3] [ 2 7 12 17 22 27 32 37 42 47 52 57 62 67 72 77]
```

```
>> %timeit np.sum(a2)
```

```
2.79 µs ± 39.9 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

2) Python versus NumPy

```
>> import numpy as np
arr1 = list(range(1000000))
arr2 = list(range(1000000, 2000000))
# Convert lists to NumPy arrays
arr1_np = np.array(arr1)
arr2_np = np.array(arr2)
# Timing the pure Python dot product
def python_dot_product():
      result = 0
      for x1, x2 in zip(arr1, arr2):
            result += x1 * x2
            return result
%timeit python_dot_product()
%timeit np.dot(arr1_np, arr2_np)
```

```
65.3 ms ± 2.27 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
1.64 ms ± 162 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

```
>> a=np.ones([2,3,4])
   a
```

```
array([[[1., 1., 1., 1.], [1., 1., 1., 1.], [1., 1., 1., 1.]], [[1., 1., 1., 1.], [1., 1., 1., 1.], [1., 1., 1., 1.]]])
```

3) Array Operations with NumPy

```
>> print('First array:')
print(a)
```

```
print('Second array:')
b =np.array([10,10,10])
print(b)
print('Add the two arrays:')
   print(np.add(a,b))
print('Subtract the two arrays:')
print(np.subtract(a,b))

print('Multiply the two arrays:')
print(np.multiply(a,b))
print('Divide the two arrays:')
print (np.divide(a,b))
print('Power function:')
print(np.power(a,b))
```

```
First array: [1 2 3]
Second array: [10 10 10]
Add the two arrays: [11 12 13]
Subtract the two arrays: [-9 -8 -7]
Multiply the two arrays: [10 20 30]
Divide the two arrays: [0.1 0.2 0.3]
Power function: [ 1 1024 59049]
```

```
>> import numpy as np
# Create an array of zeros with shape (10, 2)
p = np.zeros((10, 2))
# Transpose the array p
q = p.T
# Print the shape of q
print("Shape of q:", np.shape(q))
# Create a view of q
r = q.view()
r = r.reshape((20,))
r2=r.reshape((2,2,5))
print("Reshaped r:", r)
print("Reshaped r2:", r2)
print("Shape of r:", np.shape(r))
```

```
Shape of q: (2, 10)
Reshaped r: [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Reshaped r2: [[[0. 0. 0. 0. 0.] [0. 0. 0. 0. 0.]] [[0. 0. 0. 0. 0.] [0. 0. 0. 0.
0.]]]
```

```
>> ar=np.eye(4)
ar
```

```
array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0.,
1.]])
```

```
>> a2=np.linspace(0,20,5,dtype=np.int32)
print(a2)
```

```
[ 0 5 10 15 20]
```

```
>> I = np.eye(3)
print(I)

# Create a 3x4 matrix with ones on the diagonal
I_non_square = np.eye(3, 4,dtype=np.int32)
print(I_non_square)

# Create a matrix with ones on the diagonal above the main diagonal
I_offset = np.eye(3, k=1,dtype=np.int32)
print(I_offset)
```

```
[[1. 0. 0.] [0. 1. 0.] [0. 0. 1.]] [[1 0 0 0] [0 1 0 0] [0 0 1 0]] [[0 1
0] [0 0 1] [0 0 0]]
```

```
>> var1=np.random.rand(2,3)
print(var1)
```

```
[[0.18767901 0.41599472 0.62498392] [0.61180751 0.62717573 0.40913394]]
```

```
>> v2=np.random.randn(2,3)
print(v2)
```

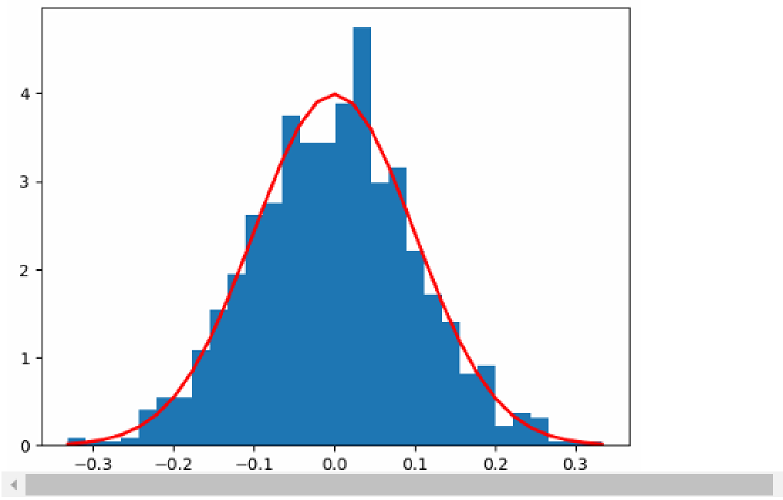
```
[[ 2.46724196 0.39013296 -0.0092183 ] [ 0.06845898 0.28148882 2.75561248]]
```

```
>> var3=np.random.ranf(4)
print(var3)
```

```
[0.43512249 0.09533848 0.86853993 0.67759174]
```

4) <u>Generating and Visualizing a Normal Distribution</u>

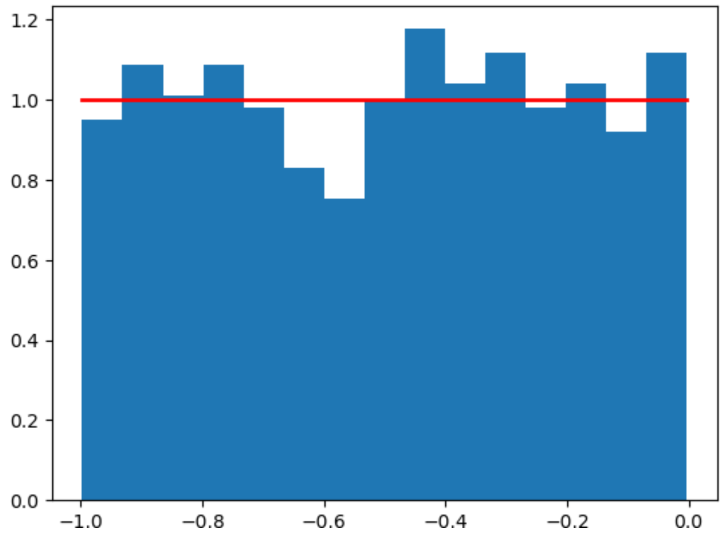
```
>>mu, sigma =0,0.1
# mean and standard deviation
s =np.random.normal(mu, sigma,1000)
import matplotlib.pyplot as plt
count, bins, ignored =plt.hist(s,30, density=True) plt.plot(bins,1/(sigma
*np.sqrt(2*np.pi))*np.exp(-(bins - mu)**2/(2*sigma**2)),linewidth=2,color='r')
plt.show()
```




```
>>s =np.random.uniform(-1,0,1000)
print(np.all(s >=-1))
```

```
print(np.all(s < 0)
import matplotlib.pyplot as plt
count, bins, ignored =plt.hist(s,15, density=True)
plt.plot(bins,np.ones_like(bins),linewidth=2,color='r')
plt.show()
```

```
True
True
```



(b) Aim:- To create a dataset of sales data for different products and analyze the total sales and average sales for each product using **Pandas**.

```
>> import pandas as pd
import numpy as np
s1=pd.Series([1,2,'hello',6.78,True])

a=np.array([1,2,3,'rh',4])
print(a.dtype)
s2=pd.Series(a)
s3=pd.Series(a,index=['a','b','c','d','e'])
dict={1:'v',2:'h',3:'d'}
```

```
dtype: object
```

```
>> a1=pd.Series(np.repeat(2,6))
print(a1)
```

```
0 2
1 2
2 2
3 2
4 2
5 2
dtype: int64
```

```
>> a=pd.Series(np.linspace(1,17,5))
print(a)
print()
b=pd.Series(np.arange(1,10,3))
```

```
print(b)
print()
```

```
0 1.0
1 5.0
2 9.0
3 13.0
4 17.0
dtype: float64
```

```
>> print("DataFrame using dictionary")
d={
     "Name":["AAA","BBB","CCC"],
     "Marks":[29,33,34],
     "rank":[3,2,1] }
n1=pd.DataFrame(d)
print(n1)

print("DataFrame using list of tuples")
d1=[("12-01-2022","23-09-21021","21-04-2011"),
    (18,34,45),
    ("low","medium","high")]
n2=pd.DataFrame(d1,columns=["Day","Temperature","T_Category"])
print(n2)

print("DataFrame using numpy arrays")
l=np.array([[1,2,3],[4,5,6],[7,4,5]])
n3=pd.DataFrame(l,index=['a','b','c'])
print(n3)

l=np.array([[1,3,4],[12,13,14],[234,44,55]])
n4=pd.DataFrame(l,index=['a','b','c'])
print(n4)
```


```
print("DataFrame using list of tuples")
d1=[("12-01-2022","23-09-21021","21-04-2011"),
    (18,34,45),
    ("low","medium","high")]
n2=pd.DataFrame(d1,columns=["Day","Temperature","T_Category"])
print(n2)

print("DataFrame using numpy arrays")
l=np.array([[1,2,3],[4,5,6],[7,4,5]])
n3=pd.DataFrame(l,index=['a','b','c'])
print(n3)

l=np.array([[1,3,4],[12,13,14],[234,44,55]])
n4=pd.DataFrame(l,index=['a','b','c'])
print(n4)
```


```
DataFrame using dictionary
    Name Marks rank
  0 AAA  29    3
  1 BBB  33    2
  2 CCC  34    1
DataFrame using list of tuples
    Day        Temperature T_Category
  0 12-01-2022 23-09-21021 21-04-2011
  1 18         34          45
  2 low        medium      high
```

```
DataFrame using numpy arrays
          0 1 2
        a 1 2 3
        b 4 5 6
        c 7 4 5

          0   1   2
        a  1    3  4
        b 12   13 14
        c 234  44 55
```

2) Arithmetic Operations

```
>> print("Addition : ")
print(n3+n4)
print("Subtraction : ")
print(n4-n3)
print("Multiplication : ")
print(n3*n4)
print("Division : ")
print(n4/n3)
print("Power : ")
print(n3^n4)
```

```
Addition :
        0  1  2
    a   2  5  7
    b  16 18 20
    c 241 48 60

Subtraction :
        0  1  2
    a   0  1  1
    b   8  8  8
    c 227 40 50

Multiplication :
         0   1   2
    a    1   6  12
    b   48  65  84
    c 1638 176 275

Division :
          0    1         2
a  1.000000  1.5  1.333333
b  3.000000  2.6  2.333333
c 33.428571 11.0 11.000000

Power :
    0  1  2
a   0  1  7
b   8  8  8
c   237 40 50
```

Dt: 30/07/2024

Experiment-03
Statistical analysis

Aim:- To perform Statistical analysis (Mean, Median, Mode and Standard deviation) on a Dataset.

1) Load a CSV data file into a Pandas DataFrame object:

```
>> import pandas as pd
file=pd.read_csv('/home/anits/Downloads/Top_100_Movies.csv')
# file.head()
file.tail(3)
```

	Unnamed: 0	rank	title	description	genre	rating	id	year	imdbid	imdb_link
97	97	98	Lawrence of Arabia	The story of T.E. Lawrence, the English office...	['Adventure', 'Biography', 'Drama']	8.3	top98	1962	tt0056172	https://www.imdb.com/title/tt0056172 amazon.com/im

```
>> file.info()
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 11 columns):

#	Column	Non-Null Count	Dtype
0	Unnamed: 0	100 non-null	int64
1	rank	100 non-null	int64
2	title	100 non-null	object
3	description	100 non-null	object
4	genre	100 non-null	object
5	rating	100 non-null	float64
6	id	100 non-null	object
7	year	100 non-null	int64
8	imdbid	100 non-null	object
9	imdb_link	100 non-null	object
10	image	100 non-null	object

dtypes: float64(1), int64(3), object(7)
memory usage: 8.7+ KB

```
>> file.describe()
```

	Unnamed: 0	rank	rating	year
count	100.000000	100.000000	100.00000	100.000000
mean	49.500000	50.500000	8.52200	1988.070000
std	29.011492	29.011492	0.20869	23.069178
min	0.000000	1.000000	8.30000	1931.000000
25%	24.750000	25.750000	8.40000	1974.750000
50%	49.500000	50.500000	8.50000	1994.000000
75%	74.250000	75.250000	8.60000	2003.250000
max	99.000000	100.000000	9.30000	2023.000000

2) <u>Compute various summary statistics from the DataFrame:</u>

```
>> mv=file['rating'].min()
print(mv)
print(file['rating'].idxmin())
mv=file['rating'].max()
print(mv)
print(file['rating'].idxmax())
```

```
8.3
82
9.3
0
```

```
>> m1=file['rating'].mean()
m2=file['rating'].median()
m3=file['rating'].mode()
print(m1,m2,m3)
m4=file['rating'].var()
m5=file['rating'].std()
print(m4,' ',m5)
```

```
8.521999999999998 8.5 0 8.4
Name: rating, dtype: float64
0.04355151515151502 0.20868999772752653
```

```
>> import numpy as np
cols=['A','B','C','D']
df1=pd.DataFrame([[np.nan,2,np.nan,0],
                  [3,4,np.nan,1],
                  [np.nan,np.nan,np.nan,np.nan],
                  [np.nan,3,np.nan,4]],columns=cols)
df1.cov() #co variance-measures relationship b/w 2 variables
```

	A	B	C	D
A	NaN	NaN	NaN	NaN
B	NaN	1.0	NaN	0.500000
C	NaN	NaN	NaN	NaN
D	NaN	0.5	NaN	4.333333

```
>> df1.corr()
```

	A	B	C	D
A	NaN	NaN	NaN	NaN
B	NaN	1.000000	NaN	0.240192
C	NaN	NaN	NaN	NaN
D	NaN	0.240192	NaN	1.000000

Dt: 13/08/2024

Experiment-04

Matplotlib and Seaborn

Aim:- To perform Visualization using Box Plot, Correlogram, and Heatmap

```
>> import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df=pd.read_csv('/home/anits/Downloads/starbucks.csv')
```

	category	name	prep	Calories	Fat	TransFat	Carb	Cholesterol	Sugar	Protein	Caffeine
0	Coffee	Brewed Coffee	Short	3	0.1	0.0	5	0	0	0.3	175
1	Coffee	Brewed Coffee	Tall	4	0.1	0.0	10	0	0	0.5	260
2	Coffee	Brewed Coffee	Grande	5	0.1	0.0	10	0	0	1.0	330
3	Coffee	Brewed Coffee	Venti	5	0.1	0.0	10	0	0	1.0	410
4	Classic Espresso Drinks	Caffè Latte	Short Nonfat Milk	70	0.1	0.1	75	10	9	6.0	75

```
>> print(df.isnull().sum())
df = df.dropna()
df.info()
```

```
category       0
name           0
prep           0
Calories       0
Fat            0
TransFat       0
 Carb          0
Cholesterol    0
 Sugar         0
 Protein       0
Caffeine       1
dtype: int64

<class 'pandas.core.frame.DataFrame'>
Index: 241 entries, 0 to 241
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  ---------
 0   category     241 non-null    object
 1   name         241 non-null    object
 2   prep         241 non-null    object
 3   Calories     241 non-null    int64
 4   Fat          241 non-null    object
 5   TransFat     241 non-null    float64
 6    Carb        241 non-null    int64
 7   Cholesterol  241 non-null    int64
 8    Sugar       241 non-null    int64
 9    Protein     241 non-null    float64
 10  Caffeine     241 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 22.6+ KB
```

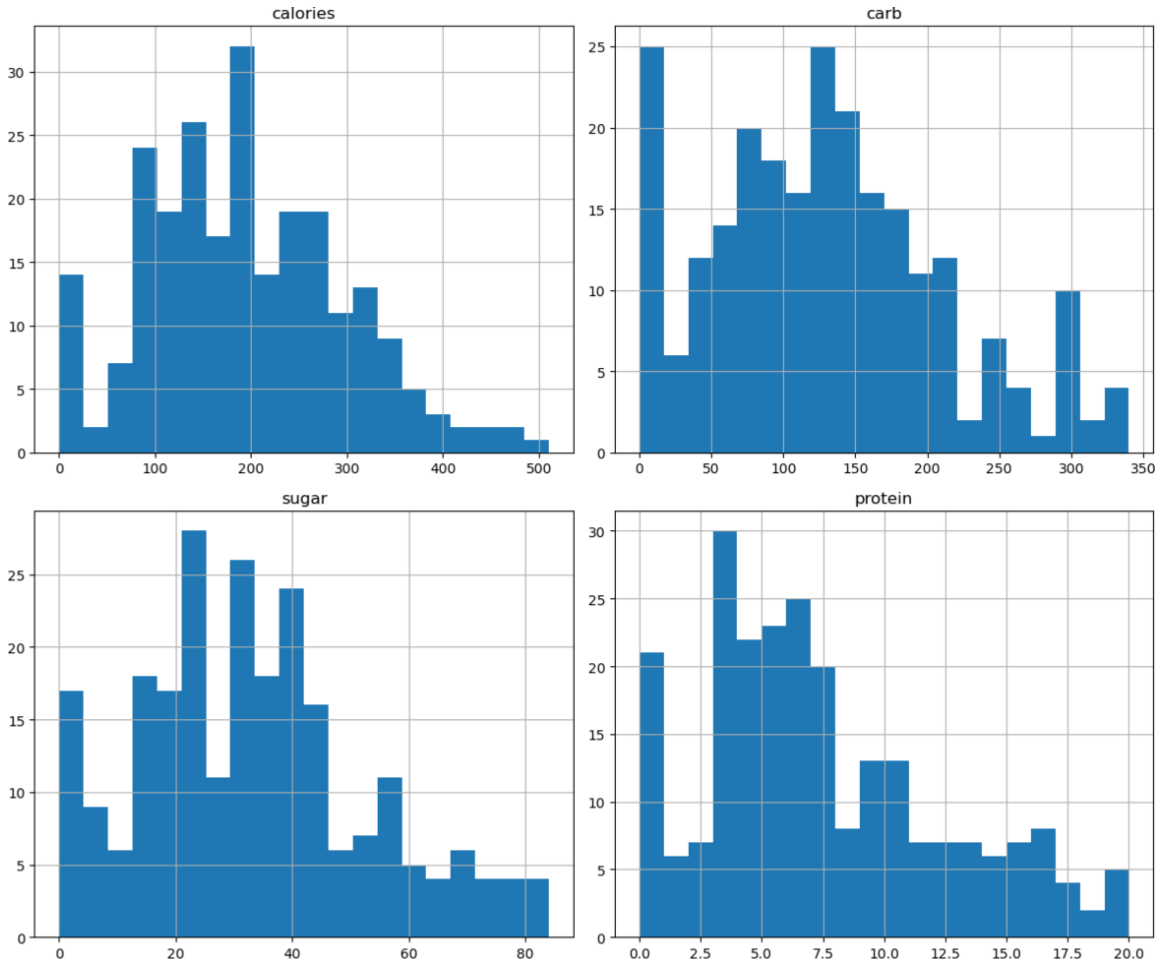
```
>> df.describe()
```

	Calories	TransFat	Carb	Cholesterol	Sugar	Protein
count	241.000000	241.000000	241.000000	241.000000	241.000000	241.000000
mean	194.302905	1.310373	129.315353	36.066390	33.024896	6.999170
std	102.858173	1.642843	82.200315	20.805942	19.747558	4.871165
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	120.000000	0.100000	70.000000	21.000000	18.000000	3.000000
50%	190.000000	0.500000	125.000000	34.000000	32.000000	6.000000
75%	260.000000	2.000000	170.000000	51.000000	44.000000	10.000000
max	510.000000	9.000000	340.000000	90.000000	84.000000	20.000000

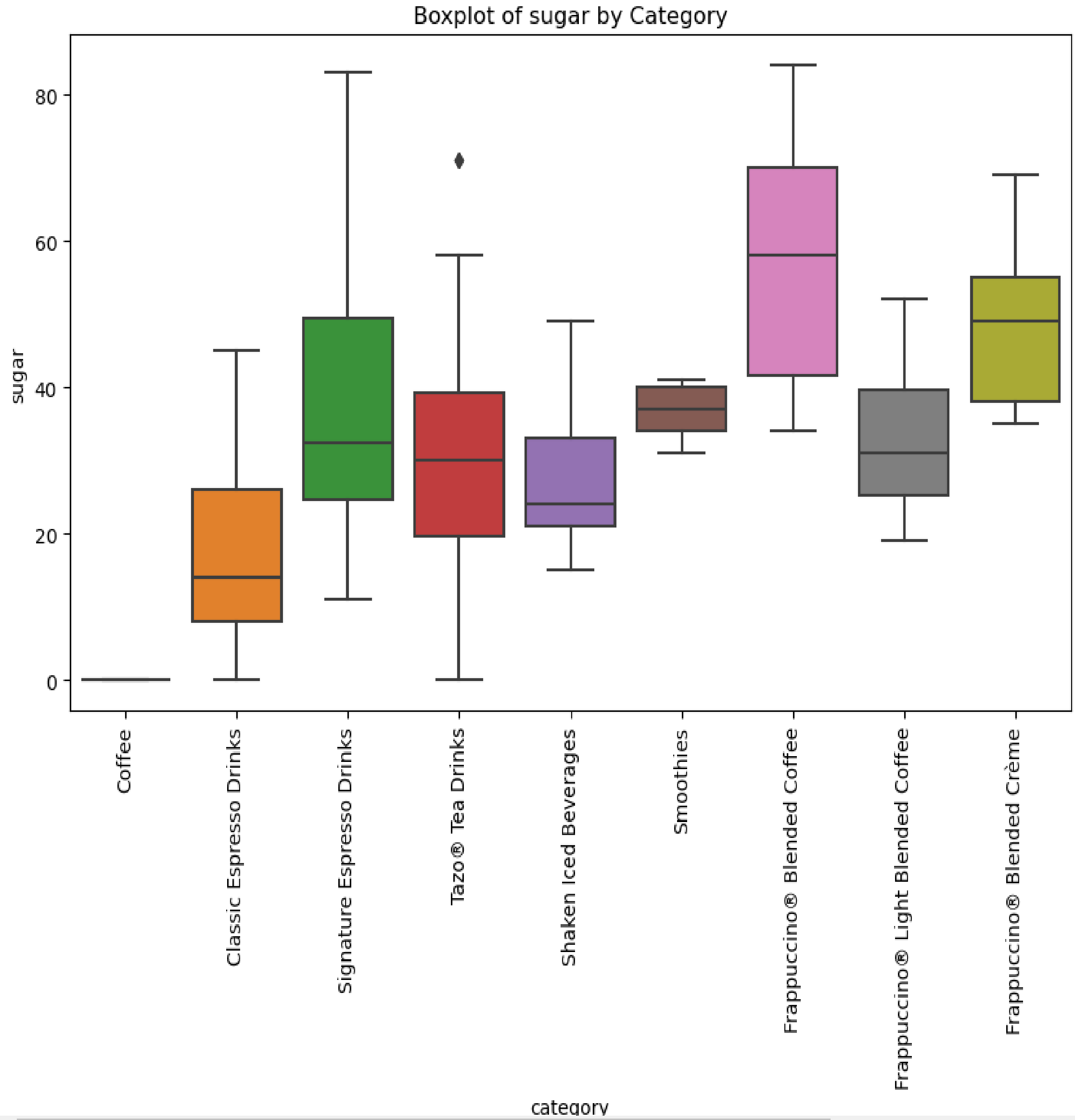
```
>> df['Calories'].mode()

0 150
1 180
2 190
Name: Calories, dtype: int64

>> df['Calories'].mean()

194.30290456431536

>> df['Calories'].median()

190.0

>> print(df.columns)

Index(['category', 'name', 'prep', 'Calories', 'Fat', 'TransFat', ' Carb',
'Cholesterol', ' Sugar', ' Protein', 'Caffeine'], dtype='object')

>> df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
print(df.columns)
# Example usage after renaming
stats = df[['calories', 'fat', 'carb', 'sugar', 'protein', 'caffeine']].describe()
print(stats)
```

```
Index(['category', 'name', 'prep', 'calories', 'fat', 'transfat', 'carb',
       'cholesterol', 'sugar', 'protein', 'caffeine'],
      dtype='object')
         calories        carb       sugar     protein
count  241.000000  241.000000  241.000000  241.000000
mean   194.302905  129.315353   33.024896    6.999170
std    102.858173   82.200315   19.747558    4.871165
min      0.000000    0.000000    0.000000    0.000000
25%    120.000000   70.000000   18.000000    3.000000
50%    190.000000  125.000000   32.000000    6.000000
75%    260.000000  170.000000   44.000000   10.000000
max    510.000000  340.000000   84.000000   20.000000
```

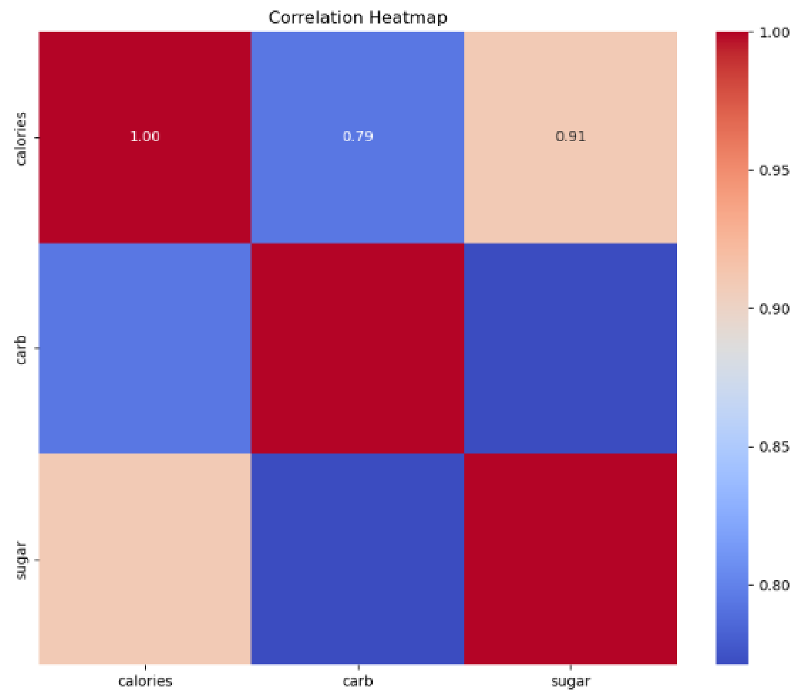
```
>> import matplotlib.pyplot as plt
import seaborn as sns
# Plot histograms for numerical features
df[['calories', 'fat', 'carb', 'sugar', 'protein', 'caffeine']].hist(bins=20,
figsize=(12, 10))
plt.tight_layout()
plt.show()
```



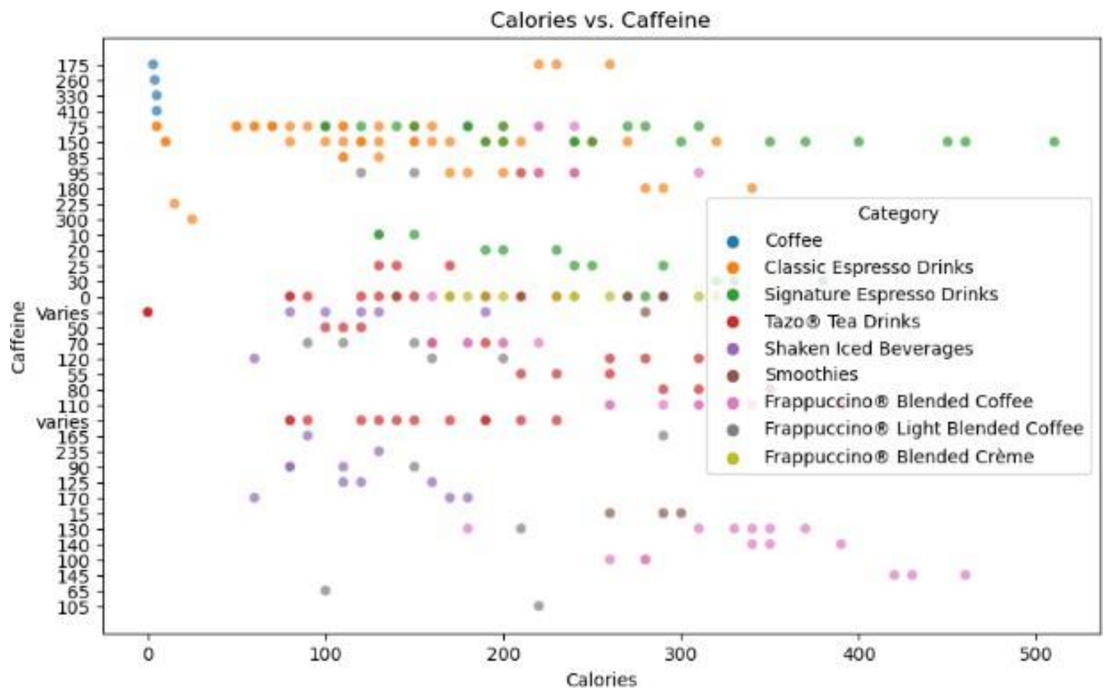

```
#Plot boxplots for numerical features by beverage category

>> features = [ 'sugar']
for feature in features:
      plt.figure(figsize=(10, 6))
      sns.boxplot(x='category', y=feature, data=df)
      plt.title(f'Boxplot of {feature} by Category')
      plt.xticks(rotation=90)
      plt.show()
```



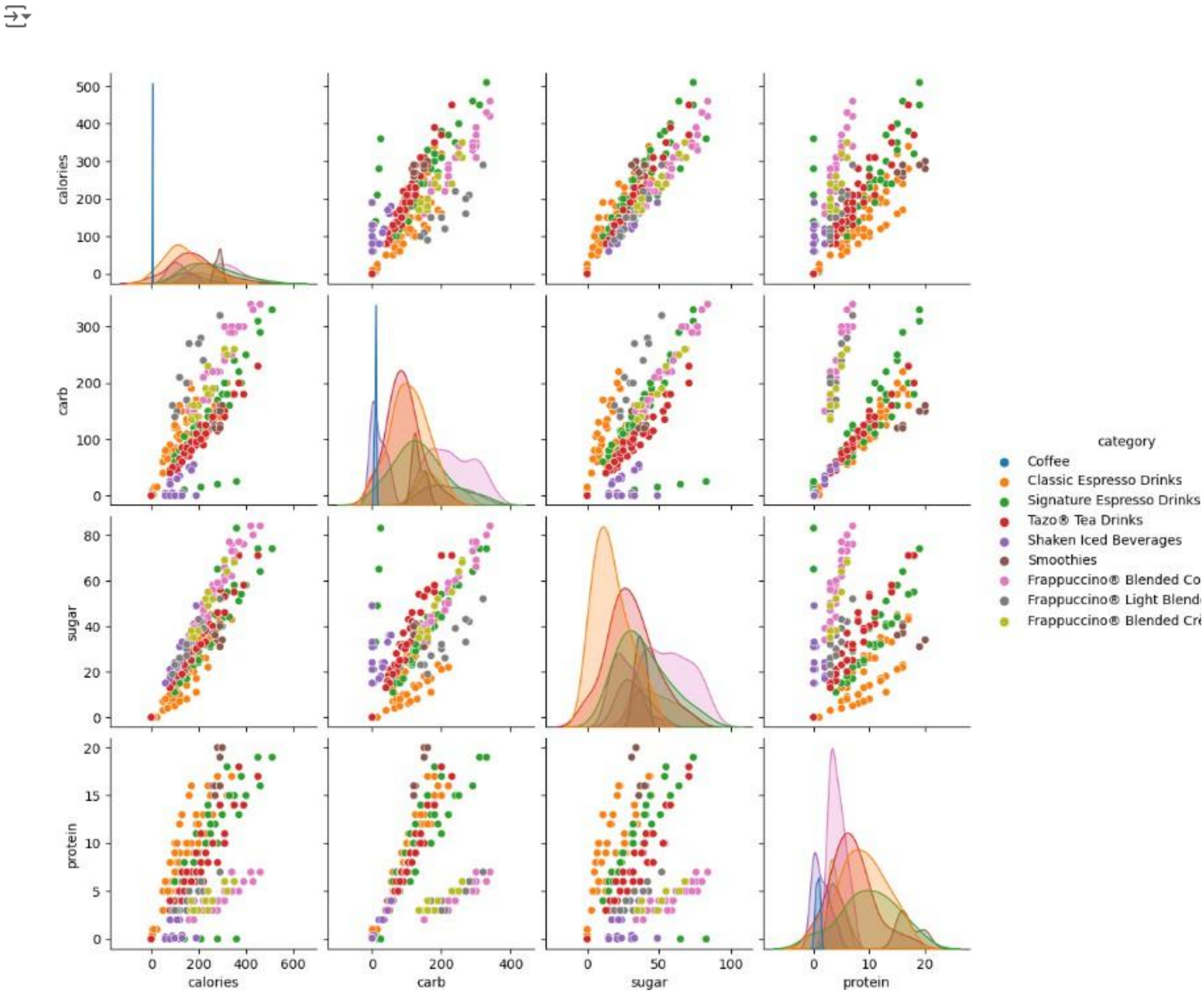

```
#Compute the correlation matrix
>> corr = df[['calories', 'carb', 'sugar' ]].corr()
# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```




```
>> plt.figure(figsize=(10, 6))
sns.scatterplot(x='calories', y='caffeine', data=df, hue='category', alpha=0.7)
plt.title('Calories vs. Caffeine')
plt.xlabel('Calories')
plt.ylabel('Caffeine')
plt.legend(title='Category')
plt.show()
```



```
# Pairplot to show pairwise relationships
>> sns.pairplot(df[['calories', 'fat', 'carb', 'sugar', 'protein', 'caffeine',
'category']], hue='category')
 plt.show()
```



Dt: 20/08/2024

Experiment-05
Geospatial Data visualization using choropleth map

Aim:- To visualize geospatial data using choropleth map

- A choropleth map is a type of thematic map in which a set of pre-defined areas is colored or paerned in propoion to a statistical variable that represents an aggregate summary of a geographic characteristic within each area, such as population density or percapita income.
- In simpler words, it displays divided geographical areas or regions that are colored, shaded, or paerned according to a data variable.
- **Syntax** –plotly.express.choropleth((data_frame=None, lat=None, lon=None, locations=None, locationmode=None, geojson=None, color=None, scope=None, center=None, title=None, width=None, height=None)

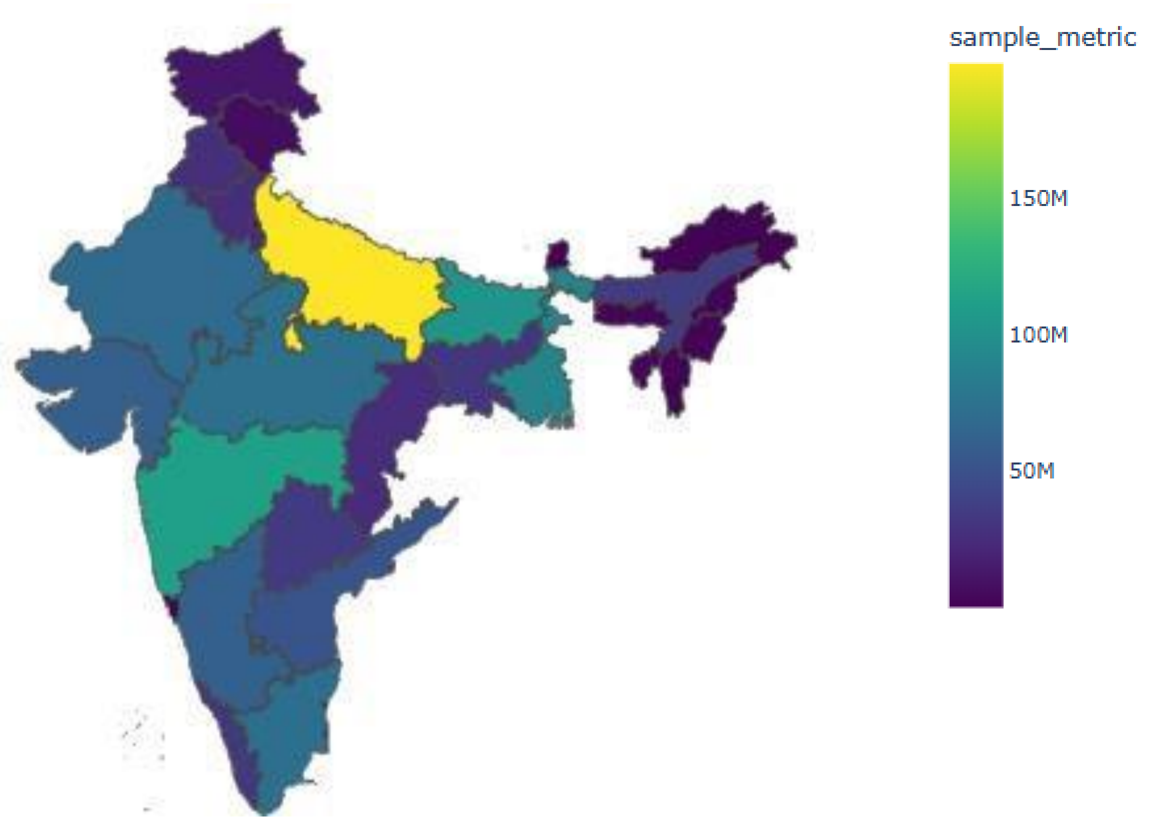
Creating a Choropleth Map of Indian States and Union Territories Using Sample Data for Visualization

```
import pandas as pd
import plotly.express as px
import requests

url = "https://raw.githubusercontent.com/geohacker/india/master/state/india_telengana.geojson"
geojson_data = requests.get(url).json()

data = pd.DataFrame({
    "state": [
        "Andhra Pradesh", "Arunachal Pradesh", "Assam", "Bihar", "Chhattisgarh",
        "Goa", "Gujarat", "Haryana", "Himachal Pradesh", "Jharkhand", "Karnataka",
        "Kerala", "Madhya Pradesh", "Maharashtra", "Manipur", "Meghalaya", "Mizoram",
        "Nagaland", "Odisha", "Punjab", "Rajasthan", "Sikkim", "Tamil Nadu",
        "Telangana", "Tripura", "Uttar Pradesh", "Uttarakhand", "West Bengal",
        "Andaman and Nicobar Islands", "Chandigarh", "Dadra and Nagar Haveli and Daman and
Diu",
        "Delhi", "Jammu and Kashmir", "Ladakh", "Lakshadweep", "Puducherry"
    ],
    "sample_metric": [
        52221000, 1504000, 35571000, 104099000, 25545000, 1458500, 60439692,
        25353000, 6865000, 32988000, 61095297, 33406061, 72627000, 112374333,
        2855794, 2964000, 1097206, 1978500, 41947000, 27743000, 68548437,
        610577, 72147030, 35193978, 3673900, 199812341, 10086292, 91276115,
        380581, 1055450, 585764, 16787941, 12267032, 274000, 64473, 1247953
    ]
})

fig = px.choropleth(
    data,
    geojson=geojson_data,
    locations="state",
    featureidkey="properties.NAME_1",
    color="sample_metric",
    hover_name="state",
    color_continuous_scale="Viridis",
    title="Sample Metric by State and Union Territory in India"
```

```
)

fig.update_geos(fitbounds="locations", visible=False)
fig.show()
```

OUTPUT

Sample Metric by State and Union Territory in India



Dt: 27/08/2024

Experiment-06

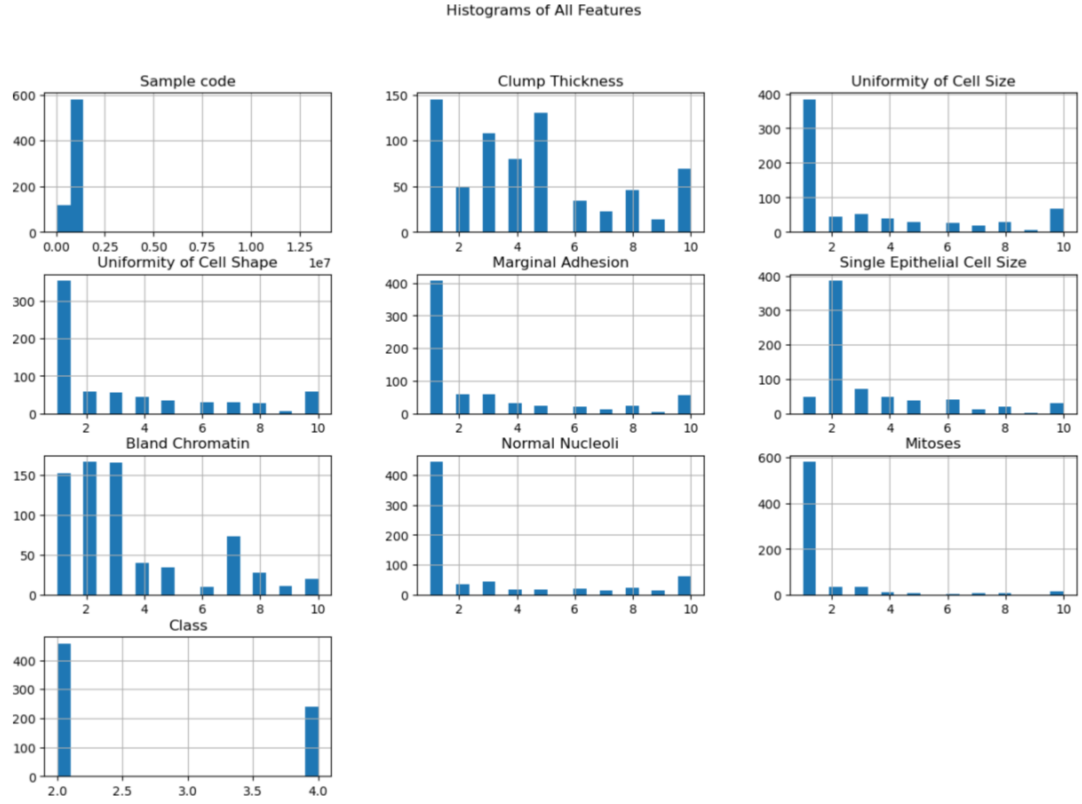
Preprocessing

Aim:- To perform data preprocessing on a given dataset.

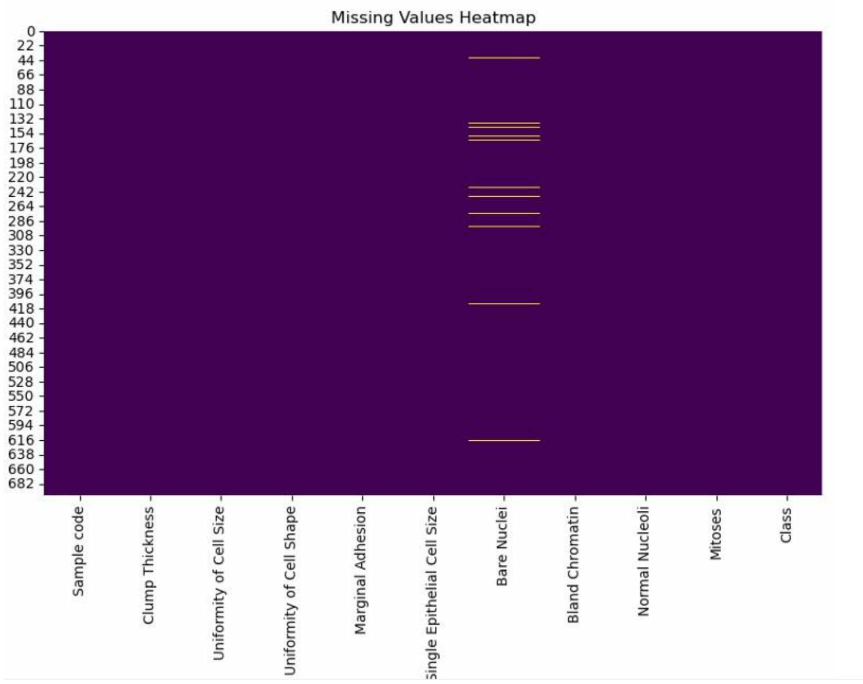
```
>> import pandas as pd
data = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data',
heade data.columns = ['Sample code', 'Clump Thickness', 'Uniformity of Cell Size',
'Uniformity of Cell Shape', 'Marginal Adhesion', 'Single Epithelial Cell Size',
'Bare Nuclei', 'Bland Chromatin', 'Normal Nucleoli', 'Mitoses','Class']
data.head()

import numpy as np

data = data.replace('?',np.nan)
print('Number of instances = %d' % (data.shape[0]))
print('Number of attributes = %d' % (data.shape[1]))
print('Number of missing values:')
for col in data.columns:
      print('\t%s: %d' % (col,data[col].isna().sum()))
data2 = data['Bare Nuclei'] print (data2)
data2.fillna(0)
```

```
Number of instances = 699
Number of attributes = 11
Number of missing values:
      Sample code: 0
      Clump Thickness: 0
      Uniformity of Cell Size: 0
      Uniformity of Cell Shape: 0
      Marginal Adhesion: 0
      Single Epithelial Cell Size: 0
      Bare Nuclei: 16
      Bland Chromatin: 0
      Normal Nucleoli: 0
      Mitoses: 0
      Class: 0
      0 1
      1 10
      2 2
      3 4
      4 1
      ..
      694 2
      695 1
      696 3
      697 4
      698 5
Name: Bare Nuclei, Length: 699, dtype: object
0 1
1 10
2 2
3 4
4 1
..
694 2
695 1
696 3
697 4
698 5
Name: Bare Nuclei, Length: 699, dtype: object
```

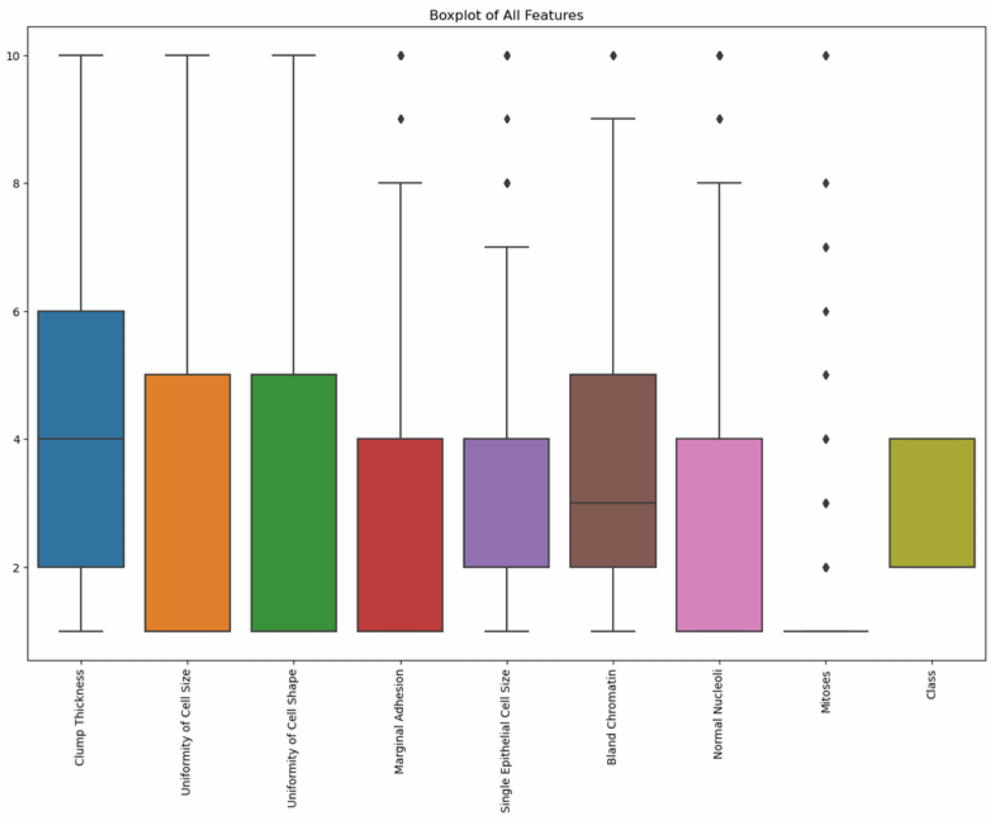
```
>> import matplotlib.pyplot as plt
import seaborn as sns
data.hist(bins=20, figsize=(15, 10))
plt.suptitle('Histograms of All Features')
plt.show()
```



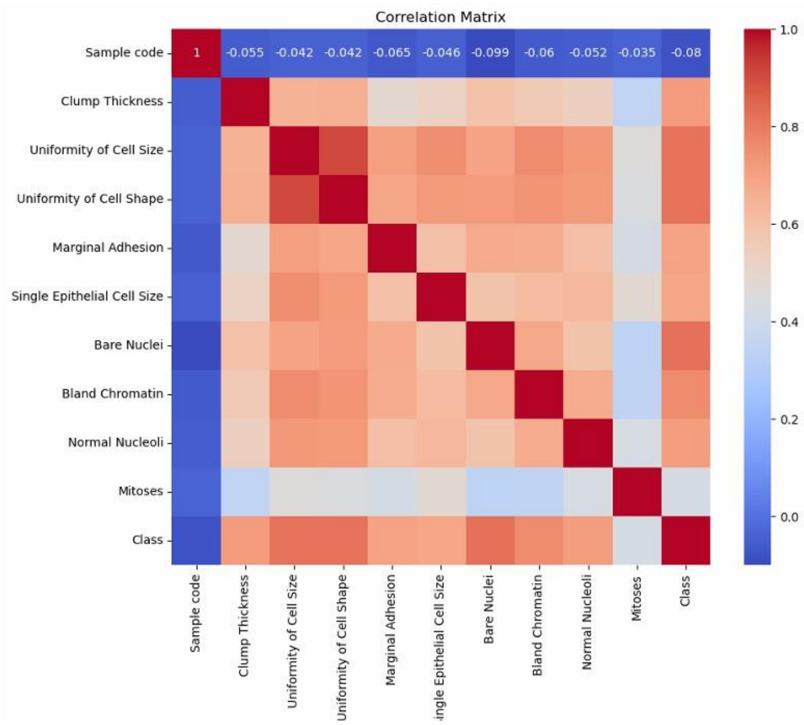

```
>>  plt.figure(figsize=(10, 6))
    sns.heatmap(data.isnull(), cbar=False, cmap='viridis')
    plt.title('Missing Values Heatmap')
    plt.show()
```



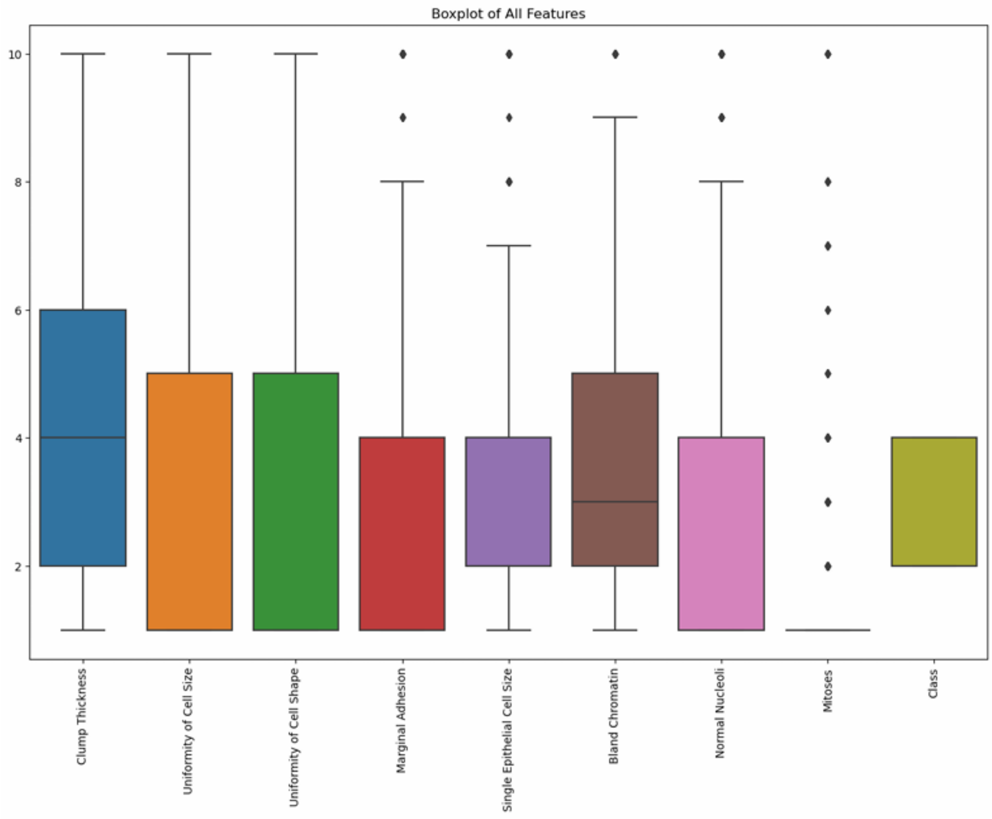
```
>> plt.figure(figsize=(15, 10))
sns.boxplot(data=data.drop('Sample code', axis=1))
plt.xticks(rotation=90)
plt.title('Boxplot of All Features')
plt.show()
```



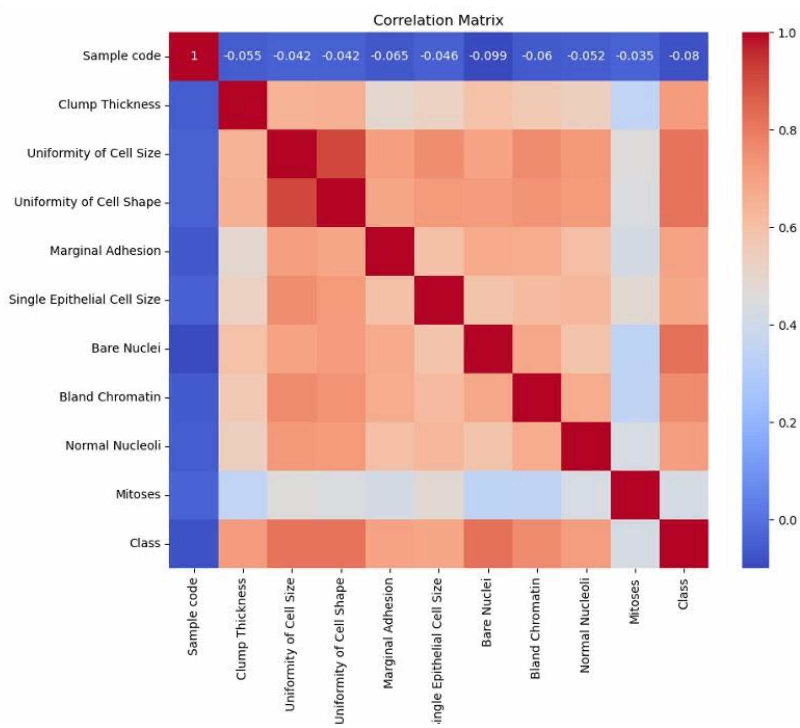

```
>> plt.figure(figsize=(10, 8))
corr = data.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



```
>> plt.figure(figsize=(15, 10))
sns.boxplot(data=data.drop('Sample code', axis=1))
plt.xticks(rotation=90)
plt.title('Boxplot of All Features')
plt.show()
```




```
>> plt.figure(figsize=(10, 8))
corr = data.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



```
>> sns.pairplot(data.dropna(), hue='Class')
plt.show()
```

Dt: 17/09/2024

Experiment-07

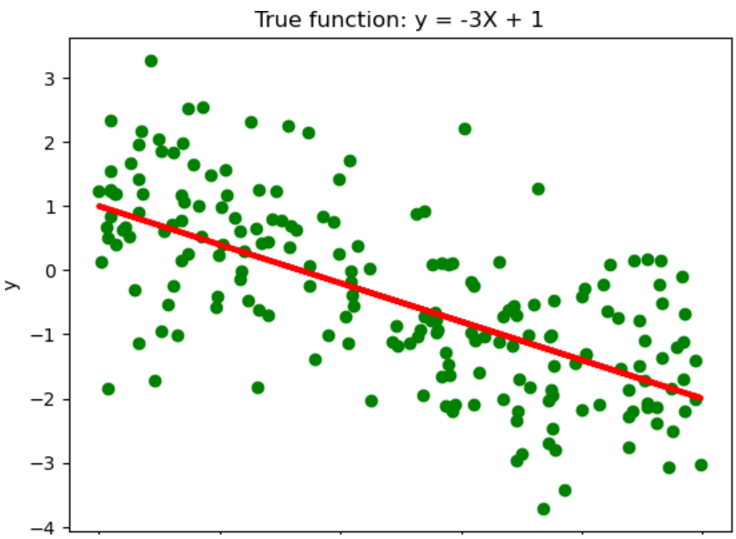
Linear Regression Analysis

Aim:- To perform Simple Linear Regression and Multiple Linear Regression.

1) Generate a random 1-dimensional vector of predictor variables, x, from a uniform distribution.

```
>> %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
seed =1# seed for random number generation
numInstances=200# number of data instances
np.random.seed(seed)
X =np.random.rand(numInstances,1).reshape(-1,1)
y_true=-3*X +1
y =y_true+np.random.normal(size=numInstances).reshape(-
1,1)
plt.scatter(X, y,color='green')
plt.plot(X,y_true,color='red', linewidth=3)
plt.title('True function: y = -3X + 1')
plt.xlabel('X')
plt.ylabel('y')
```

Text(0, 0.5, 'y')

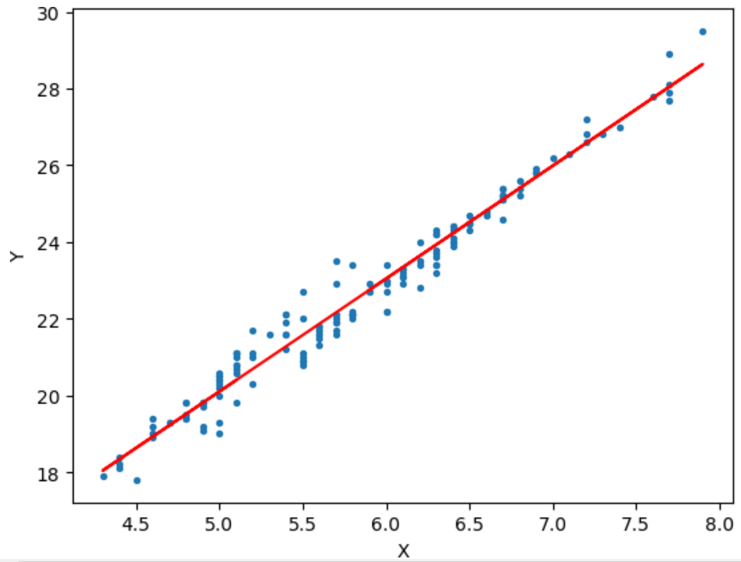


2)Illustrate how to use Python scikit-learn package to fit a **multiple linear regression (MLR) model**.

```
>> import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
```

```
import pandas as pd
import statsmodels.api as sm
data=pd.read_csv("/home/anits/Downloads/IRIS.csv")


x=np.array(data["sepal_length"]).reshape(150,1)
c=np.array(data["sepal_width"]).reshape(150,1)
y=2+3*x+c
reg_model=LinearRegression()
reg_model.fit(x,y)
#lr_1 = sm.OLS(x,y).fit()
y_predicted=reg_model.predict(x)
msr=mean_squared_error(y,y_predicted)
r2=r2_score(y,y_predicted)
print("The Coefficient is ",reg_model.coef_)
print("The Intercept is ",reg_model.intercept_)
print("The Mean Squared Error is ",msr)
print("The R^2 Error is ",r2)
plt.scatter(x,y,s=8)
plt.plot(x,y_predicted,color="red")
plt.xlabel("X")
plt.ylabel("Y")
```

```
The Coefficient is  [[2.94273177]]
The Intercept is  [5.38863738]
The Mean Squared Error is  0.18451682376035183
The R^2 Error is  0.9696658610843356
Text(0, 0.5, 'Y')
```



3) **Ordinary Least Squares (OLS) regression**

```
>> import statsmodels.formula.api as smf
lr_1 =sm.OLS(x,y).fit()
lr_1.summary()
```

OLS Regression Results

Dep. Variable:	y	**R-squared (uncentered):**	0.999
Model:	OLS	**Adj. R-squared (uncentered):**	0.999
Method:	Least Squares	**F-statistic:**	1.021e+05
Date:	Tue, 17 Sep 2024	**Prob (F-statistic):**	3.25e-213
Time:	10:18:44	**Log-Likelihood:**	10.691
No. Observations:	150	**AIC:**	-19.38
Df Residuals:	149	**BIC:**	-16.37
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
x1	0.2596	0.001	319.461	0.000	0.258	0.261

Omnibus:	24.734	**Durbin-Watson:**	0.421
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6.886
Skew:	-0.154	**Prob(JB):**	0.0320
Kurtosis:	1.996	**Cond. No.**	1.00

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

ANOVA

1) Performing **one way ANOVA** with Statsmodels

```
>> import pandas as pd
import numpy as np
from statsmodels.formula.api import ols
import statsmodels.api as sm
# Sample data
data = {
   'group': ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'C', 'C'],
    'value': [10, 12, 11, 15, 16, 14, 20, 22, 21] }
df = pd.DataFrame(data)
# Fit the model
model = ols('value ~ C(group)', data=df).fit()
# Perform ANOVA
anova_table = sm.stats.anova_lm(model, typ=1)
print(anova_table)
```

	df	sum_sq	mean_sq	F	PR(>F)
C(group)	2.0	152.0	76.0	76.0	0.000055
Residual	6.0	6.0	1.0	NaN	NaN

2) **Two way ANOVA**

```
>> import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols
# Sample data
data = {
        'factor1': ['A', 'A', 'A', 'B', 'B', 'B', 'C', 'C', 'C'] * 3,
        'factor2': ['X', 'Y', 'Z'] * 9,
        'value': [10, 12, 14, 15, 17, 19, 20, 22, 24, 11, 13, 15, 16, 18, 20,
21, 23, 25, 12, 14, 16, 17, 19,   21, 22, 24, 26] }
df = pd.DataFrame(data)
# Fit the model
model = ols('value ~ C(factor1) * C(factor2)', data=df).fit()
# Perform ANOVA
anova_table = sm.stats.anova_lm(model, typ=2)
print(anova_table)
```

	sum_sq	df	F	PR(>F)
C(factor1)	4.500000e+02	2.0	2.250000e+02	1.841789e-13
C(factor2)	7.200000e+01	2.0	3.600000e+01	5.120000e-07
C(factor1):C(factor2)	4.638502e-28	4.0	1.159626e-28	1.000000e+00
Residual	1.800000e+01	18.0	NaN	NaN

Dt: 15/10/2024

Experiment-08

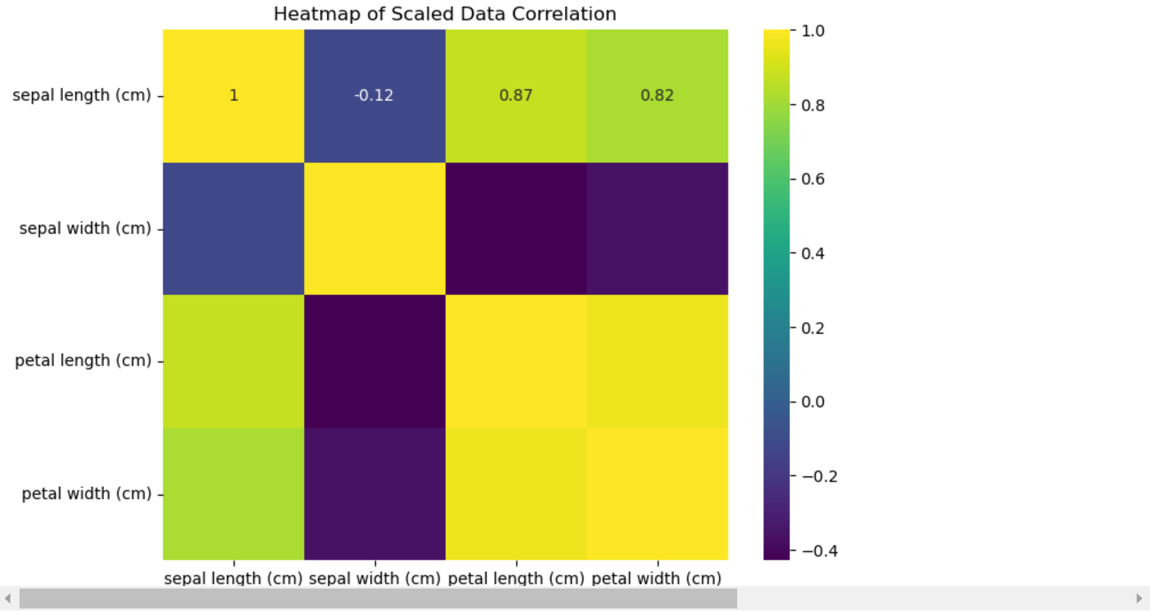
Dimensionality Reduction using PCA

Aim:- To perform dimensionality reduction operation using PCA on a Dataset.

1) Apply **Principal Component Analysis (PCA)** to identify the combination of attributes (principal components, or directions in the feature space) that account for the most variance in data.

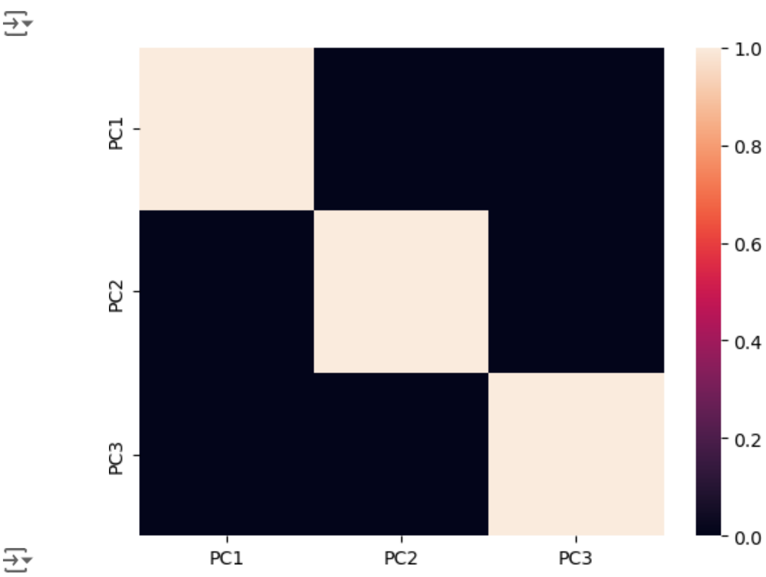
```
>> import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler  # Corrected class name

# Load the Iris dataset
iris = datasets.load_iris()
df = pd.DataFrame(iris['data'], columns=iris['feature_names'])

print(df.head())

scaler = StandardScaler()

# Scale the data
scaled_data = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
print(scaled_data.head())

# Generate a heatmap of the scaled data correlation
plt.figure(figsize=(8, 6))
sns.heatmap(scaled_data.corr(), annot=True, cmap='viridis')
plt.title('Heatmap of Scaled Data Correlation')
plt.show()
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
0  -0.900681           1.019004         -1.340227          -1.315444
1  -1.143017          -0.131979         -1.340227          -1.315444
2  -1.385353           0.328414         -1.397064          -1.315444
3  -1.506521           0.098217         -1.283389          -1.315444
4  -1.021849           1.249201         -1.340227          -1.315444
```



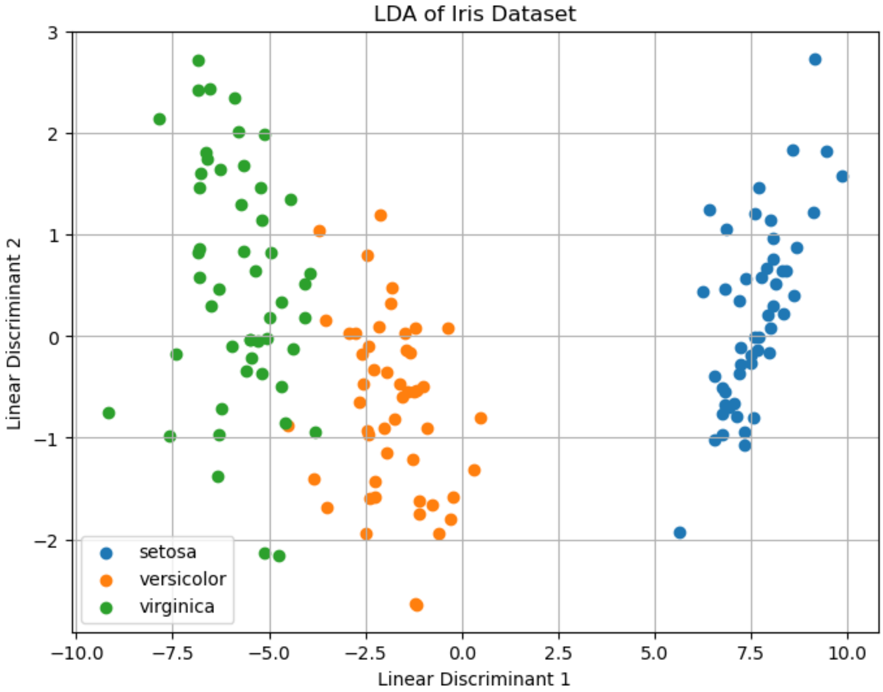
2) Apply **Principal Component Analysis (PCA)**

```
pca=PCA(n_components=3)
pca.fit(scaled_data)
data_pca=pca.transform(scaled_data)
data_pca=pd.DataFrame(data_pca,columns=['PC1','PC2','PC3'])
data_pca.head()
sns.heatmap(data_pca.corr())
```



3) Apply **Linear Discriminant Analysis (LDA)**

```
>> import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Target labels
target_names = iris.target_names

# Apply LDA
lda = LDA(n_components=2)
X_lda = lda.fit_transform(X, y)

# Explained variance ratio
print("Explained variance ratio (LDA):", lda.explained_variance_ratio_)

# Plotting LDA results
plt.figure(figsize=(8, 6))
for i, target_name in zip(range(len(target_names)), target_names):
    plt.scatter(X_lda[y == i, 0], X_lda[y == i, 1], label=target_name)
plt.title('LDA of Iris Dataset')
plt.xlabel('Linear Discriminant 1')
plt.ylabel('Linear Discriminant 2')
plt.legend()
plt.grid()
plt.show()
```



Dt: 22/10/2024

Experiment-09

K- Means Clustering

Aim:- To perform K-Means clustering operation and visualize the clusters

Perform k-means clustering on a toy example of the Iris flower dataset based on sepal length and width measurements.

```
>>
    import pandas as pd
    import numpy as np
    from sklearn.datasets import load_iris
    from sklearn.cluster import KMeans
    from sklearn.preprocessing import StandardScaler
    from tabulate import tabulate

    iris = load_iris()
    X = iris.data
    y = iris.target
    feature_names = iris.feature_names

    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    kmeans = KMeans(n_clusters=3, random_state=42)
    kmeans.fit(X_scaled)

    clusters = kmeans.labels_

    df = pd.DataFrame(X, columns=feature_names)
    df['cluster'] = clusters

    summary_stats = df.groupby('cluster').agg(['mean', 'median', 'var', 'count'])

    for feature in feature_names:
        print(f"\n===Summary Statistics for {feature}===")
        feature_stats = summary_stats[feature]
        print(tabulate(feature_stats, headers='keys', tablefmt='pretty'))

    inertia = kmeans.inertia_
    print(f'\nInertia: {inertia:.4f}')

    cluster_sizes = pd.Series(clusters).value_counts().sort_index()
    print("\n===Cluster Sizes===")
    print(tabulate(cluster_sizes.reset_index(), headers=['Cluster', 'Size'],
    tablefmt='pretty'))
```

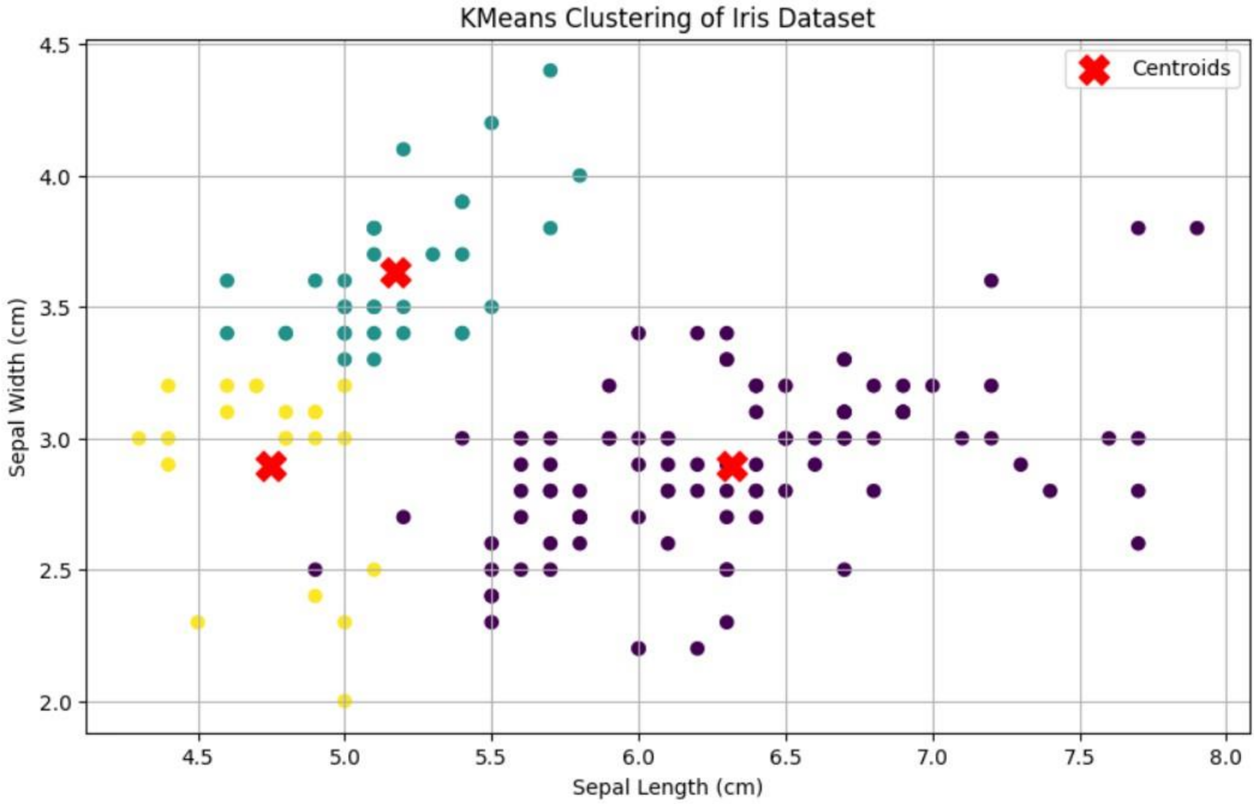
OUTPUT

```
===Summary Statistics for sepal length (cm)===
+---------+-------------------+--------+----------------------+-------+
| cluster |       mean        | median |         var          | count |
+---------+-------------------+--------+----------------------+-------+
|    0    | 6.314583333333334 |  6.3   |  0.3877850877192982  | 96.0  |
|    1    | 5.16969696969697  |  5.1   | 0.08342803030303036  | 33.0  |
|    2    | 4.747619047619048 |  4.8   | 0.057619047619047695 | 21.0  |
```

```
+--------------+-----------------------------+------------+-------------------------------+----------+

===Summary Statistics for sepal width (cm)===
+--------------+-----------------------------+------------+-------------------------------+----------+
| cluster |         mean          | median |          var           | count |
+--------------+-----------------------------+------------+-------------------------------+----------+
|    0    | 2.8958333333333335 |  2.9   | 0.09956140350877189 | 96.0  |
|    1    | 3.6303030303030304 |  3.5   | 0.0734280303030303  | 33.0  |
|    2    | 2.895238095238095  |  3.0   | 0.13047619047619058 | 21.0  |
+--------------+-----------------------------+------------+-------------------------------+----------+

===Summary Statistics for petal length (cm)===
+--------------+-----------------------------+------------+---------------------------------+----------+
| cluster |         mean          | median |           var            | count |
+--------------+-----------------------------+------------+---------------------------------+----------+
|    0    | 4.973958333333333  |  4.9   |  0.5922620614035091  | 96.0  |
|    1    | 1.493939393939394  |  1.5   | 0.033087121212121214 | 33.0  |
|    2    | 1.7571428571428571 |  1.4   |  0.5915714285714286  | 21.0  |
+--------------+-----------------------------+------------+---------------------------------+----------+

===Summary Statistics for petal width (cm)===
+--------------+-----------------------------+------------+---------------------------------+----------+
| cluster |         mean          | median |           var            | count |
+--------------+-----------------------------+------------+---------------------------------+----------+
|    0    |       1.703125        |  1.65  | 0.16935855263157887  | 96.0  |
|    1    | 0.2727272727272727 |  0.2   | 0.013295454545454546 | 33.0  |
|    2    | 0.3523809523809524 |  0.2   | 0.11461904761904762  | 21.0  |
+--------------+-----------------------------+------------+---------------------------------+----------+

Inertia: 191.0247

===Cluster Sizes===
+---+-------------+---------+
|   | Cluster | Size |
+---+-------------+---------+
| 0 |    0    |  96  |
| 1 |    1    |  33  |
| 2 |    2    |  21  |
+---+-------------+---------+
```

\>>

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.scatter(df['sepal length (cm)'],df['sepal width
(cm)'],c=clusters,cmap='viridis',marker='0')
plt.title('KMeans Clustering of Iris Dataset')
plt.xlabel('Sepal Length(cm)')
plt.ylabel('Sepal Width(cm)')

centroids=kmeans.cluster_centers_
centroids_original=scaler.inverse_transform(centroids)
plt.scatter(centroids_original[:,0],centroids_original[:,1],c='red',marker='X',s=20
0,label='centroids')

plt.legend()
plt.grid()
plt.show()
```


KMeans Clustering of Iris Dataset
Centroids
Sepal Width (cm)
Sepal Length (cm)
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0