Ex.No 1 **Working with Numpy Arrays**

Aim:

To write a Python Program to work with Numpy Arrays

Program:

a) # Python program to demonstrate basic array characteristics

```
import numpy as np

# Creating array object
arr = np.array( [[ 1, 2, 3], [ 4, 2, 5]] )

# Printing type of arr object
print("Array is of type: ", type(arr))

# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)

# Printing shape of array
print("Shape of array: ", arr.shape)

# Printing size (total number of elements) of array
print("Size of array: ", arr.size)

# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)
```

OUTPUT

```
Array is of type:
No. of dimensions:  2
Shape of array:  (2, 3)
Size of array:  6
Array stores elements of type:  int64
```

Program:

b) # Python program to demonstrate array creation techniques

```
import numpy as np

# Creating array from list with type float
a = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')
print ("Array created using passed list:\n", a)

# Creating array from tuple
b = np.array((1 , 3, 2))
print ("\nArray created using passed tuple:\n", b)

# Creating a 3X4 array with all zeros
c = np.zeros((3, 4))
print ("\nAn array initialized with all zeros:\n", c)

# Create a constant value array of complex type
d = np.full((3, 3), 6, dtype = 'complex')
print ("\nAn array initialized with all 6s."
            "Array type is complex:\n", d)

# Create an array with random values
e = np.random.random((2, 2))
print ("\nA random array:\n", e)

# Create a sequence of integers
# from 0 to 30 with steps of 5
f = np.arange(0, 30, 5)
print ("\nA sequential array with steps of 5:\n", f)

# Create a sequence of 10 values in range 0 to 5
g = np.linspace(0, 5, 10)
print ("\nA sequential array with 10 values between"
                                        "0 and 5:\n", g)

# Reshaping 3X4 array to 2X2X3 array
arr = np.array([[1, 2, 3, 4], [5, 2, 4, 2], [1, 2, 0, 1]])

newarr = arr.reshape(2, 2, 3)

print ("\nOriginal array:\n", arr)
print ("Reshaped array:\n", newarr)

# Flatten array
arr = np.array([[1, 2, 3], [4, 5, 6]])
flarr = arr.flatten()

print ("\nOriginal array:\n", arr)
print ("Fattened array:\n", flarr)
```

Output:

```
Array created using passed list:
 [[ 1.  2.  4.]
 [ 5.  8.  7.]]
Array created using passed tuple:
 [1 3 2]
An array initialized with all zeros:
 [[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]

An array initialized with all 6s. Array type is complex:
 [[ 6.+0.j  6.+0.j  6.+0.j]
 [ 6.+0.j  6.+0.j  6.+0.j]
 [ 6.+0.j  6.+0.j  6.+0.j]]

A random array:
 [[ 0.46829566  0.67079389]
 [ 0.09079849  0.95410464]]

A sequential array with steps of 5:
 [ 0  5 10 15 20 25]

A sequential array with 10 values between 0 and 5:
 [ 0.          0.55555556  1.11111111  1.66666667  2.22222222  2.77777778
  3.33333333  3.88888889  4.44444444  5.        ]

Original array:
 [[1 2 3 4]
 [5 2 4 2]
 [1 2 0 1]]
Reshaped array:
 [[[1 2 3]
  [4 5 2]]

 [[4 2 1]
  [2 0 1]]]
Original array:
 [[1 2 3]
 [4 5 6]]
Fattened array:
 [1 2 3 4 5 6]
```

Program

c)
```
# Python program to demonstrate indexing in numpy
import numpy as np

# An exemplar array
arr = np.array([[-1, 2, 0, 4],
                [4, -0.5, 6, 0],
                [2.6, 0, 7, 8],
                [3, -7, 4, 2.0]])

# Slicing array
temp = arr[:2, ::2]
print ("Array with first 2 rows and alternate"
                    "columns(0 and 2):\n", temp)

# Integer array indexing example
temp = arr[[0, 1, 2, 3], [3, 2, 1, 0]]
print ("\nElements at indices (0, 3), (1, 2), (2, 1),"
                                    "(3, 0):\n", temp)

# boolean array indexing example
cond = arr > 0 # cond is a boolean array
temp = arr[cond]
print ("\nElements greater than 0:\n", temp)
```

Output:

```
Array with first 2 rows and alternatecolumns(0 and 2):
 [[-1.  0.]
 [ 4.  6.]]

Elements at indices (0, 3), (1, 2), (2, 1),(3, 0):
 [ 4.  6.  0.  3.]

Elements greater than 0:
 [ 2.   4.   4.   6.   2.6  7.   8.   3.   4.   2. ]
```

Program:

d) # Python program to demonstrate basic operations on single array

```
import numpy as np

a = np.array([1, 2, 5, 3])

# add 1 to every element
print ("Adding 1 to every element:", a+1)

# subtract 3 from each element
print ("Subtracting 3 from each element:", a-3)

# multiply each element by 10
print ("Multiplying each element by 10:", a*10)

# square each element
print ("Squaring each element:", a**2)

# modify existing array
a *= 2
print ("Doubled each element of original array:", a)

# transpose of array
a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])

print ("\nOriginal array:\n", a)
print ("Transpose of array:\n", a.T)
```

Output:

```
Adding 1 to every element: [2 3 6 4]
Subtracting 3 from each element: [-2 -1  2  0]
Multiplying each element by 10: [10 20 50 30]
Squaring each element: [ 1  4 25  9]
Doubled each element of original array: [ 2  4 10  6]

Original array:
 [[1 2 3]
 [3 4 5]
 [9 6 0]]
Transpose of array:
 [[1 3 9]
 [2 4 6]
 [3 5 0]]
```

e) Python program to demonstrate horizontal and vertical stacking

```
import numpy as np

a = np.array([[1, 2],
              [3, 4]])

b = np.array([[5, 6],
              [7, 8]])

# vertical stacking
print("Vertical stacking:\n", np.vstack((a, b)))

# horizontal stacking
print("\nHorizontal stacking:\n", np.hstack((a, b)))

c = [5, 6]

# stacking columns
print("\nColumn stacking:\n", np.column_stack((a, c)))

# concatenation method
print("\nConcatenating to 2nd axis:\n", np.concatenate((a, b), 1))
```

Output:

```
Vertical stacking:
 [[1 2]
 [3 4]
 [5 6]
 [7 8]]

Horizontal stacking:
 [[1 2 5 6]
 [3 4 7 8]]

Column stacking:
 [[1 2 5]
 [3 4 6]]

Concatenating to 2nd axis:
 [[1 2 5 6]
 [3 4 7 8]]
```

f) Python program to demonstrate horizontal and vertical spliting

```
import numpy as np

a = np.array([[1, 3, 5, 7, 9, 11],
            [2, 4, 6, 8, 10, 12]])

# horizontal splitting
print("Splitting along horizontal axis into 2 parts:\n", np.hsplit(a, 2))

# vertical splitting
print("\nSplitting along vertical axis into 2 parts:\n", np.vsplit(a, 2))
```

Result:

Thus the program to implement basic numpy arrays was executed successfully

Output:

```
Splitting along horizontal axis into 2 parts:
 [array([[1, 3, 5],
       [2, 4, 6]]), array([[ 7,  9, 11],
       [ 8, 10, 12]])]

Splitting along vertical axis into 2 parts:
 [array([[ 1,  3,  5,  7,  9, 11]]), array([[ 2,  4,  6,  8, 10, 12]])]
```

EXNO:

DATE: **PANDAS-INDEXING AND SELECTING OPERATIONS**

AIM:

To do the data analysis using pandas package for a csv or excel file.

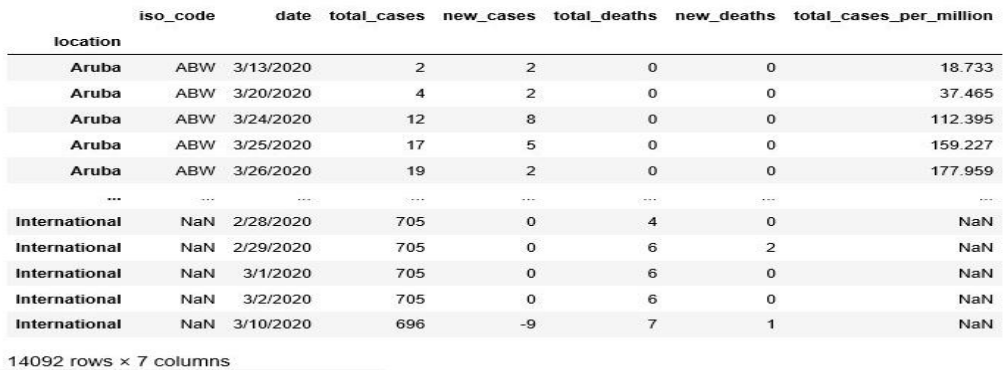
Program:

Reading dataset:

```
import pandas as pd
df=pd.read_csv(" D: \covid.csv")
df
```

a) Make the first column as index:

```
df.set_index("location",inplace=True)
df
```

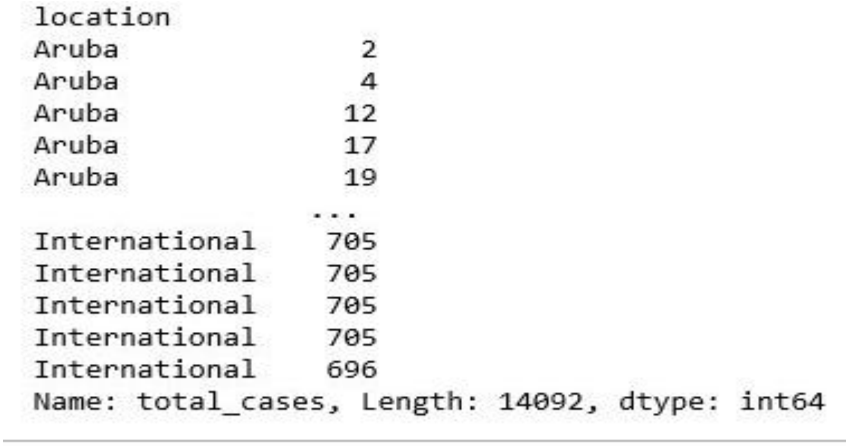


	iso_code	date	total_cases	new_cases	total_deaths	new_deaths	total_cases_per_million
location							
Aruba	ABW	3/13/2020	2	2	0	0	18.733
Aruba	ABW	3/20/2020	4	2	0	0	37.465
Aruba	ABW	3/24/2020	12	8	0	0	112.395
Aruba	ABW	3/25/2020	17	5	0	0	159.227
Aruba	ABW	3/26/2020	19	2	0	0	177.959
...	...	...	...	...	...	...	...
International	NaN	2/28/2020	705	0	4	0	NaN
International	NaN	2/29/2020	705	0	6	2	NaN
International	NaN	3/1/2020	705	0	6	0	NaN
International	NaN	3/2/2020	705	0	6	0	NaN
International	NaN	3/10/2020	696	-9	7	1	NaN

14092 rows × 7 columns

b) Select single column and print the data

```
y=df['total_cases']
y
```



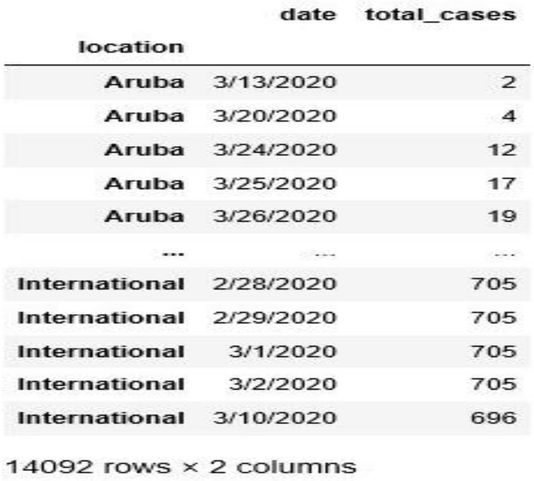
```
location
Aruba              2
Aruba              4
Aruba             12
Aruba             17
Aruba             19
                ...
International    705
International    705
International    705
International    705
International    696
Name: total_cases, Length: 14092, dtype: int64
```

c) Select multiple column and print the data:

```
y=df[["date","total_cases"]]
y
```

d) Select single row and print the last five elements of the data.

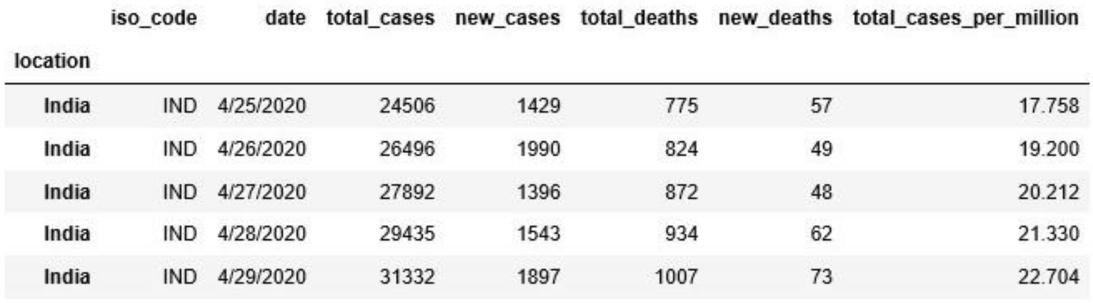
```
x=df.loc['India']
```



	date	total_cases
location		
Aruba	3/13/2020	2
Aruba	3/20/2020	4
Aruba	3/24/2020	12
Aruba	3/25/2020	17
Aruba	3/26/2020	19
...	...	...
International	2/28/2020	705
International	2/29/2020	705
International	3/1/2020	705
International	3/2/2020	705
International	3/10/2020	696

14092 rows × 2 columns

```
x.tail(5)
```

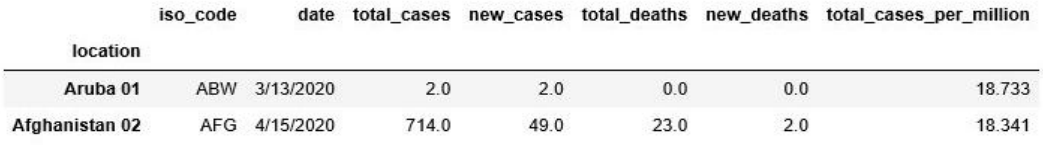


	iso_code	date	total_cases	new_cases	total_deaths	new_deaths	total_cases_per_million
location							
India	IND	4/25/2020	24506	1429	775	57	17.758
India	IND	4/26/2020	26496	1990	824	49	19.200
India	IND	4/27/2020	27892	1396	872	48	20.212
India	IND	4/28/2020	29435	1543	934	62	21.330
India	IND	4/29/2020	31332	1897	1007	73	22.704

e) Select multiple rows and print the first five elements of the data

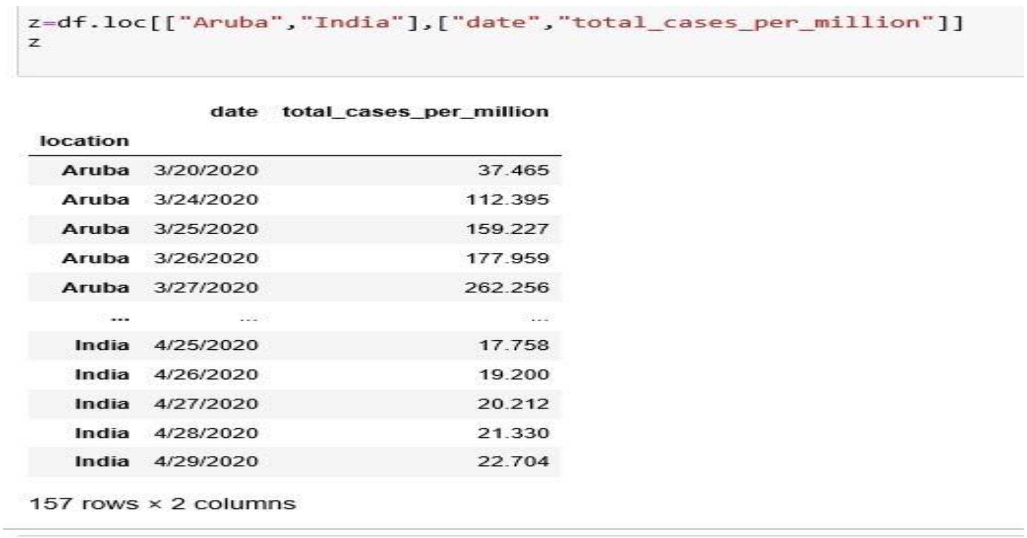
```
x=df.loc[["Aruba 01","Afghanistan 02"]]
x.head(5)
```

f) Select multiple rows and columns from the data set and print it.



	iso_code	date	total_cases	new_cases	total_deaths	new_deaths	total_cases_per_million
location							
Aruba 01	ABW	3/13/2020	2.0	2.0	0.0	0.0	18.733
Afghanistan 02	AFG	4/15/2020	714.0	49.0	23.0	2.0	18.341

```
z=df.loc[["Aruba","India"],["date","total_cases_per_million"]]
z
```



```
z=df.loc[["Aruba","India"],["date","total_cases_per_million"]]
z
```

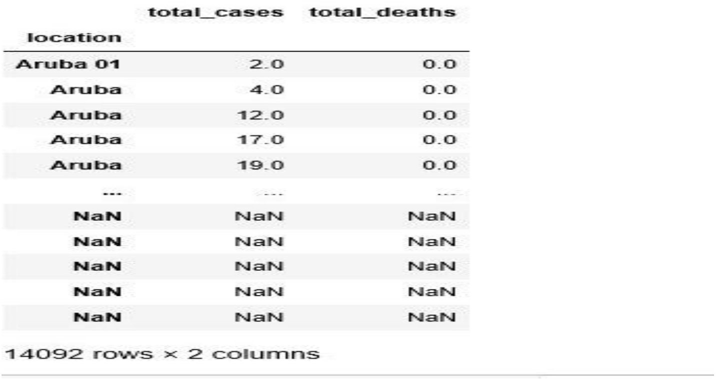
	date	total_cases_per_million
location		
Aruba	3/20/2020	37.465
Aruba	3/24/2020	112.395
Aruba	3/25/2020	159.227
Aruba	3/26/2020	177.959
Aruba	3/27/2020	262.256
...	...	...
India	4/25/2020	17.758
India	4/26/2020	19.200
India	4/27/2020	20.212
India	4/28/2020	21.330
India	4/29/2020	22.704

157 rows × 2 columns

g) Select all the rows and some columns (more than two) from the data set and print it.

```
R=df.loc[:,["total_cases","total_deaths"]]
R
```

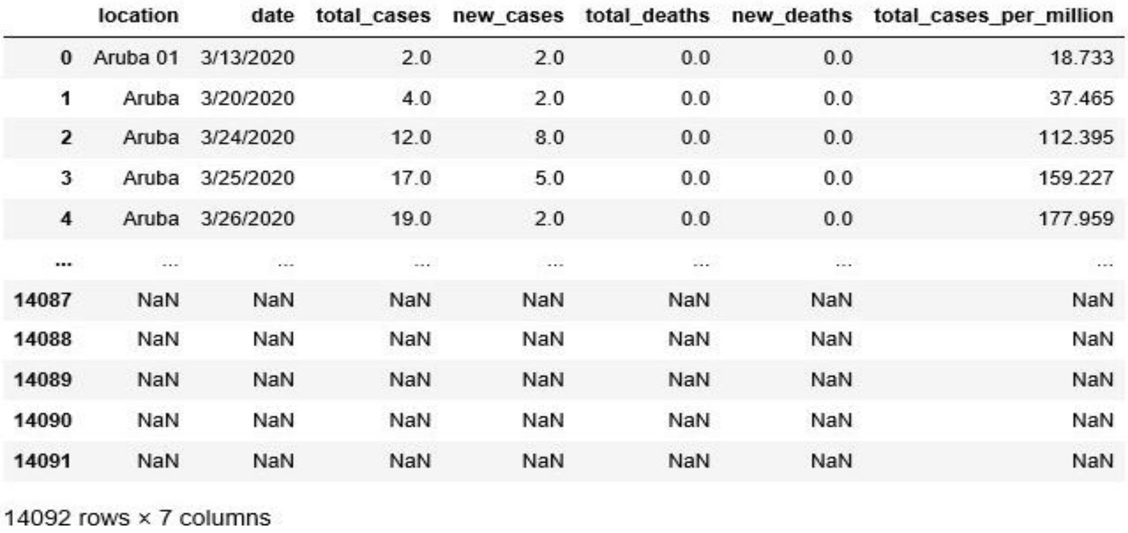
h) Print the same data set again and delete the first column from the data set and print it.



	total_cases	total_deaths
location		
Aruba 01	2.0	0.0
Aruba	4.0	0.0
Aruba	12.0	0.0
Aruba	17.0	0.0
Aruba	19.0	0.0
...	...	...
NaN	NaN	NaN
NaN	NaN	NaN
NaN	NaN	NaN
NaN	NaN	NaN
NaN	NaN	NaN

14092 rows × 2 columns

```
import pandas as pd
data=pd.read_csv("D:/owid-covid-data1.csv")
d=data.drop(["iso_code"],axis=1)
d
```

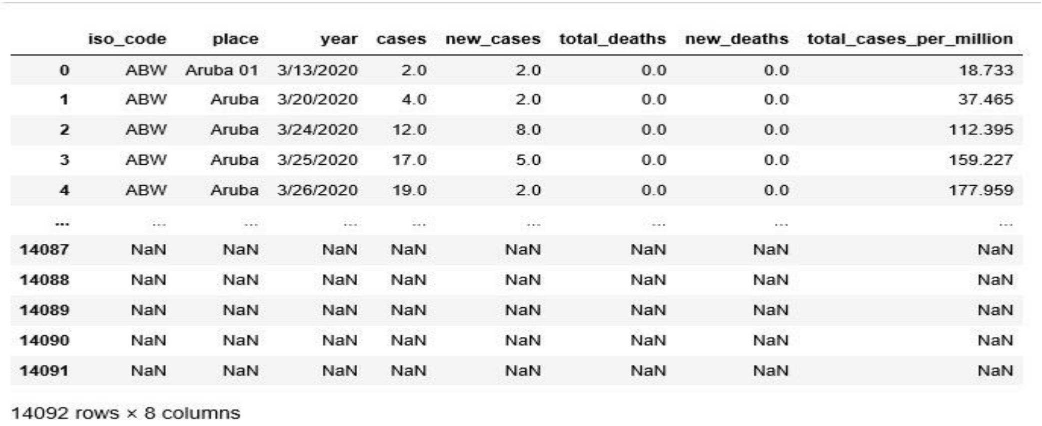


	location	date	total_cases	new_cases	total_deaths	new_deaths	total_cases_per_million
0	Aruba 01	3/13/2020	2.0	2.0	0.0	0.0	18.733
1	Aruba	3/20/2020	4.0	2.0	0.0	0.0	37.465
2	Aruba	3/24/2020	12.0	8.0	0.0	0.0	112.395
3	Aruba	3/25/2020	17.0	5.0	0.0	0.0	159.227
4	Aruba	3/26/2020	19.0	2.0	0.0	0.0	177.959
...	...	...	...	...	...	...	...
14087	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14088	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14089	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14090	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14091	NaN	NaN	NaN	NaN	NaN	NaN	NaN

14092 rows × 7 columns

i) Change the 1st, 2nd and 3rd columns name and print it

```
import pandas as pd
df=pd.read_csv("D:/owid-covid-data1.csv")
data=df.rename(columns={"location":"place","date":"year","total_cases ":"cases"})
data
```



	iso_code	place	year	cases	new_cases	total_deaths	new_deaths	total_cases_per_million
0	ABW	Aruba 01	3/13/2020	2.0	2.0	0.0	0.0	18.733
1	ABW	Aruba	3/20/2020	4.0	2.0	0.0	0.0	37.465
2	ABW	Aruba	3/24/2020	12.0	8.0	0.0	0.0	112.395
3	ABW	Aruba	3/25/2020	17.0	5.0	0.0	0.0	159.227
4	ABW	Aruba	3/26/2020	19.0	2.0	0.0	0.0	177.959
...	...	...	...	...	...	...	...	...
14087	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14088	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14089	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14090	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14091	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

14092 rows × 8 columns

Result:

Thus the program to implement pandas was executed successfully

EXNO:

PANDAS-MERGING OPERATIONS

DATE:

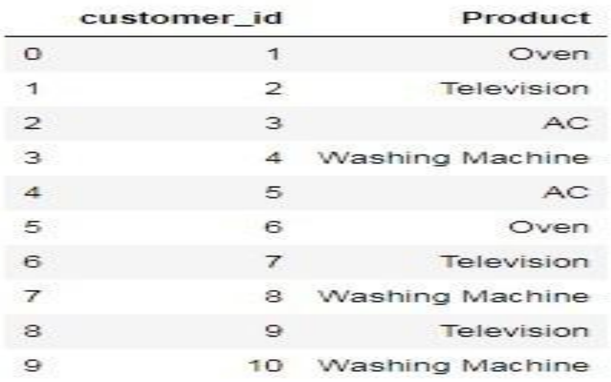
Merge the two data sets (any two csv files) and perform the following join operations
a)Natural join
b) Full outer join
c) Left outer join
d) Right outer join

AIM:

To do the merging/joining operations for two csv files.

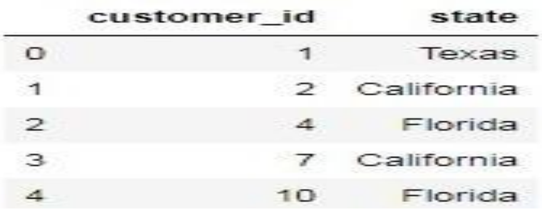
Program:
Reading dataset:

```
import pandas as pd
df1=pd.read_csv("D:/df1.csv")
df1
```



	customer_id	Product
0	1	Oven
1	2	Television
2	3	AC
3	4	Washing Machine
4	5	AC
5	6	Oven
6	7	Television
7	8	Washing Machine
8	9	Television
9	10	Washing Machine

```
import pandas as pd
df2=pd.read_csv("D: \df2.csv")
df2
```



	customer_id	state
0	1	Texas
1	2	California
2	4	Florida
3	7	California
4	10	Florida

a)Natural Join:

Natural join keeps only rows that match from the data frames (df1 and df2).

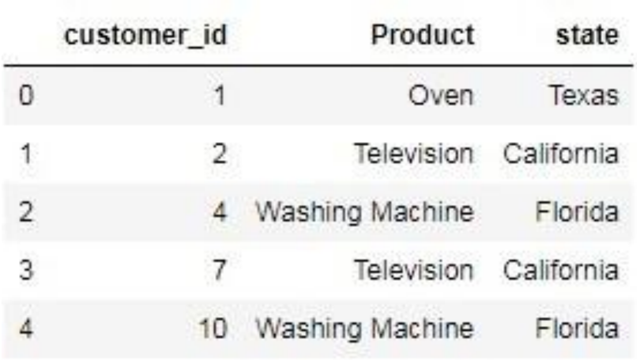
SYNTAX:

Pd.merge(df1,df2,on=column, how='inner')

Return only the rows in which the left table have matching keys in the right table.

Code: pd.merge(df1,df2, o n= 'customer_id', how='inner')

b)Full outer join:



	customer_id	Product	state
0	1	Oven	Texas
1	2	Television	California
2	4	Washing Machine	Florida
3	7	Television	California
4	10	Washing Machine	Florida

Full outer join keeps all rows from both data frames.

SYNTAX:

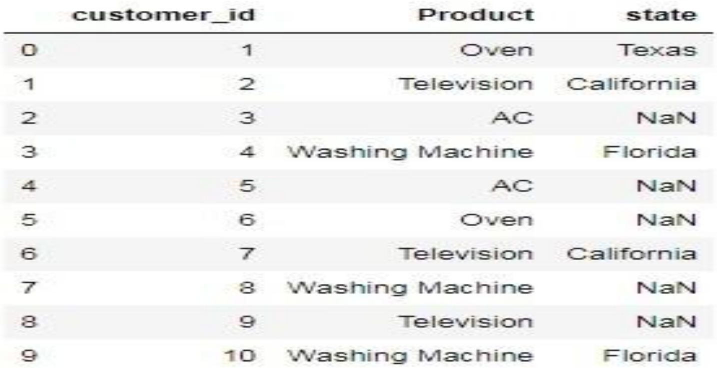
Pd.merge(df1,df2,on=‘column’, how=‘outer’)

Return all rows from both table , join records from left which have matching keys in the right table.

Code:

pd.merge(df1,df2,on='customer_id',how="outer“)

c)Left outer join:



	customer_id	Product	state
0	1	Oven	Texas
1	2	Television	California
2	3	AC	NaN
3	4	Washing Machine	Florida
4	5	AC	NaN
5	6	Oven	NaN
6	7	Television	California
7	8	Washing Machine	NaN
8	9	Television	NaN
9	10	Washing Machine	Florida

Left outer join includes all the rows of your data frame df1 and only those from df2 that match .

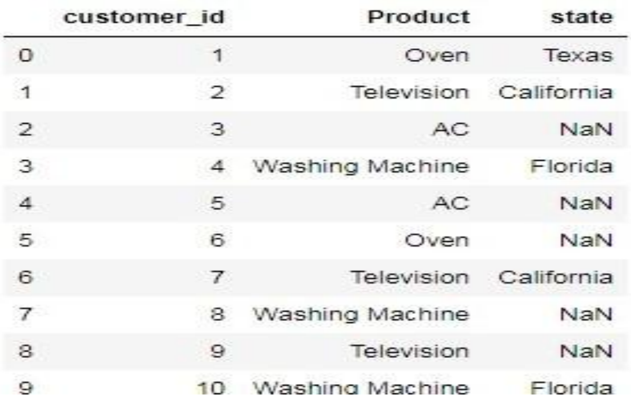
SYNTAX:

Pd.merge(df1,df2,on=“column”, how=“left”)

Return all rows from the left table ,and any rows with matching keys from the right table

Code:

pd.merge(df1,df2,on='customer_id',how="left")



	customer_id	Product	state
0	1	Oven	Texas
1	2	Television	California
2	3	AC	NaN
3	4	Washing Machine	Florida
4	5	AC	NaN
5	6	Oven	NaN
6	7	Television	California
7	8	Washing Machine	NaN
8	9	Television	NaN
9	10	Washing Machine	Florida

d)Right outer join:

Return all rows from df2 table and any rows with matching keys from the df1 table.

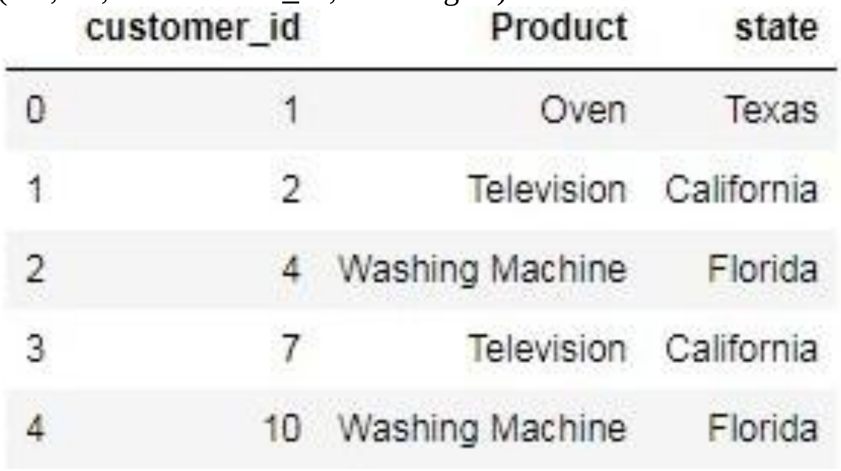
SYNTAX:

pd.merge(df1,df2,on="column", how="right")

Return all the rows from the right table and any rows with matching keys from the left table

Code:

```
pd.merge(df1,df2,on='customer_id',how="right")
```



	customer_id	Product	state
0	1	Oven	Texas
1	2	Television	California
2	4	Washing Machine	Florida
3	7	Television	California
4	10	Washing Machine	Florida

Result:

Thus the program to implement pandas merging operation was executed successfully

EX.NO: **DATE:**

EXPLORATORY DATA ANALYSIS FOR LOAN PREDICTION DATASET

Do the EDA (Exploratory Data Analysis) for loan prediction dataset.

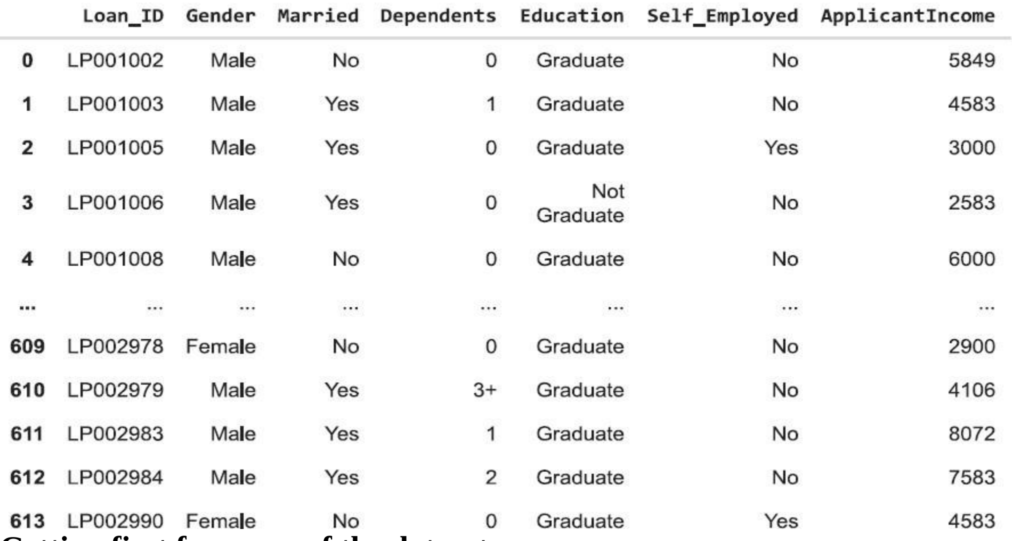
AIM:

To do the Exploratory Data Analysis for loan prediction dataset.

Procedure and Code:

Reading dataset:

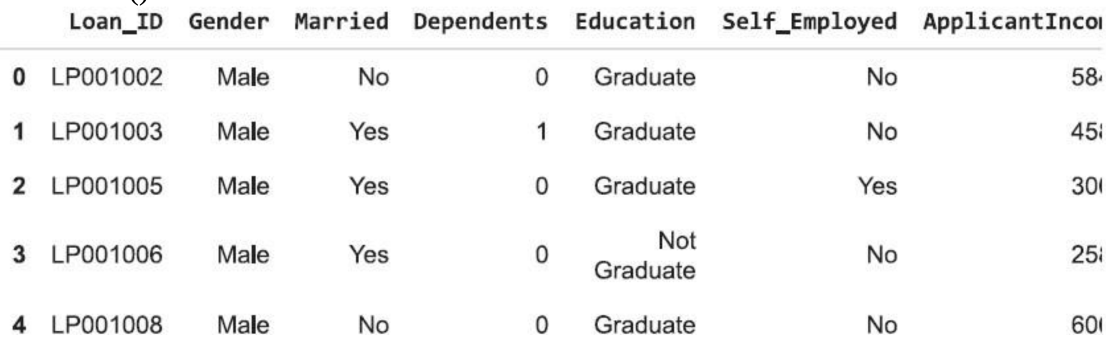
```
 import pandas as pd
 import numpy as np
data=pd.read_csv(" D:\ loan_data.csv")
data
```



	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome
0	LP001002	Male	No	0	Graduate	No	5849
1	LP001003	Male	Yes	1	Graduate	No	4583
2	LP001005	Male	Yes	0	Graduate	Yes	3000
3	LP001006	Male	Yes	0	Not Graduate	No	2583
4	LP001008	Male	No	0	Graduate	No	6000
...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900
610	LP002979	Male	Yes	3+	Graduate	No	4106
611	LP002983	Male	Yes	1	Graduate	No	8072
612	LP002984	Male	Yes	2	Graduate	No	7583
613	LP002990	Female	No	0	Graduate	Yes	4583

Getting first few rows of the dataset:

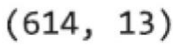
Data.head()



	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantInco
0	LP001002	Male	No	0	Graduate	No	58
1	LP001003	Male	Yes	1	Graduate	No	45
2	LP001005	Male	Yes	0	Graduate	Yes	30
3	LP001006	Male	Yes	0	Not Graduate	No	25
4	LP001008	Male	No	0	Graduate	No	60

Getting shape of the data:

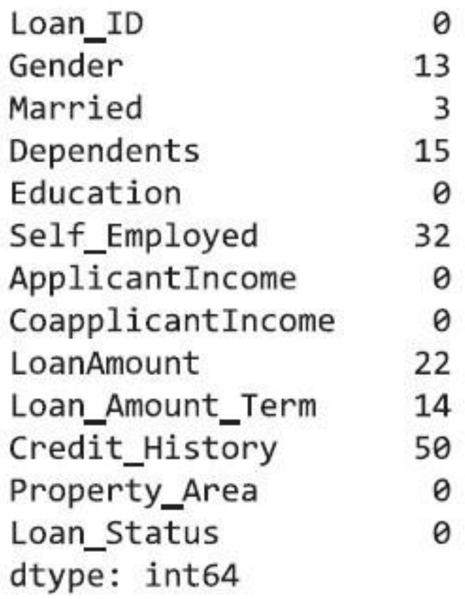
```
data.shape
```



```
(614, 13)
```

Checking missing values in the data:

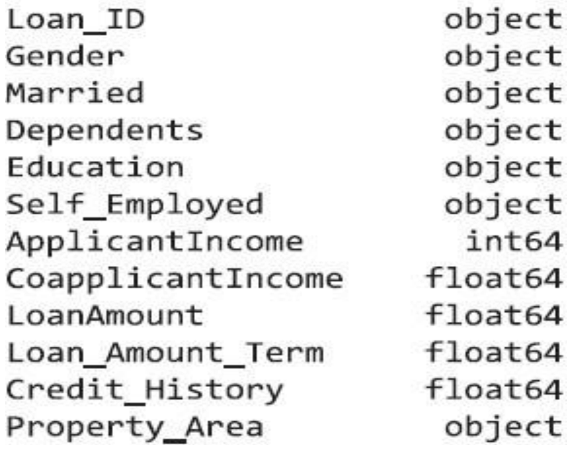
data.isnull().sum()



```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

Checking data types:

data.dtypes



```
Loan_ID                 object
Gender                  object
Married                 object
Dependents              object
Education               object
Self_Employed           object
ApplicantIncome          int64
CoapplicantIncome      float64
LoanAmount             float64
Loan_Amount_Term       float64
Credit_History         float64
Property_Area           object
```

Filling missing values with categorical variable mode:

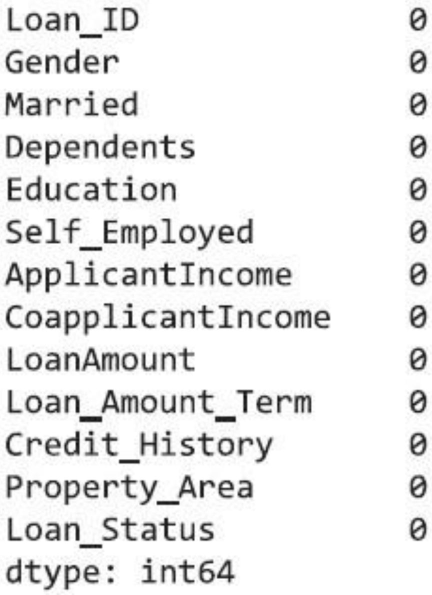
```
data[“Gender”].fillna(data[“Gender”].mode()[0],inplace=True)
data[“Married”].fillna(data[“Married”].mode()[0],inplace=True)
data[“Dependents”].fillna(data[“Dependents”].mode()[0],inplace=True)
data[“Self_Employed”].fillna(data[“Self_Employed”].mode()[0],inplace=True
)
data[“Loan_Amount_Term”].fillna(data[“Loan_Amount_Ter
m”].mode()[0],inplace=True)
data[“Credit_History”].fillna(data[“Credit_History”].mode()[
0],inplace=True)
```

Filling missing values with continuous variable with mean:

```
data[“LoanAmount”].fillna(data[“LoanAmount].mean(),inplace=True)
```

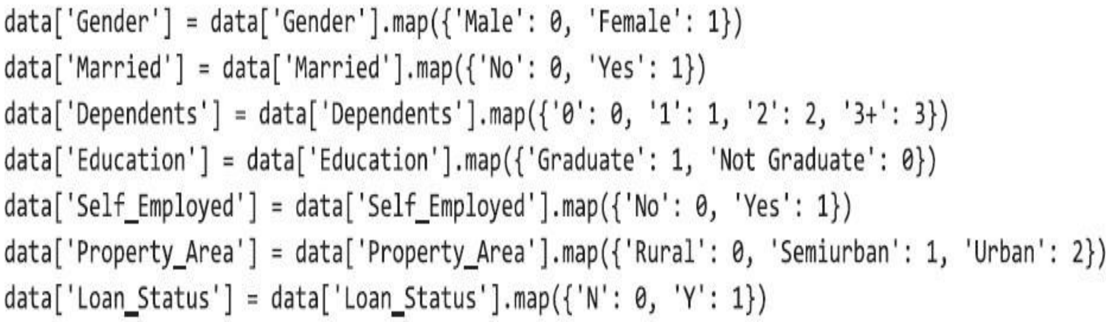
Checking missing values:

data.isnull().sum()



```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

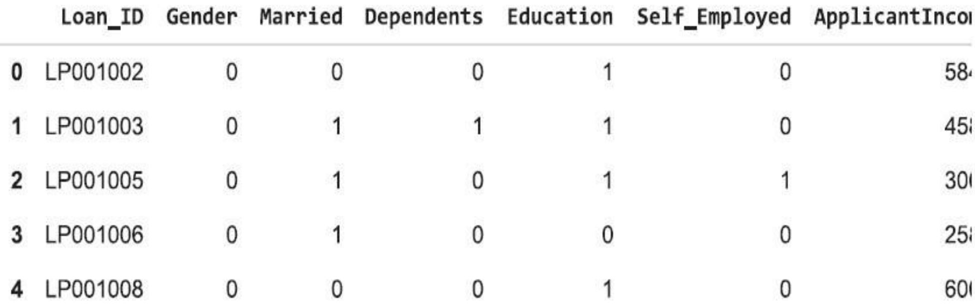
Converting Categorical into numerical:



```
data['Gender'] = data['Gender'].map({'Male': 0, 'Female': 1})
data['Married'] = data['Married'].map({'No': 0, 'Yes': 1})
data['Dependents'] = data['Dependents'].map({'0': 0, '1': 1, '2': 2, '3+': 3})
data['Education'] = data['Education'].map({'Graduate': 1, 'Not Graduate': 0})
data['Self_Employed'] = data['Self_Employed'].map({'No': 0, 'Yes': 1})
data['Property_Area'] = data['Property_Area'].map({'Rural': 0, 'Semiurban': 1, 'Urban': 2})
data['Loan_Status'] = data['Loan_Status'].map({'N': 0, 'Y': 1})
```

Checking data values:

data.head()



	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantInco
0	LP001002	0	0	0	1	0	58
1	LP001003	0	1	1	1	0	45
2	LP001005	0	1	0	1	1	30
3	LP001006	0	1	0	0	0	25
4	LP001008	0	0	0	1	0	60

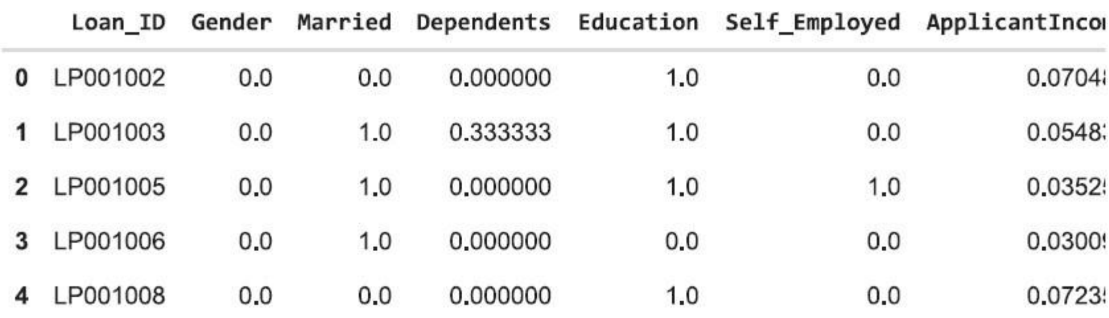
Data Normalisation:

Using for loop we can convert the all the values in the range between o to 1

```
for i in data.columns[1::]:

 data[i]=(data[i]-
data[i].min())/(data[i].max()-
data[i].min())
```

Checking values:

data.head()



	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantInco
0	LP001002	0.0	0.0	0.000000	1.0	0.0	0.0704
1	LP001003	0.0	1.0	0.333333	1.0	0.0	0.0548
2	LP001005	0.0	1.0	0.000000	1.0	1.0	0.0352
3	LP001006	0.0	1.0	0.000000	0.0	0.0	0.0300
4	LP001008	0.0	0.0	0.000000	1.0	0.0	0.0723

Saving the pre-processed data:

Data.to_csv(“new_data.csv”,index=False)

Result:

Thus the program to implement pandas EDA was executed successfully

EXNO: **CREATING A DATA FRAME FROM DICTIONARY AND**
DATE: **ACCESSING THE DATA USING PANDAS PACKAGE**

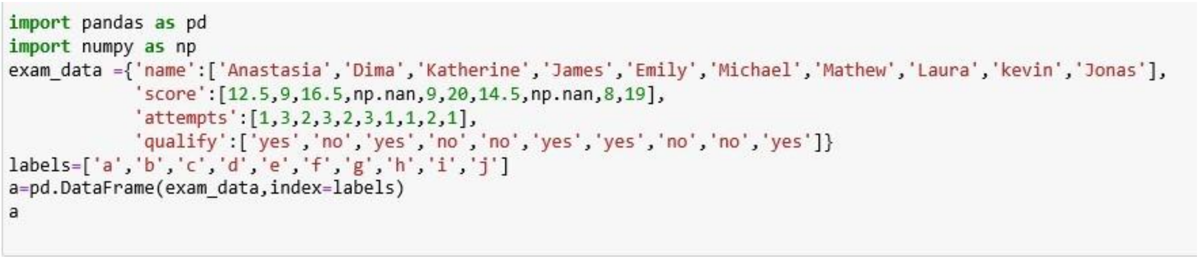
Consider a dictionary and do the following operation
exam_data = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],
'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

AIM:

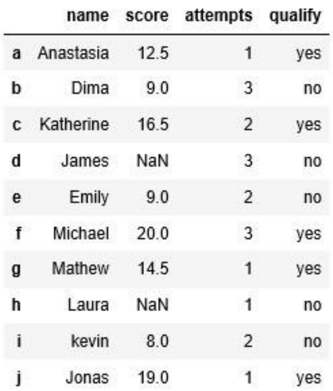
To create the data frame for a given dictionary and execute the following operations.

Procedure and Code:

1) **Write a Pandas program to create and display a Data Frame from a specified dictionary data which has the index labels.**

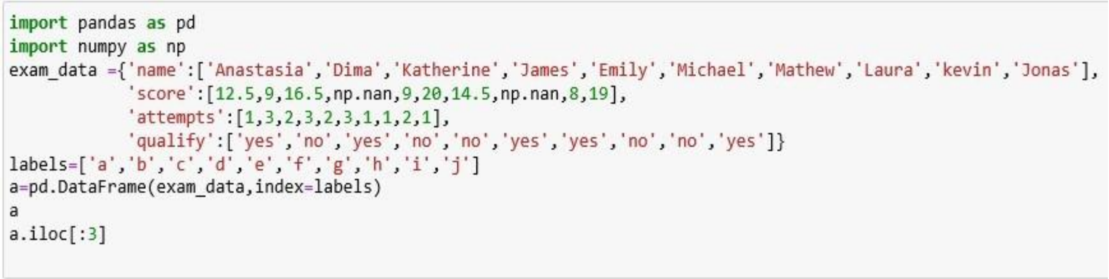


```
import pandas as pd
import numpy as np
exam_data ={'name':['Anastasia','Dima','Katherine','James','Emily','Michael','Mathew','Laura','kevin','Jonas'],
            'score':[12.5,9,16.5,np.nan,9,20,14.5,np.nan,8,19],
            'attempts':[1,3,2,3,2,3,1,1,2,1],
            'qualify':['yes','no','yes','no','no','yes','yes','no','no','yes']}
labels=['a','b','c','d','e','f','g','h','i','j']
a=pd.DataFrame(exam_data,index=labels)
a
```

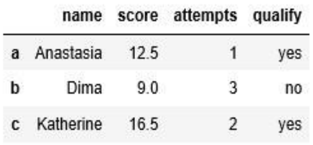


	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes
d	James	NaN	3	no
e	Emily	9.0	2	no
f	Michael	20.0	3	yes
g	Mathew	14.5	1	yes
h	Laura	NaN	1	no
i	kevin	8.0	2	no
j	Jonas	19.0	1	yes

2) Write a Pandas program to get the first 3 rows of a given DataFrame

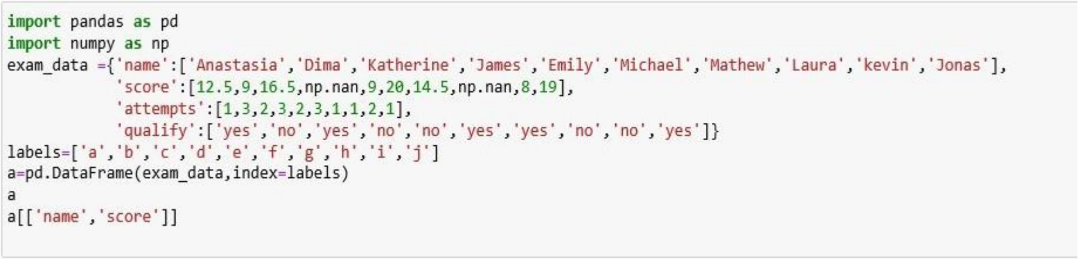


```python
import pandas as pd
import numpy as np
exam_data ={'name':['Anastasia','Dima','Katherine','James','Emily','Michael','Mathew','Laura','kevin','Jonas'],
            'score':[12.5,9,16.5,np.nan,9,20,14.5,np.nan,8,19],
            'attempts':[1,3,2,3,2,3,1,1,2,1],
            'qualify':['yes','no','yes','no','no','yes','yes','no','no','yes']}
labels=['a','b','c','d','e','f','g','h','i','j']
a=pd.DataFrame(exam_data,index=labels)
a
a.iloc[:3]
```

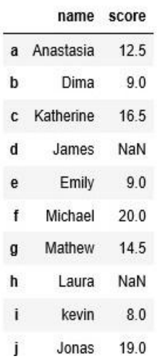


	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes

3) Write a Pandas program to select the 'name' and 'score' columns from the following DataFrame

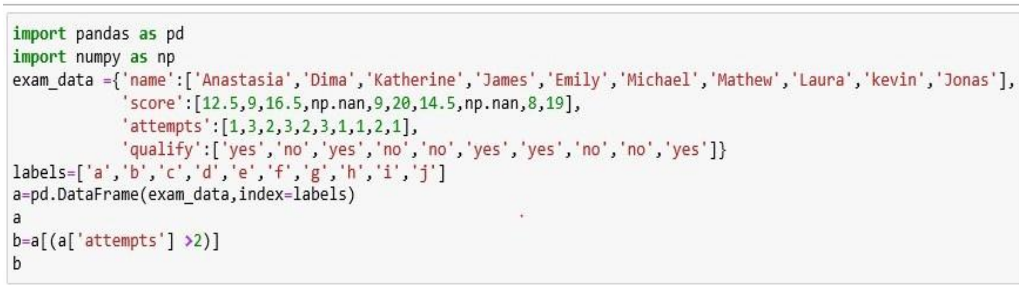


```python
import pandas as pd
import numpy as np
exam_data ={'name':['Anastasia','Dima','Katherine','James','Emily','Michael','Mathew','Laura','kevin','Jonas'],
            'score':[12.5,9,16.5,np.nan,9,20,14.5,np.nan,8,19],
            'attempts':[1,3,2,3,2,3,1,1,2,1],
            'qualify':['yes','no','yes','no','no','yes','yes','no','no','yes']}
labels=['a','b','c','d','e','f','g','h','i','j']
a=pd.DataFrame(exam_data,index=labels)
a
a[['name','score']]
```

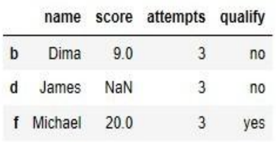


	name	score
a	Anastasia	12.5
b	Dima	9.0
c	Katherine	16.5
d	James	NaN
e	Emily	9.0
f	Michael	20.0
g	Mathew	14.5
h	Laura	NaN
i	kevin	8.0
j	Jonas	19.0

4) Write a Pandas program to select the rows where the number of attempts in the examination is greater than 2.

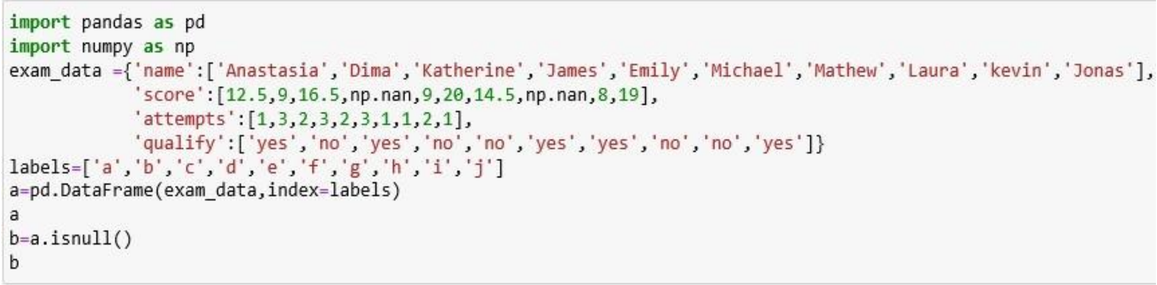


```python
import pandas as pd
import numpy as np
exam_data ={'name':['Anastasia','Dima','Katherine','James','Emily','Michael','Mathew','Laura','kevin','Jonas'],
            'score':[12.5,9,16.5,np.nan,9,20,14.5,np.nan,8,19],
            'attempts':[1,3,2,3,2,3,1,1,2,1],
            'qualify':['yes','no','yes','no','no','yes','yes','no','no','yes']}
labels=['a','b','c','d','e','f','g','h','i','j']
a=pd.DataFrame(exam_data,index=labels)
a
b=a[(a['attempts'] >2)]
b
```

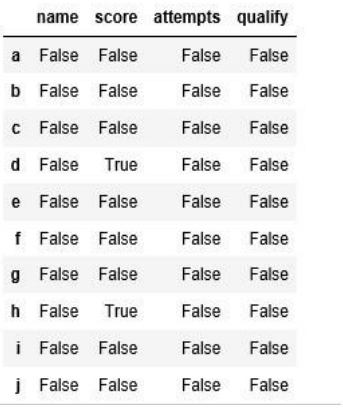


	name	score	attempts	qualify
b	Dima	9.0	3	no
d	James	NaN	3	no
f	Michael	20.0	3	yes

5)Write a Pandas program to select the rows where the score is missing, i.e. is NaN

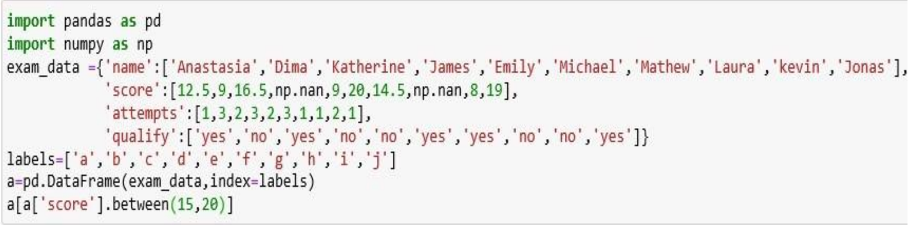


```
import pandas as pd
import numpy as np
exam_data ={'name':['Anastasia','Dima','Katherine','James','Emily','Michael','Mathew','Laura','kevin','Jonas'],
            'score':[12.5,9,16.5,np.nan,9,20,14.5,np.nan,8,19],
            'attempts':[1,3,2,3,2,3,1,1,2,1],
            'qualify':['yes','no','yes','no','no','yes','yes','no','no','yes']}
labels=['a','b','c','d','e','f','g','h','i','j']
a=pd.DataFrame(exam_data,index=labels)
a
b=a.isnull()
b
```

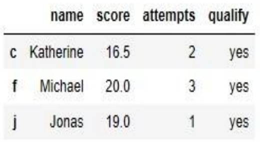


```
   name  score  attempts  qualify
a  False  False    False    False
b  False  False    False    False
c  False  False    False    False
d  False   True    False    False
e  False  False    False    False
f  False  False    False    False
g  False  False    False    False
h  False   True    False    False
i  False  False    False    False
j  False  False    False    False
```

6)Write a Pandas program to select the rows the score is between 15 and 20 (inclusive).

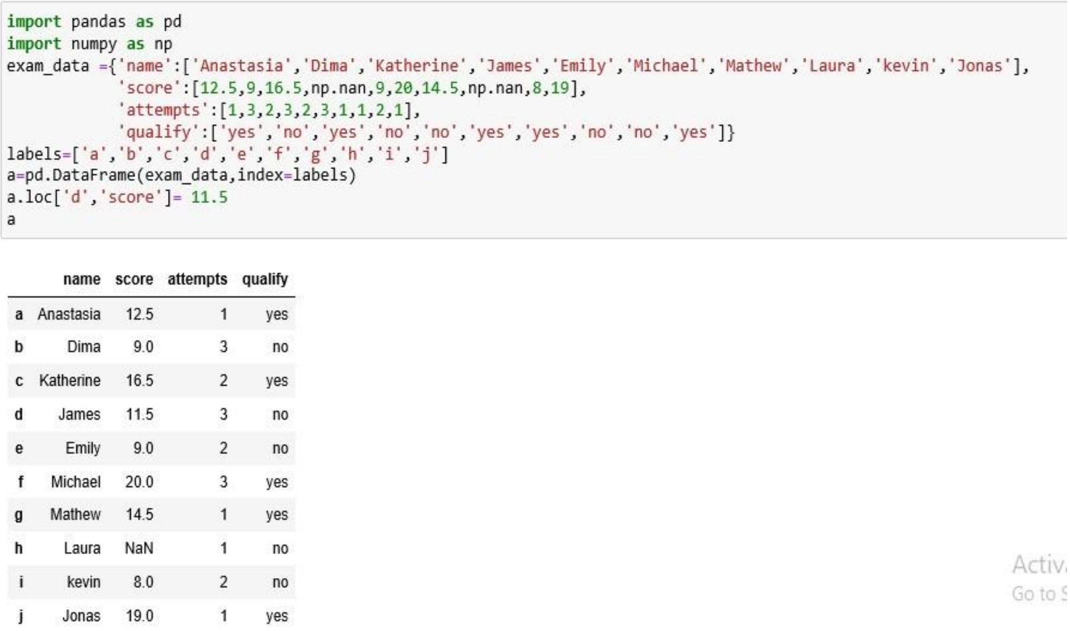


```
import pandas as pd
import numpy as np
exam_data ={'name':['Anastasia','Dima','Katherine','James','Emily','Michael','Mathew','Laura','kevin','Jonas'],
            'score':[12.5,9,16.5,np.nan,9,20,14.5,np.nan,8,19],
            'attempts':[1,3,2,3,2,3,1,1,2,1],
            'qualify':['yes','no','yes','no','no','yes','yes','no','no','yes']}
labels=['a','b','c','d','e','f','g','h','i','j']
a=pd.DataFrame(exam_data,index=labels)
a[a['score'].between(15,20)]
```



```
        name  score  attempts  qualify
c  Katherine   16.5         2      yes
f    Michael   20.0         3      yes
j      Jonas   19.0         1      yes
```

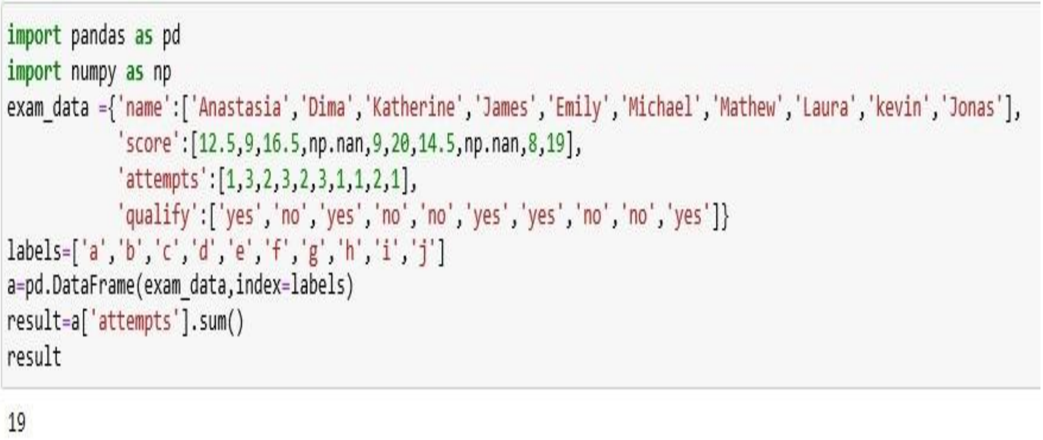
7) Write a Pandas program to change the score in row’d’ to 11.5.



```
import pandas as pd
import numpy as np
exam_data ={'name':['Anastasia','Dima','Katherine','James','Emily','Michael','Mathew','Laura','kevin','Jonas'],
            'score':[12.5,9,16.5,np.nan,9,20,14.5,np.nan,8,19],
            'attempts':[1,3,2,3,2,3,1,1,2,1],
            'qualify':['yes','no','yes','no','no','yes','yes','no','no','yes']}
labels=['a','b','c','d','e','f','g','h','i','j']
a=pd.DataFrame(exam_data,index=labels)
a.loc['d','score']= 11.5
a
```

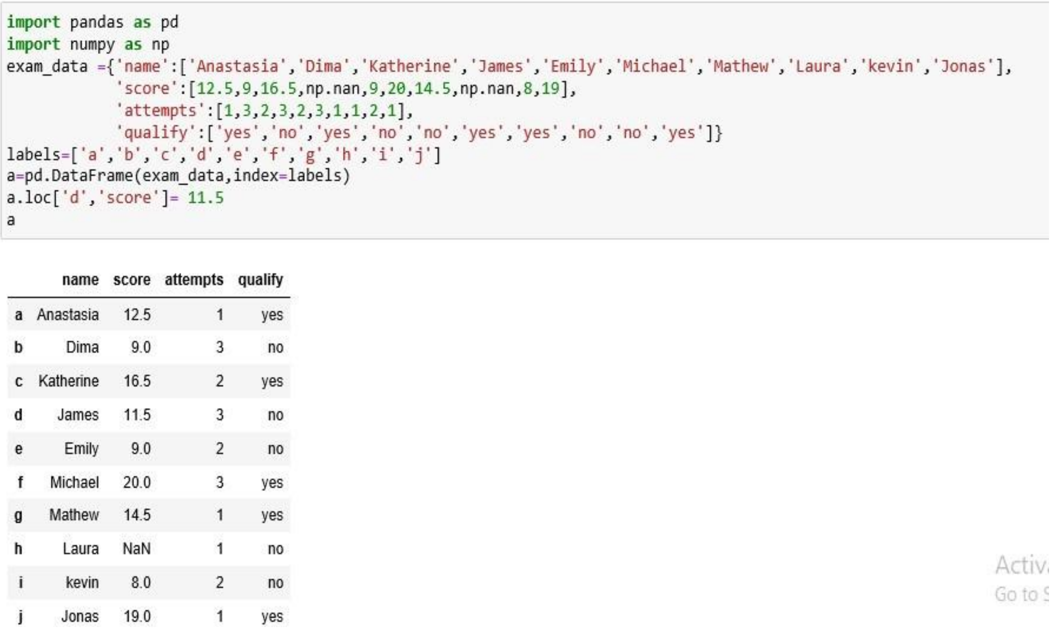
	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes
d	James	11.5	3	no
e	Emily	9.0	2	no
f	Michael	20.0	3	yes
g	Mathew	14.5	1	yes
h	Laura	NaN	1	no
i	kevin	8.0	2	no
j	Jonas	19.0	1	yes

8) Write a Pandas program to calculate the sum of the examination attempts by the students.



```
import pandas as pd
import numpy as np
exam_data ={'name':['Anastasia','Dima','Katherine','James','Emily','Michael','Mathew','Laura','kevin','Jonas'],
            'score':[12.5,9,16.5,np.nan,9,20,14.5,np.nan,8,19],
            'attempts':[1,3,2,3,2,3,1,1,2,1],
            'qualify':['yes','no','yes','no','no','yes','yes','no','no','yes']}
labels=['a','b','c','d','e','f','g','h','i','j']
a=pd.DataFrame(exam_data,index=labels)
result=a['attempts'].sum()
result
```

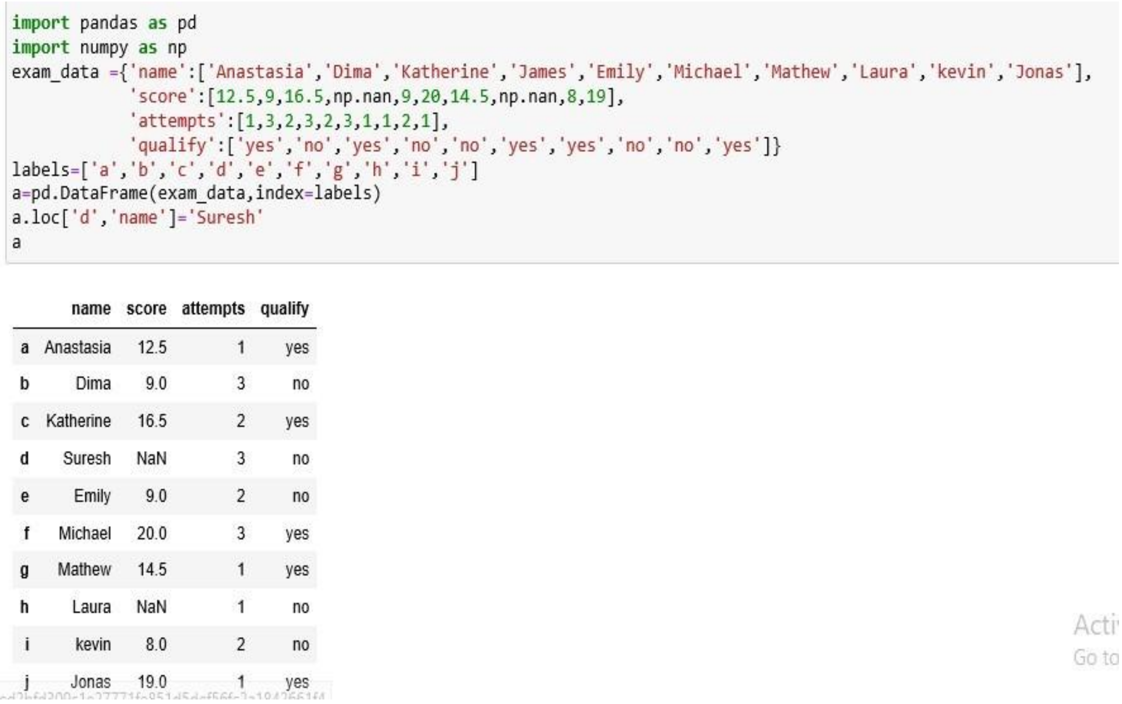
```
19
```



```
import pandas as pd
import numpy as np
exam_data ={'name':['Anastasia','Dima','Katherine','James','Emily','Michael','Mathew','Laura','kevin','Jonas'],
            'score':[12.5,9,16.5,np.nan,9,20,14.5,np.nan,8,19],
            'attempts':[1,3,2,3,2,3,1,1,2,1],
            'qualify':['yes','no','yes','no','no','yes','yes','no','no','yes']}
labels=['a','b','c','d','e','f','g','h','i','j']
a=pd.DataFrame(exam_data,index=labels)
a.loc['d','score']= 11.5
a
```

	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes
d	James	11.5	3	no
e	Emily	9.0	2	no
f	Michael	20.0	3	yes
g	Mathew	14.5	1	yes
h	Laura	NaN	1	no
i	kevin	8.0	2	no
j	Jonas	19.0	1	yes

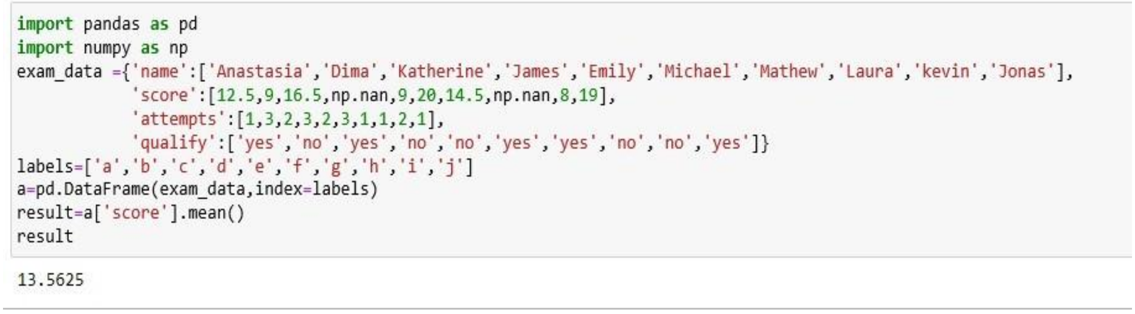
9) Write a Pandas program to change the name 'James' to 'Suresh' in name column of the data frame.



```
import pandas as pd
import numpy as np
exam_data ={'name':['Anastasia','Dima','Katherine','James','Emily','Michael','Mathew','Laura','kevin','Jonas'],
            'score':[12.5,9,16.5,np.nan,9,20,14.5,np.nan,8,19],
            'attempts':[1,3,2,3,2,3,1,1,2,1],
            'qualify':['yes','no','yes','no','no','yes','yes','no','no','yes']}
labels=['a','b','c','d','e','f','g','h','i','j']
a=pd.DataFrame(exam_data,index=labels)
a.loc['d','name']='Suresh'
a
```

	name	score	attempts	qualify
a	Anastasia	12.5	1	yes
b	Dima	9.0	3	no
c	Katherine	16.5	2	yes
d	Suresh	NaN	3	no
e	Emily	9.0	2	no
f	Michael	20.0	3	yes
g	Mathew	14.5	1	yes
h	Laura	NaN	1	no
i	kevin	8.0	2	no
j	Jonas	19.0	1	yes

10) Write a Pandas program to calculate the mean score for each different student in data frame.



```
import pandas as pd
import numpy as np
exam_data ={'name':['Anastasia','Dima','Katherine','James','Emily','Michael','Mathew','Laura','kevin','Jonas'],
            'score':[12.5,9,16.5,np.nan,9,20,14.5,np.nan,8,19],
            'attempts':[1,3,2,3,2,3,1,1,2,1],
            'qualify':['yes','no','yes','no','no','yes','yes','no','no','yes']}
labels=['a','b','c','d','e','f','g','h','i','j']
a=pd.DataFrame(exam_data,index=labels)
result=a['score'].mean()
result
```

```
13.5625
```

Result:

Thus the program to implement pandas data frame for a given dictionary and execute operations was executed successfully

Ex.No 3 a) Basic plots using Matplotlib

Aim:

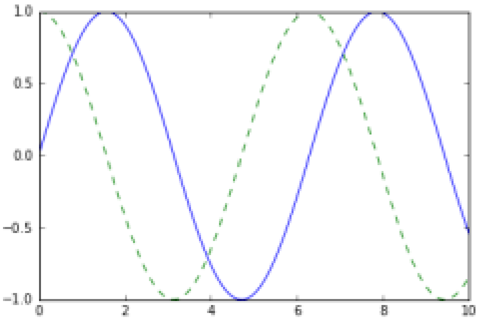
To write a Python Program to perform Basic Plots with matplotlib

Program:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.show()
```

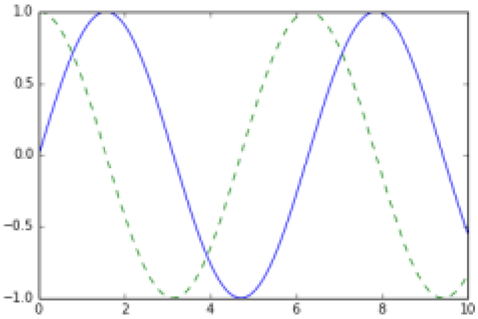
```
In[4]: import numpy as np
       x = np.linspace(0, 10, 100)
       fig = plt.figure()
       plt.plot(x, np.sin(x), '-')
       plt.plot(x, np.cos(x), '--');
```

Output:



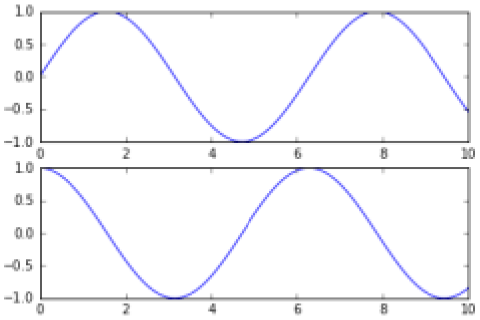
```
In[5]: fig.savefig('my_figure.png')
```

```
In[7]: from IPython.display import Image
       Image('my_figure.png')
```



```
In[9]: plt.figure()  # create a plot figure
       # create the first of two panels and set current axis
       plt.subplot(2, 1, 1) # (rows, columns, panel number)
       plt.plot(x, np.sin(x))
       # create the second panel and set current axis
       plt.subplot(2, 1, 2)
       plt.plot(x, np.cos(x));
```

Output:



Result:

Thus the program to implement Basic Plots with matplotlib was executed successfully

Ex.No 3 b) **Line plots using Matplotlib**

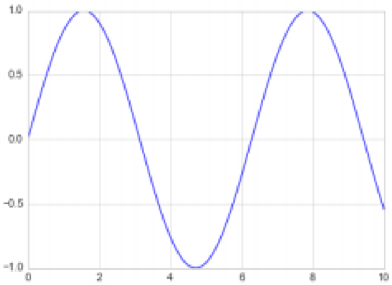
Aim:

To write a Python Program to perform Basic Plots with matplotlib

Program:

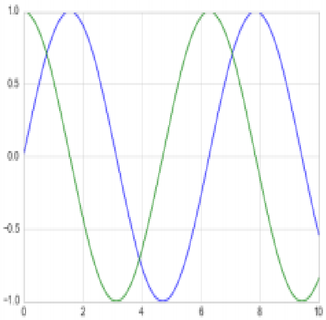
```
In[1]: %matplotlib inline
       import matplotlib.pyplot as plt
       plt.style.use('seaborn-whitegrid')
       import numpy as np
       fig = plt.figure()
       ax = plt.axes()
       x = np.linspace(0, 10, 1000)
       ax.plot(x, np.sin(x));
```

Output:



```
In[5]: plt.plot(x, np.sin(x))
       plt.plot(x, np.cos(x));
```

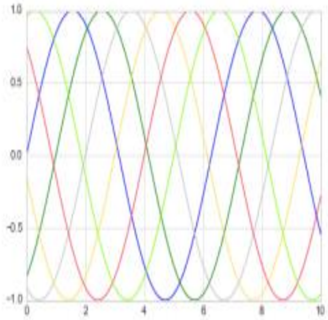
Output:



Adjusting the Plot: Line Colors and Styles

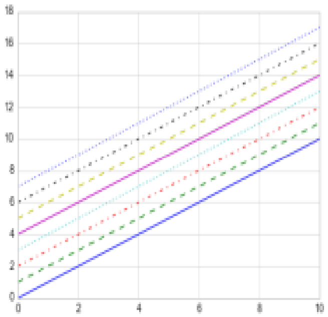
```
In[6]:
plt.plot(x, np.sin(x - 0), color='blue')       # specify color by name
plt.plot(x, np.sin(x - 1), color='g')          # short color code (rgbcmyk)
plt.plot(x, np.sin(x - 2), color='0.75')       # Grayscale between 0 and 1
plt.plot(x, np.sin(x - 3), color='#FFDD44')    # Hex code (RRGGBB from 00 to FF)
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3)) # RGB tuple, values 0 and 1
plt.plot(x, np.sin(x - 5), color='chartreuse'); # all HTML color names supported
```

Output:



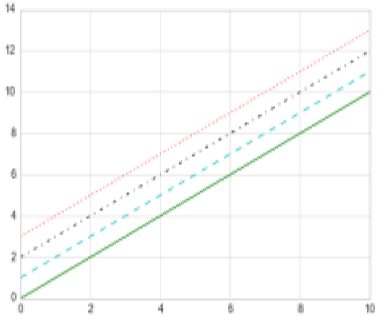
```
In[7]: plt.plot(x, x + 0, linestyle='solid')
       plt.plot(x, x + 1, linestyle='dashed')
       plt.plot(x, x + 2, linestyle='dashdot')
       plt.plot(x, x + 3, linestyle='dotted');
       # For short, you can use the following codes:
       plt.plot(x, x + 4, linestyle='-')  # solid
       plt.plot(x, x + 5, linestyle='--') # dashed
       plt.plot(x, x + 6, linestyle='-.') # dashdot
       plt.plot(x, x + 7, linestyle=':');  # dotted
```

Output:



```
In[8]: plt.plot(x, x + 0, '-g')  # solid green
       plt.plot(x, x + 1, '--c') # dashed cyan
       plt.plot(x, x + 2, '-.k') # dashdot black
       plt.plot(x, x + 3, ':r');  # dotted red
```

Output:

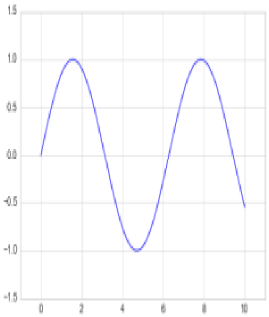


Adjusting the Plot: Axes Limits

```
In[9]: plt.plot(x, np.sin(x))
```

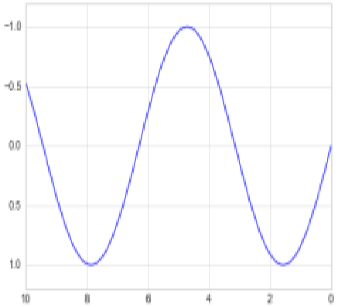
```
        plt.xlim(-1, 11)
        plt.ylim(-1.5, 1.5);
```

Output:

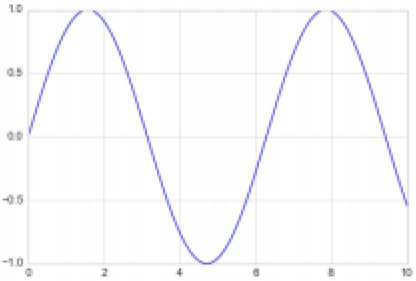


```
In[11]: plt.plot(x, np.sin(x))
        plt.axis([-1, 11, -1.5, 1.5]);
```

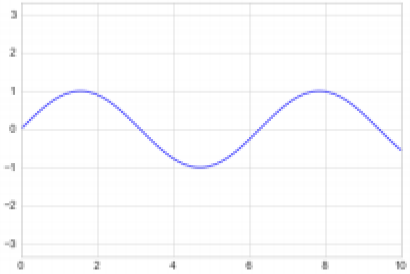
Output:



```
In[12]: plt.plot(x, np.sin(x))
        plt.axis('tight');
```



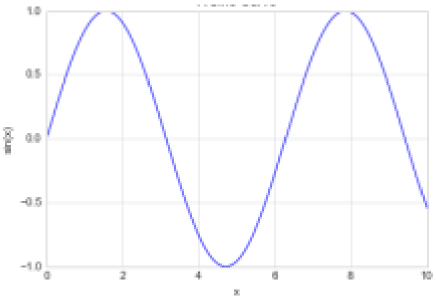
```
In[13]: plt.plot(x, np.sin(x))
        plt.axis('equal');
```



Labeling Plots

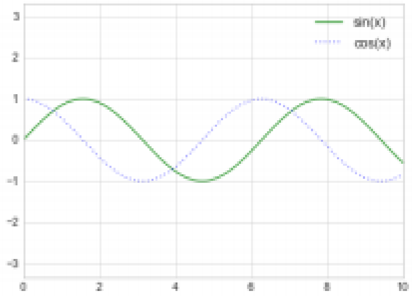
```
In[14]: plt.plot(x, np.sin(x))
        plt.title("A Sine Curve")
        plt.xlabel("x")
        plt.ylabel("sin(x)");
```

Output:



```
In[15]: plt.plot(x, np.sin(x), '-g', label='sin(x)')
        plt.plot(x, np.cos(x), ':b', label='cos(x)')
        plt.axis('equal')
        plt.legend();
```

Output:



Result:

Thus the program to implement Line Plot was executed successfully

Ex.No 3 c) **Scatter plots using Matplotlib**

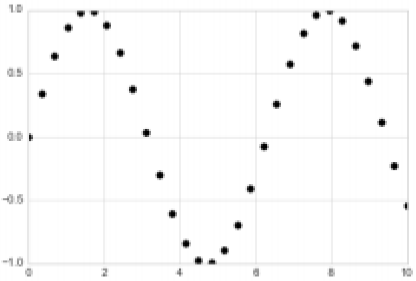
Aim:

To write a Python Program to perform Basic Plots with matplotlib

Program:

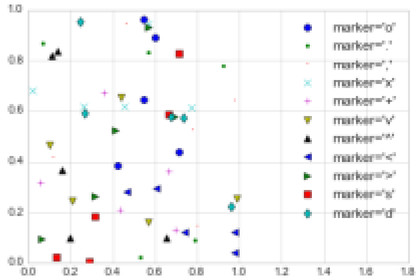
```
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')
import numpy as np
x = np.linspace(0, 10, 30)
y = np.sin(x)
plt.plot(x, y, 'o', color='black');
```

Output



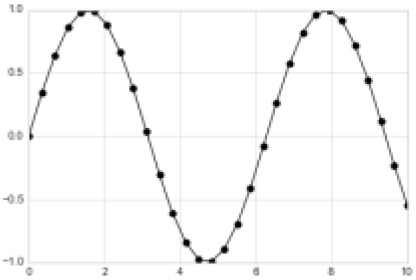
```
rng = np.random.RandomState(0)
for marker in ['o', '.', ',', 'x', '+', 'v', '^', '<', '>', 's', 'd']:
    plt.plot(rng.rand(5), rng.rand(5), marker,
             label="marker='{0}'".format(marker))
plt.legend(numpoints=1)
plt.xlim(0, 1.8);
```

Output



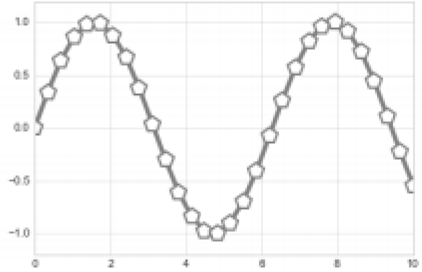
```
plt.plot(x, y, '-ok');   # line (-), circle marker (o), black (k)
```

Output



```
plt.plot(x, y, '-p', color='gray',
     markersize=15, linewidth=4,
     markerfacecolor='white',
             markeredgecolor='gray',
             markeredgewidth=2)
    plt.ylim(-1.2, 1.2);
```

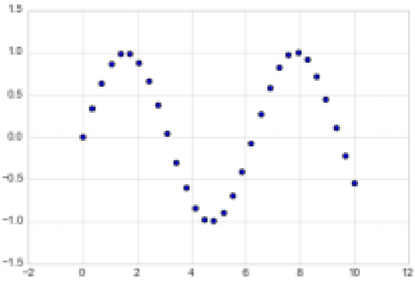
Output



Scatter Plots with plt.scatter

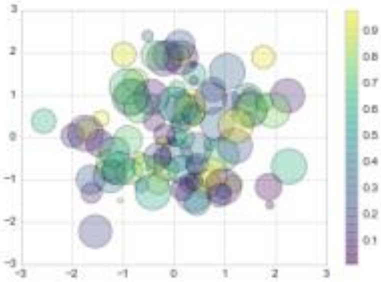
```
In[6]: plt.scatter(x, y, marker='o');
```

Output



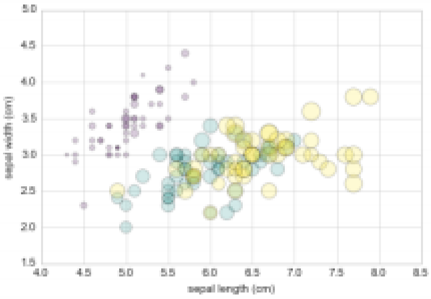
```
In[7]: rng = np.random.RandomState(0)
    x = rng.randn(100)
    y = rng.randn(100)
    colors = rng.rand(100)
    sizes = 1000 * rng.rand(100)
    plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,
             cmap='viridis')
    plt.colorbar();  # show color scale
```

Output



```
In[8]: from sklearn.datasets import load_iris
    iris = load_iris()
    features = iris.data.T
    plt.scatter(features[0], features[1], alpha=0.2,
             s=100*features[3], c=iris.target, cmap='viridis')
    plt.xlabel(iris.feature_names[0])
    plt.ylabel(iris.feature_names[1]);
```

Output



Result:

Thus the

Ex.No 3 d) Basic plots using Matplotlib

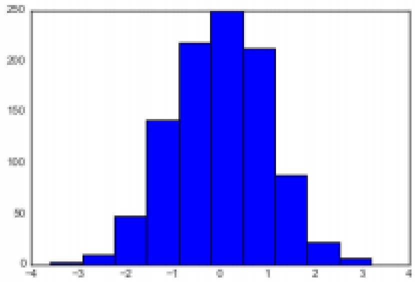
Aim:

To write a Python Program to perform Histograms, binnings, and density with matplotlib

Program:

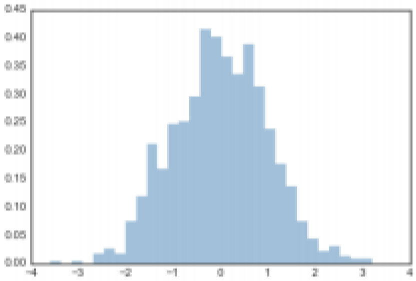
```
%matplotlib inline
 import numpy as np
 import matplotlib.pyplot as plt
 plt.style.use('seaborn-white')
 data = np.random.randn(1000)
plt.hist(data);
```

Output:



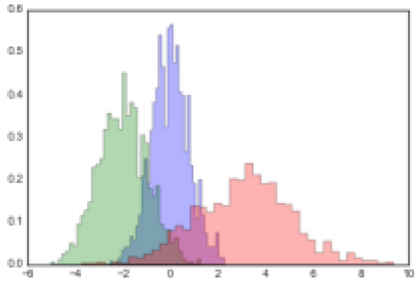
```
plt.hist(data, bins=30, normed=True, alpha=0.5,
    histtype='stepfilled', color='steelblue',
    edgecolor='none');
```

Output:



```
x1 = np.random.normal(0, 0.8, 1000)
     x2 = np.random.normal(-2, 1, 1000)
     x3 = np.random.normal(3, 2, 1000)
     kwargs = dict(histtype='stepfilled', alpha=0.3, normed=True, bins=40)
     plt.hist(x1, **kwargs)
     plt.hist(x2, **kwargs)
     plt.hist(x3, **kwargs);
```

Output:



```
        counts, bin_edges = np.histogram(data, bins=5)
     print(counts)
```

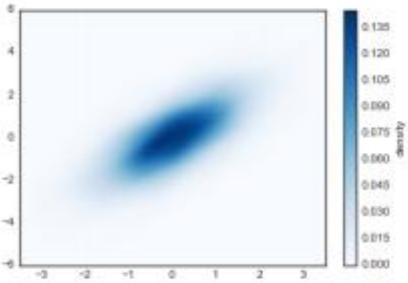
Output:

```
[ 12 190 468 301  29]
```

Kernel density estimation

```
In[10]: from scipy.stats import gaussian_kde
      # fit an array of size [Ndim, Nsamples]
      data = np.vstack([x, y])
      kde = gaussian_kde(data)
      # evaluate on a regular grid
      xgrid = np.linspace(-3.5, 3.5, 40)
      ygrid = np.linspace(-6, 6, 40)
      Xgrid, Ygrid = np.meshgrid(xgrid, ygrid)
      Z = kde.evaluate(np.vstack([Xgrid.ravel(), Ygrid.ravel()]))
      # Plot the result as an image
      plt.imshow(Z.reshape(Xgrid.shape),
              origin='lower', aspect='auto',
              extent=[-3.5, 3.5, -6, 6],
              cmap='Blues')
      cb = plt.colorbar()
      cb.set_label("density")
```

Output:



Result:

Thus the program to implement histograms,binning,density was executed successfully

Ex.No 4 Frequency Distributions

Aim:

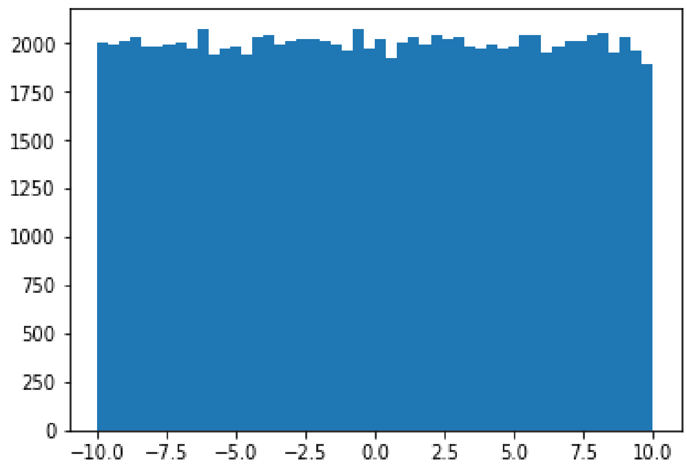
To write a Python Program to perform Frequency Distributions

Algorithm

Uniform Distribution

```
 %matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

values = np.random.uniform(-10.0, 10.0, 100000)
plt.hist(values, 50)
plt.show()
```

Output:

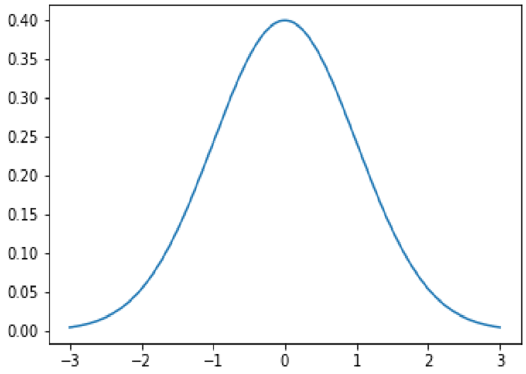


Normal / Gaussian

Visualize the probability density function:

```
from scipy.stats import norm
import matplotlib.pyplot as plt

x = np.arange(-3, 3, 0.001)
plt.plot(x, norm.pdf(x))
```

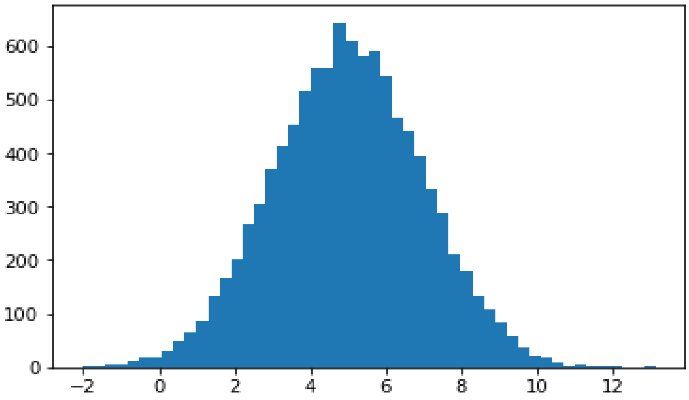
Output:



Generate some random numbers with a normal distribution. "mu" is the desired mean, "sigma" is the standard deviation:

```
import numpy as np
import matplotlib.pyplot as plt

mu = 5.0
sigma = 2.0
values = np.random.normal(mu, sigma, 10000)
plt.hist(values, 50)
plt.show()
```

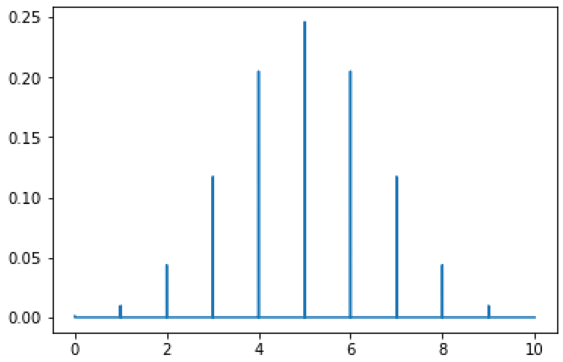
Output:



Binomial Probability Mass Function

```
from scipy.stats import binom
import matplotlib.pyplot as plt

n, p = 10, 0.5
x = np.arange(0, 10, 0.001)
plt.plot(x, binom.pmf(x, n, p))
```

Output:



Result:

Thus the program to implement frequency was executed successfully

Ex.No 5 Averages

Aim:

To write a Python Program to perform averages

Program:

```
import numpy as np

# 1D array
arr = [20, 2, 7, 1, 34]

print("arr : ", arr)
print("mean of arr : ", np.mean(arr))
```

Output

```
arr :  [20, 2, 7, 1, 34]

mean of arr :  12.8
```

Program:

```
import numpy as np



# 2D array
arr = [[14, 17, 12, 33, 44],
       [15, 6, 27, 8, 19],
       [23, 2, 54, 1, 4, ]]

# mean of the flattened array
print("\nmean of arr, axis = None : ", np.mean(arr))

# mean along the axis = 0
print("\nmean of arr, axis = 0 : ", np.mean(arr, axis = 0))

# mean along the axis = 1
print("\nmean of arr, axis = 1 : ", np.mean(arr, axis = 1))

out_arr = np.arange(3)
print("\nout_arr : ", out_arr)
print("mean of arr, axis = 1 : ",
      np.mean(arr, axis = 1, out = out_arr))
```

Output

```
mean of arr, axis = None :  18.6


mean of arr, axis = 0 :  [17.33333333  8.33333333 31.          14.
22.33333333]


mean of arr, axis = 1 :  [24.  15.  16.8]
```

```
out_arr :  [0 1 2]

mean of arr, axis = 1 :  [24 15 16]
```

Program:

```
# Python Program illustrating numpy.std() method
import numpy as np

# 1D array
arr = [20, 2, 7, 1, 34]

print("arr : ", arr)
print("std of arr : ", np.std(arr))

print ("\nMore precision with float32")
print("std of arr : ", np.std(arr, dtype = np.float32))

print ("\nMore accuracy with float64")
print("std of arr : ", np.std(arr, dtype = np.float64))
```

Output

```
arr :  [20, 2, 7, 1, 34]

std of arr :  12.576167937809991


More precision with float32

std of arr :  12.576168


More accuracy with float64

std of arr :  12.576167937809991
```

Program:

```
# Python Program illustrating
# numpy.std() method
import numpy as np


# 2D array
arr = [[2, 2, 2, 2, 2],
       [15, 6, 27, 8, 2],
       [23, 2, 54, 1, 2, ],
       [11, 44, 34, 7, 2]]


# std of the flattened array
print("\nstd of arr, axis = None : ", np.std(arr))

# std along the axis = 0
print("\nstd of arr, axis = 0 : ", np.std(arr, axis = 0))

# std along the axis = 1
```

```
print("\nstd of arr, axis = 1 : ", np.std(arr, axis = 1))
```

Output

```
std of arr, axis = None :  15.3668474320532


std of arr, axis = 0 :  [ 7.56224173 17.68473918 18.59267329
3.04138127  0.        ]


std of arr, axis = 1 :  [ 0.          8.7772433  20.53874388
16.40243884]
```

Ex.No 6 Variability

Aim:

To write a Python Program to perform variability

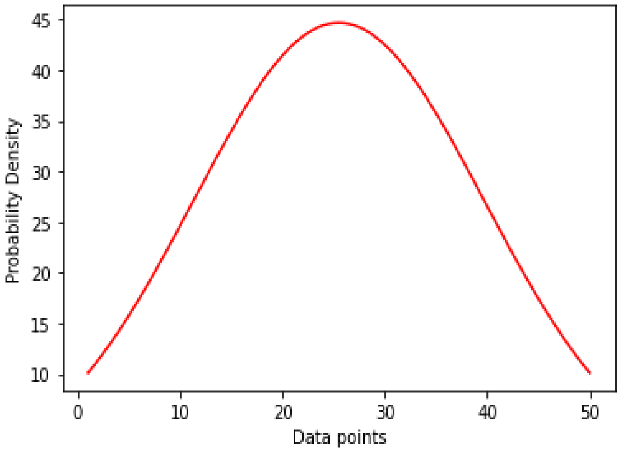
Algorithm

1. Import required libraries
2. Creating a series of data of in range of 1-50.
3. Create a Function to compute normal distribution.
4. Calculate mean and Standard deviation.
5. Apply function to the data.
6. Plot the Results
7. End

Program:

```
 import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(1,50,200)
def normal_dist(x , mean , sd):
    prob_density = (np.pi*sd) * np.exp(-0.5*((x-mean)/sd)**2)
    return prob_density
mean = np.mean(x)
sd = np.std(x)
 pdf = normal_dist(x,mean,sd)
 plt.plot(x,pdf , color = 'red')
plt.xlabel('Data points')
plt.ylabel('Probability Density')
```

Output:



Result:

Thus the program to implement variability was executed successfully

Ex.No 7 a Normal Curves

Aim:

To write a Python Program to implement Normal Curves

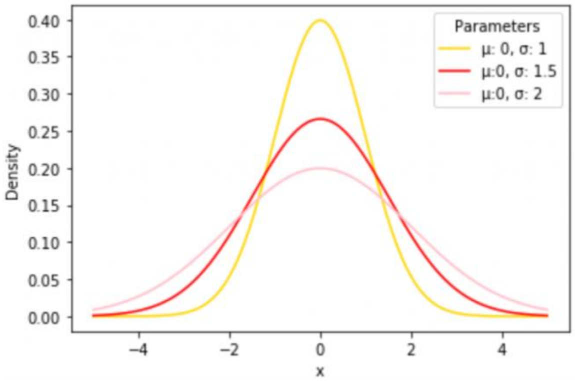
Algorithm

1. Import required libraries
2. x-axis ranges from -5 and 5 with .001 steps
3. Define multiple normal distributions
4. Add legend to plot
5. Add axes labels and a title

Program:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
x = np.arange(-5, 5, 0.001)
plt.plot(x, norm.pdf(x, 0, 1), label='μ: 0, σ: 1', color='gold')
plt.plot(x, norm.pdf(x, 0, 1.5), label='μ:0, σ: 1.5', color='red')
plt.plot(x, norm.pdf(x, 0, 2), label='μ:0, σ: 2', color='pink')
plt.legend(title='Parameters')
plt.ylabel('Density')
plt.xlabel('x')
plt.title('Normal Distributions', fontsize=14)
```

Output:



Result:

Thus the program to implement Normal Curves was executed successfully

Ex.No 7 b Normal Curves

Aim:

To write a Python Program to implement Normal Curves

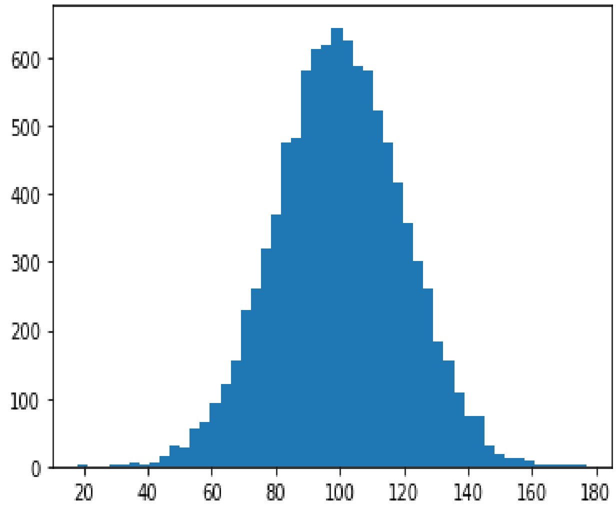
Algorithm

1. Import required libraries
2. x-axis ranges from -5 and 5 with .001 steps
3. Define multiple normal distributions
4. Add legend to plot
5. Add axes labels and a title

Program

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
incomes = np.random.normal(100.0, 20.0, 10000)
plt.hist(incomes, 50)
plt.show()
```

Output



Result:

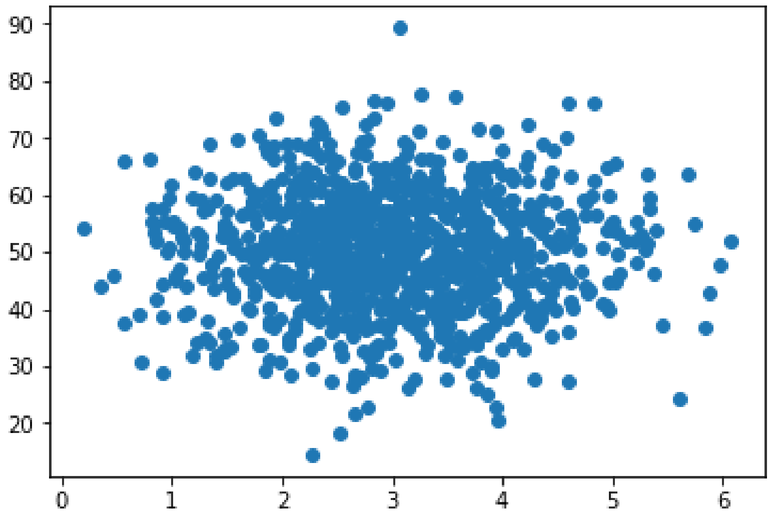
Thus the program to implement Normal Curves was executed successfully

Correlation

```
import numpy as np
from pylab import *

def de_mean(x):
    xmean = mean(x)
    return [xi - xmean for xi in x]

def covariance(x, y):
    n = len(x)
    return dot(de_mean(x), de_mean(y)) / (n-1)

pageSpeeds = np.random.normal(3.0, 1.0, 1000)
purchaseAmount = np.random.normal(50.0, 10.0, 1000)

scatter(pageSpeeds, purchaseAmount)

covariance (pageSpeeds, purchaseAmount)
```

Output

```
-0.019528192170968867
```

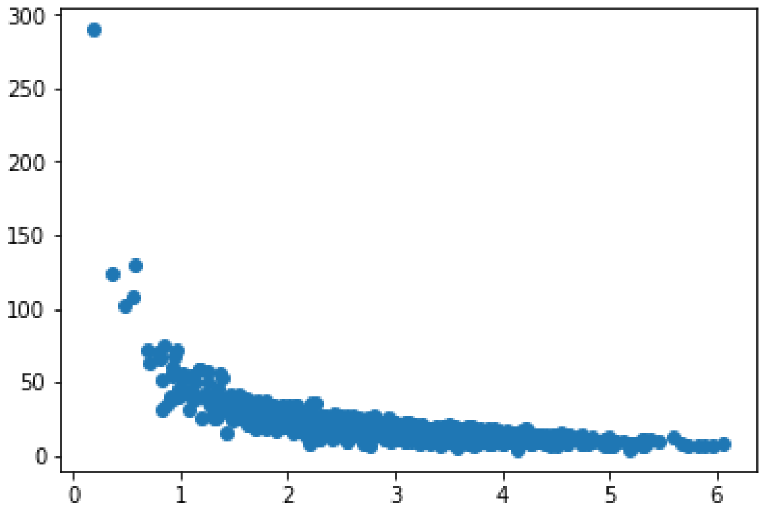


```
purchaseAmount = np.random.normal(50.0, 10.0, 1000) / pageSpeeds
```

```
scatter(pageSpeeds, purchaseAmount)
```

```
covariance (pageSpeeds, purchaseAmount)
```

```
-8.8565771898786672
```



```
def correlation(x, y):
    stddevx = x.std()
    stddevy = y.std()
    return covariance(x,y) / stddevx / stddevy  #In real life you'd check for divide by zero here
```

```
correlation(pageSpeeds, purchaseAmount)
```

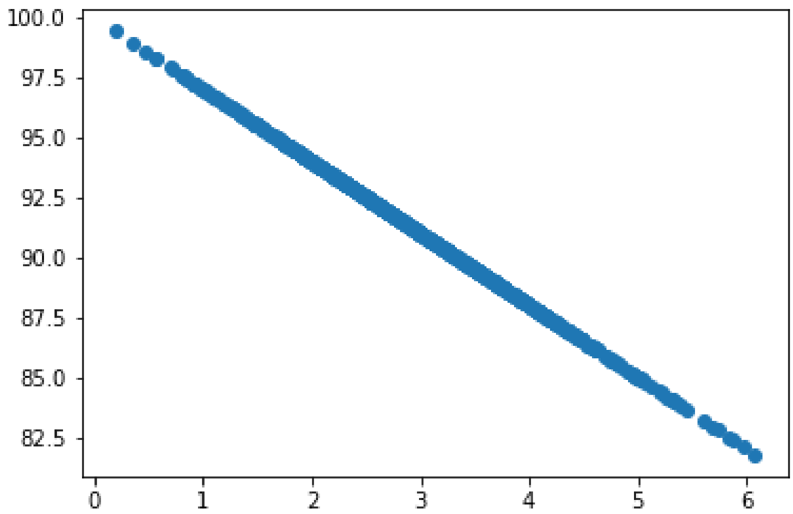
```
-0.62897824783314804
```

```
np.corrcoef(pageSpeeds, purchaseAmount)
```

```
array([[ 1.        , -0.62834927],
       [-0.62834927,  1.        ]])
```

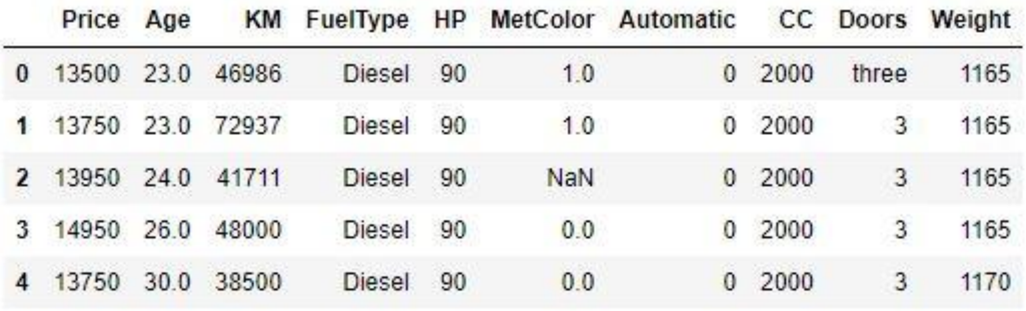
```
purchaseAmount = 100 - pageSpeeds * 3

scatter(pageSpeeds, purchaseAmount)

correlation (pageSpeeds, purchaseAmount)
```

```
-1.0010010010010009
```



Data analysis using statmodels and seaborn

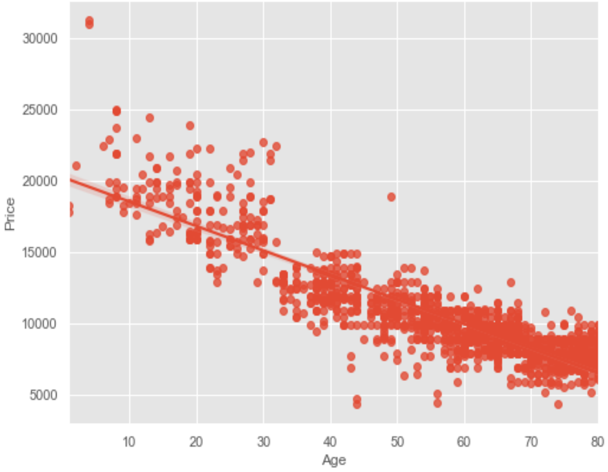
The dataset called Toyota Corolla, which is a cars dataset. Here’s the head of the dataset-



	Price	Age	KM	FuelType	HP	MetColor	Automatic	CC	Doors	Weight
0	13500	23.0	46986	Diesel	90	1.0	0	2000	three	1165
1	13750	23.0	72937	Diesel	90	1.0	0	2000	3	1165
2	13950	24.0	41711	Diesel	90	NaN	0	2000	3	1165
3	14950	26.0	48000	Diesel	90	0.0	0	2000	3	1165
4	13750	30.0	38500	Diesel	90	0.0	0	2000	3	1170

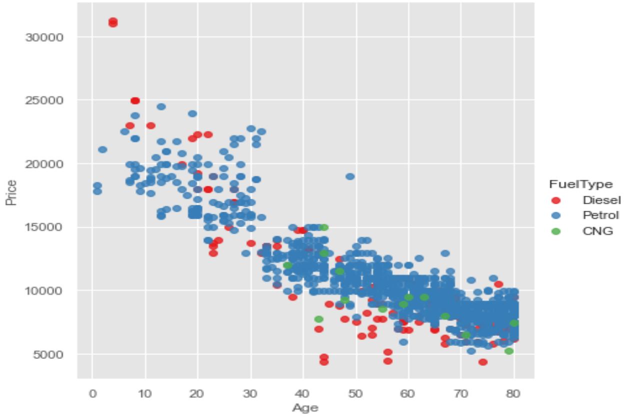
Scatter Plot:

```
plt.style.use("ggplot")
plt.figure(figsize=(8,6))
sns.regplot(x = cars_data["Age"], y = cars_data["Price"])
plt.show()
```



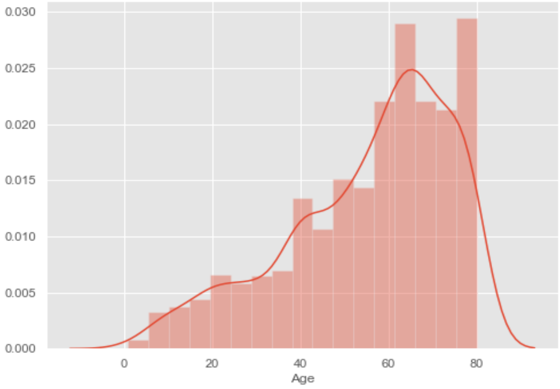
Scatter Plot (for 3 variables):

```
        sns.lmplot(x='Age', y='Price', data=cars_data,
         fit_reg=False,
         hue='FuelType',
         legend=True,
         palette="Set1",height=6)
```

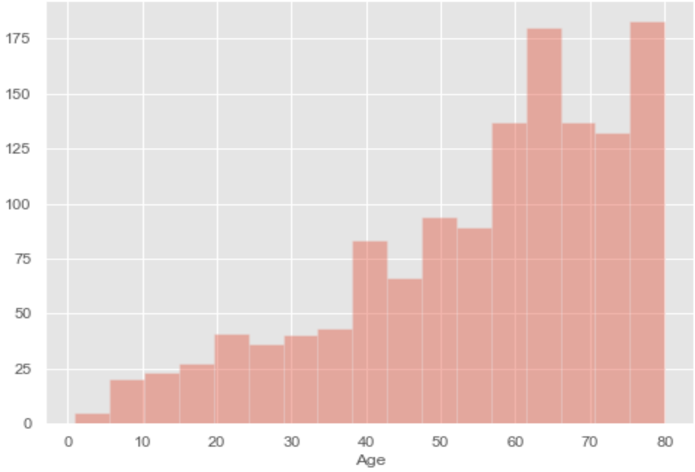


Histogram:

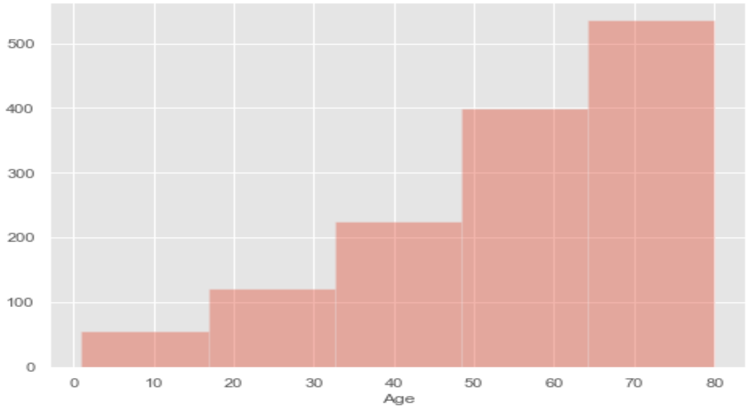
```
plt.figure(figsize=(8,6))
sns.distplot(cars_data['Age'])
plt.show()
```



```
plt.figure(figsize=(8,6))
sns.distplot(cars_data['Age'],kde=False)
plt.show()
```

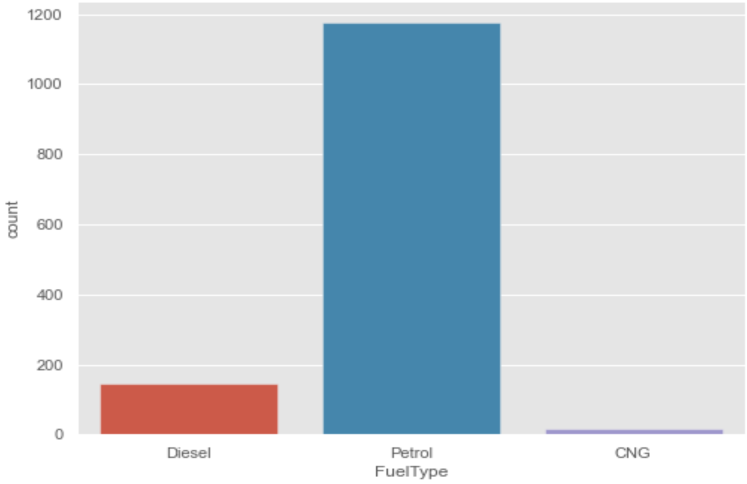


```
plt.figure(figsize=(8,6))
sns.distplot(cars_data['Age'],kde=False,bins=5)
plt.show()
```



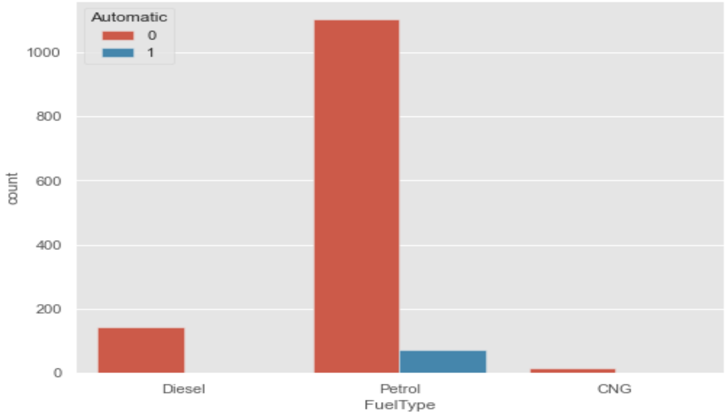
Bar Plot:

```
plt.figure(figsize=(8,6))
sns.countplot(x="FuelType", data=cars_data)
plt.show()
```



Grouped Bar Plot:

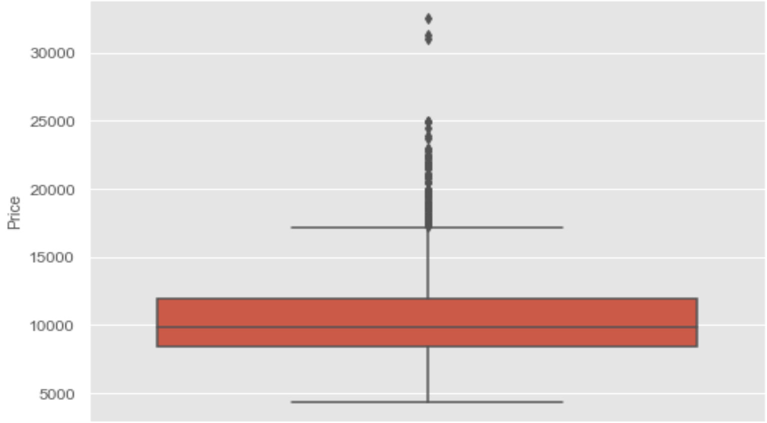
```
plt.figure(figsize=(8,6))
sns.countplot(x="FuelType", data=cars_data,
            hue="Automatic")
plt.show()
```



Box and Whiskers Plot:

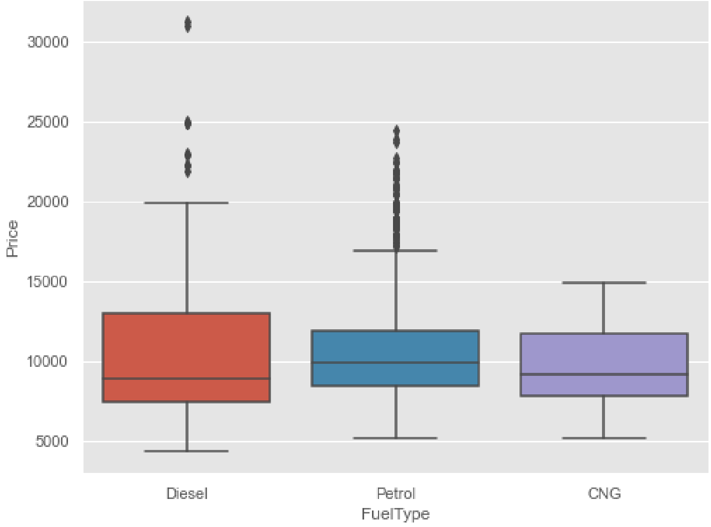
Box and whiskers plots are used for analyzing the detailed distribution of a dataset.

```
plt.figure(figsize=(8,6))
sns.boxplot(y=cars_data["Price"])
plt.show()
```



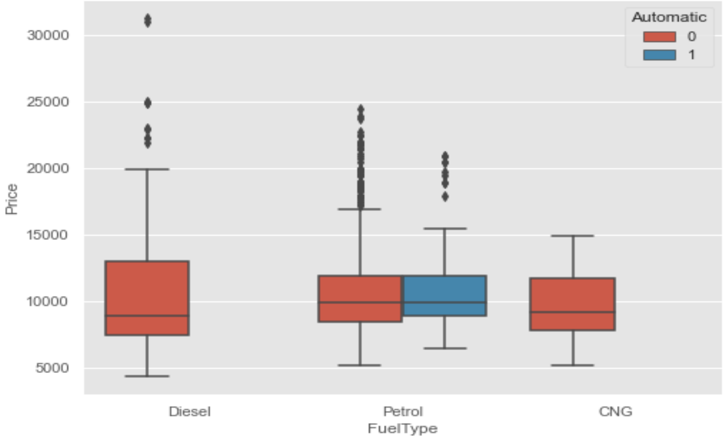
Box and Whiskers Plot(Numerical vs Categorical Variable):

```
plt.figure(figsize=(8,6))
sns.boxplot(x=cars_data["FuelType"],
          y=cars_data["Price"],
          )
plt.show()
```



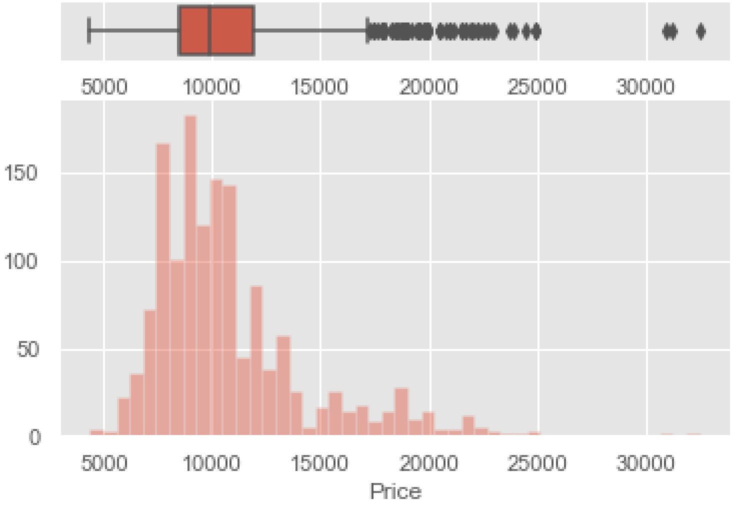
Grouped Box and Whiskers plot:

```
plt.figure(figsize=(8,6))
sns.boxplot(x="FuelType",
           y="Price",
           data=cars_data,
           hue="Automatic"
          )
plt.show()
```



Two plots on the same window:

```
f, (ax_box,ax_hist) = plt.subplots(2, gridspec_kw=
                        {"height_ratios":(.15,.85)})
sns.boxplot(cars_data["Price"], ax=ax_box)
sns.distplot(cars_data["Price"], ax=ax_hist, kde=False)
plt.show()
```

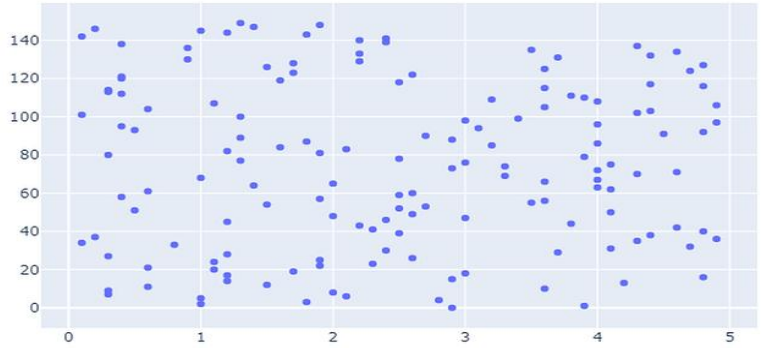


Graph plotting using plotly

Scatter Plot

```
# import all required libraries
import numpy as np
import plotly
import plotly.graph_objects as go
import plotly.offline as pyo
from plotly.offline import init_notebook_mode
 init_notebook_mode(connected=True)
 # generating 150 random integers
# from 1 to 50
x = np.random.randint(low=1, high=50, size=150)*0.1
 # generating 150 random integers
# from 1 to 50
y = np.random.randint(low=1, high=50, size=150)*0.1

# plotting scatter plot
fig = go.Figure(data=go.Scatter(x=x, y=y, mode='markers'))
 fig.show()
```

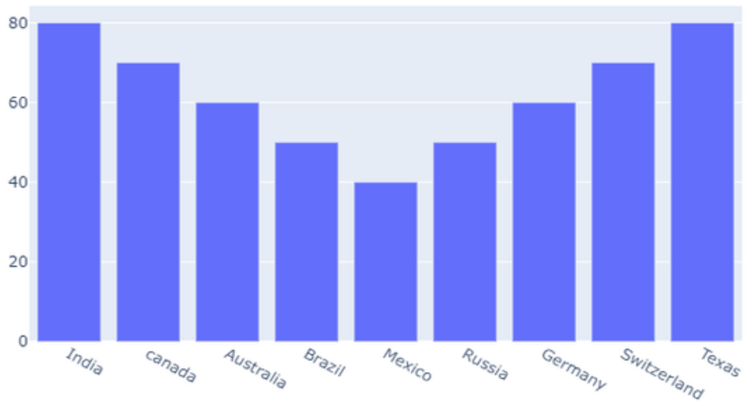
Output:



Bar charts

```
# import all required libraries
import numpy as np
import plotly
import plotly.graph_objects as go
import plotly.offline as pyo
from plotly.offline import init_notebook_mode
 init_notebook_mode(connected = True)
 # countries on x-axis
countries=['India', 'canada',
           'Australia','Brazil',
           'Mexico','Russia',
           'Germany','Switzerland',
           'Texas']
 # plotting corresponding y for each
# country in x
fig = go.Figure([go.Bar(x=countries,
                        y=[80,70,60,50,
                           40,50,60,70,80])])
 fig.show()
```

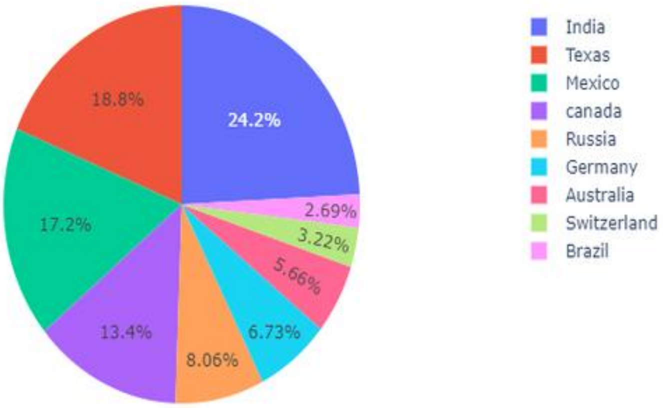
Output:



Pie chart

```
# import all required libraries
import numpy as np
import plotly
import plotly.graph_objects as go
import plotly.offline as pyo
from plotly.offline import init_notebook_mode
 init_notebook_mode(connected = True)
 # different individual parts in
# total chart
countries=['India', 'canada',
           'Australia','Brazil',
           'Mexico','Russia',
           'Germany','Switzerland',
           'Texas']
 # values corresponding to each
# individual country present in
# countries
values = [4500, 2500, 1053, 500,
          3200, 1500, 1253, 600, 3500]
 # plotting pie chart
fig = go.Figure(data=[go.Pie(labels=countries,
                      values=values)])
 fig.show()
```

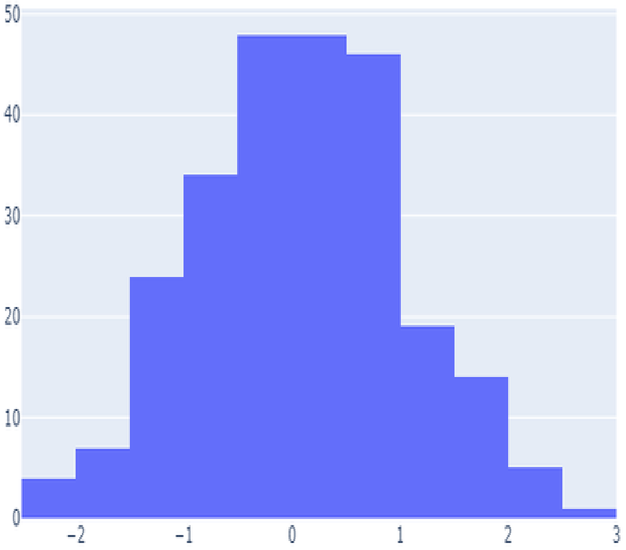
Output:



Histogram

```
# import all required libraries
import numpy as np
import plotly
import plotly.graph_objects as go
import plotly.offline as pyo
from plotly.offline import init_notebook_mode
init_notebook_mode(connected = True)
 # save the state of random
np.random.seed(42)
 # generating 250 random numbers
x = np.random.randn(250)
 # plotting histogram for x
fig = go.Figure(data=[go.Histogram(x=x)])
 fig.show()
```

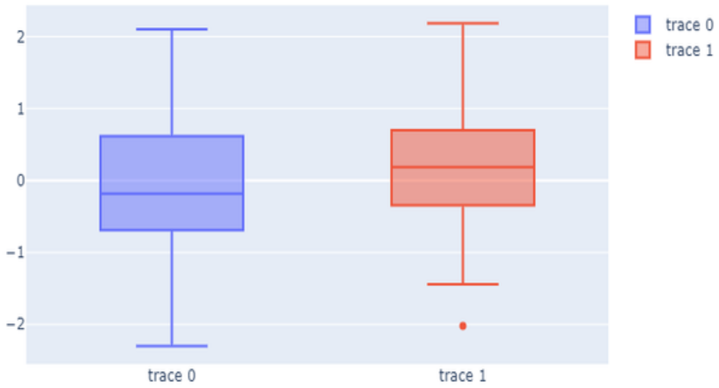
Output:



Box plot

```
# import all required libraries
import numpy as np
import plotly
import plotly.graph_objects as go
import plotly.offline as pyo
from plotly.offline import init_notebook_mode
 init_notebook_mode(connected = True)
 np.random.seed(42)
 # generating 50 random numbers
y = np.random.randn(50)
 # generating 50 random numbers
y1 = np.random.randn(50)
fig = go.Figure()
 # updating the figure with y
fig.add_trace(go.Box(y=y))
 # updating the figure with y1
fig.add_trace(go.Box(y=y1))
 fig.show()
```

Output:



Interactive data visualization using bokeh

Bokeh is a data visualization library in Python that provides high-performance interactive charts and plots. Bokeh output can be obtained in various mediums like notebook, html and server. It is possible to embed bokeh plots in Django and flask apps.

Bokeh provides two visualization interfaces to users:
bokeh.models *: A low level interface that provides high flexibility to application developers.*
bokeh.plotting *: A high level interface for creating visual glyphs.*

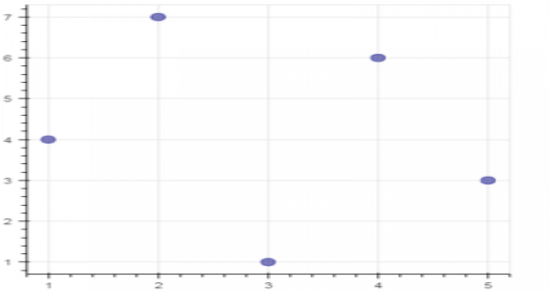
```
pip install bokeh
```

The dataset used for generating bokeh graphs is collected from Kaggle.
Code #1: Scatter Markers
To create scatter circle markers, circle() method is used.

```
# import modules
```

```
from bokeh.plotting import figure, output_notebook, show
  # output to notebook
output_notebook()
  # create figure
p = figure(plot_width = 400, plot_height = 400)
  # add a circle renderer with
# size, color and alpha
p.circle([1, 2, 3, 4, 5], [4, 7, 1, 6, 3],
         size = 10, color = "navy", alpha = 0.5)
  # show the results
show(p)
```

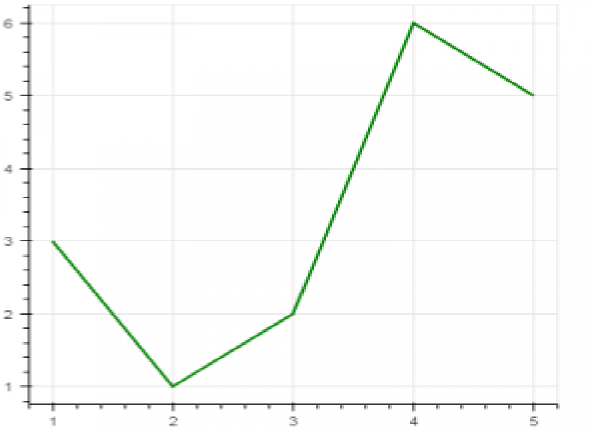
Output:



Code #2: Single line
To create a single line, line() method is used.

```
# import modules
from bokeh.plotting import figure, output_notebook, show
  # output to notebook
output_notebook()
  # create figure
p = figure(plot_width = 400, plot_height = 400)
 # add a line renderer
p.line([1, 2, 3, 4, 5], [3, 1, 2, 6, 5],
        line_width = 2, color = "green")
  # show the results
show(p)
```

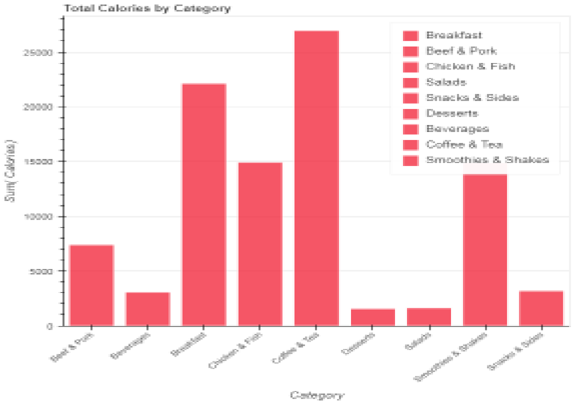
Output:



Code #3: Bar Chart
Bar chart presents categorical data with rectangular bars. The length of the bar is proportional to the values that are represented.

```
# import necessary modules
import pandas as pd
from bokeh.charts import Bar, output_notebook, show
  # output to notebook
output_notebook()
  # read data in dataframe
df = pd.read_csv("D:/kaggle/mcdonald/menu.csv")
  # create bar
p = Bar(df, "Category", values = "Calories",
        title = "Total Calories by Category",
                        legend = "top_right")
  # show the results
show(p)
```

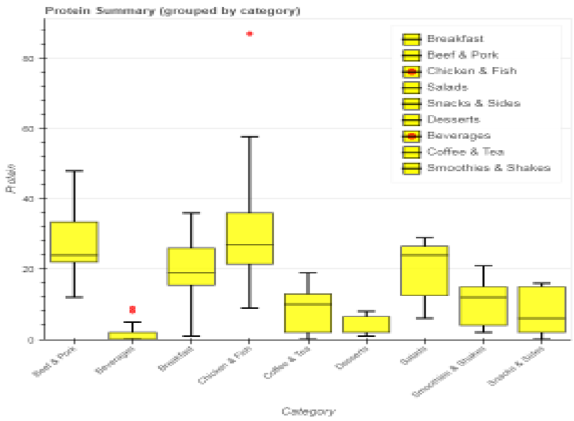
Output :



Code #4: Box Plot
Box plot is used to represent statistical data on a plot. It helps to summarize statistical properties of various data groups present in the data.

```
# import necessary modules
from bokeh.charts import BoxPlot, output_notebook, show
import pandas as pd
  # output to notebook
output_notebook()
  # read data in dataframe
df = pd.read_csv(r"D:/kaggle / mcdonald / menu.csv")
  # create bar
p = BoxPlot(df, values = "Protein", label = "Category",
            color = "yellow", title = "Protein Summary (grouped by category)",
             legend = "top_right")
  # show the results
show(p)
```

Output :

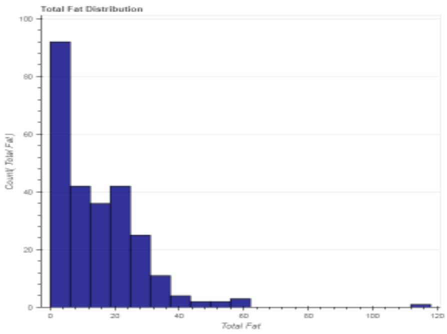


Code #5: Histogram

Histogram is used to represent distribution of numerical data. The height of a rectangle in a histogram is proportional to the frequency of values in a class interval.

```
# import necessary modules
from bokeh.charts import Histogram, output_notebook, show
import pandas as pd
  # output to notebook
output_notebook()
  # read data in dataframe
df = pd.read_csv(r"D:/kaggle / mcdonald / menu.csv")
  # create histogram
p = Histogram(df, values = "Total Fat",
               title = "Total Fat Distribution",
               color = "navy")
  # show the results
show(p)
```

Output :



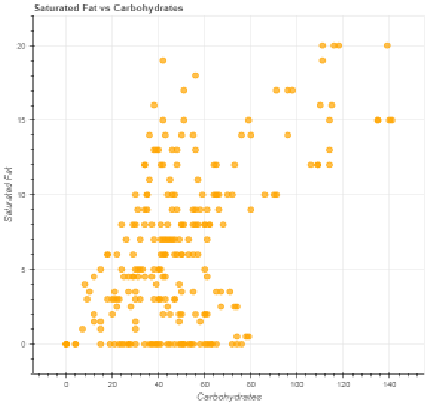
Code #6: Scatter plot

Scatter plot is used to plot values of two variables in a dataset. It helps to find correlation among the two variables that are selected.

```
# import necessary modules
from bokeh.charts import Scatter, output_notebook, show
import pandas as pd
  # output to notebook
output_notebook()
```

```
 # read data in dataframe
df = pd.read_csv(r"D:/kaggle / mcdonald / menu.csv")
  # create scatter plot
p = Scatter(df, x = "Carbohydrates", y = "Saturated Fat",
            title = "Saturated Fat vs Carbohydrates",
            xlabel = "Carbohydrates", ylabel = "Saturated Fat",
            color = "orange")
   # show the results
show(p)
```

Output :



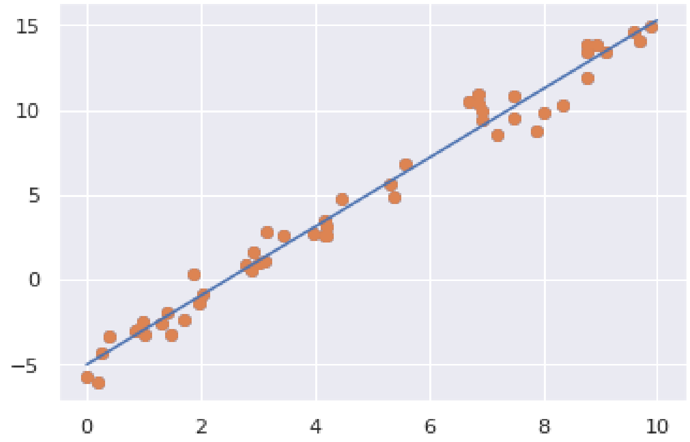
Ex.No10. Linear Regression

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
from sklearn.linear_model import LinearRegression
rng = np.random.RandomState(1)
x = 10 * rng.rand(50)
y = 2 * x - 5 + rng.randn(50)
plt.scatter(x, y);

model = LinearRegression(fit_intercept=True)

model.fit(x[:, np.newaxis], y)
```

```
xfit = np.linspace(0, 10, 1000)
yfit = model.predict(xfit[:, np.newaxis])

plt.scatter(x, y)
plt.plot(xfit, yfit);
print("Model slope:    ", model.coef_[0])
print("Model intercept:", model.intercept_)
rng = np.random.RandomState(1)
X = 10 * rng.rand(100, 3)
y = 0.5 + np.dot(X, [1.5, -2., 1.])

model.fit(X, y)
print(model.intercept_)
print(model.coef_)
```

Output

```
Model slope:     2.0272088103606953
Model intercept: -4.998577085553204
0.5000000000000051
[ 1.5 -2.   1. ]
```


15
10
5
0
−5
0
2
4
6
8
10