INSTALLATION PROCEDURE FOR PYTHON IN WINDOWS

STEP-1 : Select version of python to install. The installation procedure involves downloading the official python.exe installer and running on the system.

STEP-2 : Download python executable installer

Open the browser and navigate to Official python website

STEP-3 : Run executable installer

Run python installer once downloaded and have to select the install launcher for user. Add python to path checkboxes the add to path and select install now.

STEP-4 : Verify python was installed on windows.

Open the command prompt and type python.

STEP-5 : Verify pip is unstalled or not

If pip was not installed

Steps to download pip.

Step 1 : Download Pip get-pip.py :

Browse from official website or following command to get-pip.py file and run from command prompt

https://bootstrap.pypa.io/get-pip.py -o get-pip.py .

Step-2 : Install pip on windows

Python get-pip.py

If pip was already installed

enter command PIP-V , it shows the version of python.

STEP-6 : Add python path to environmental variables

INSTALLATION FOR PYTHON ON LINUX

STEP-1 : Open the command prompt and run

python3 --version

STEP 2 : If python is there, update python by

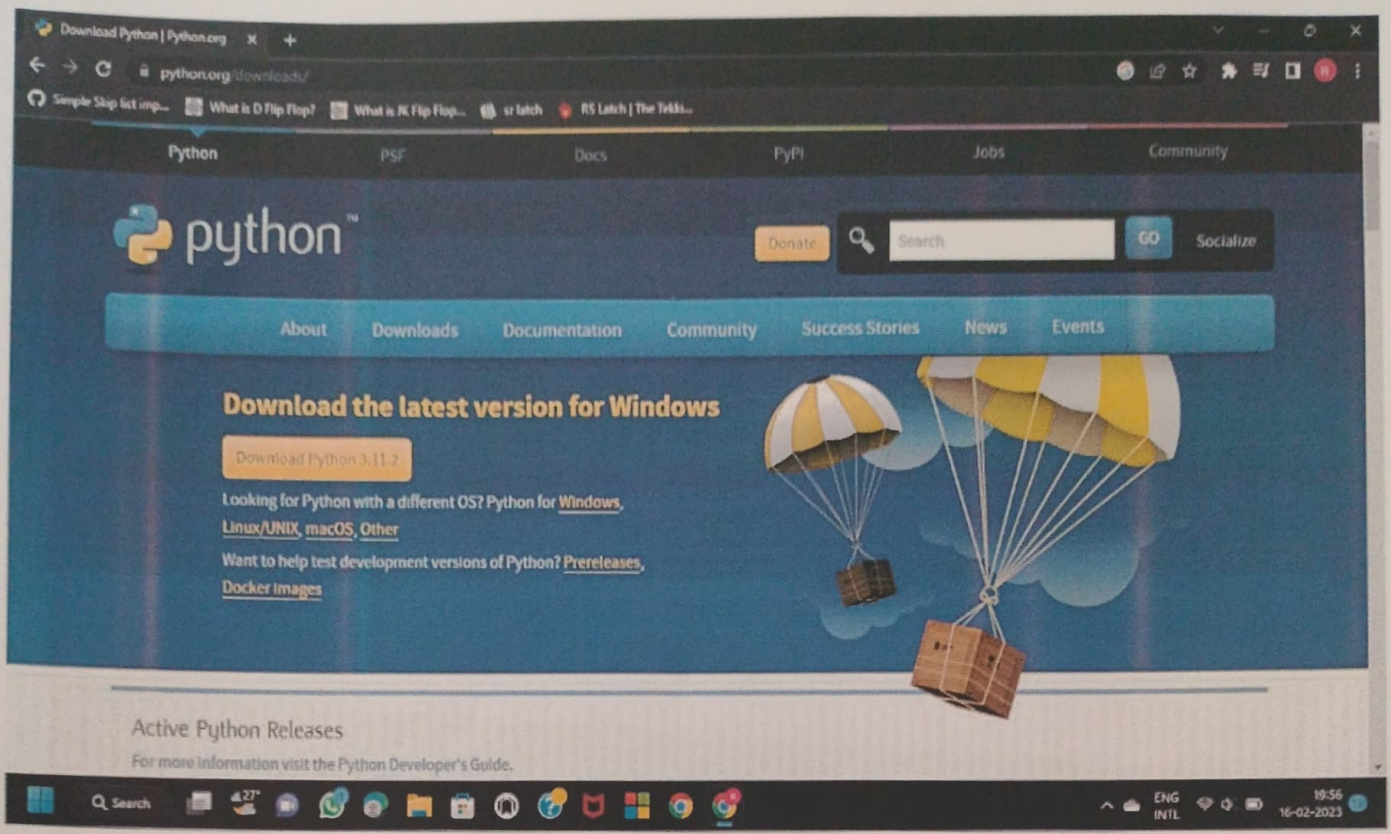
$ sudo apt -get update

If python is not present, install it by

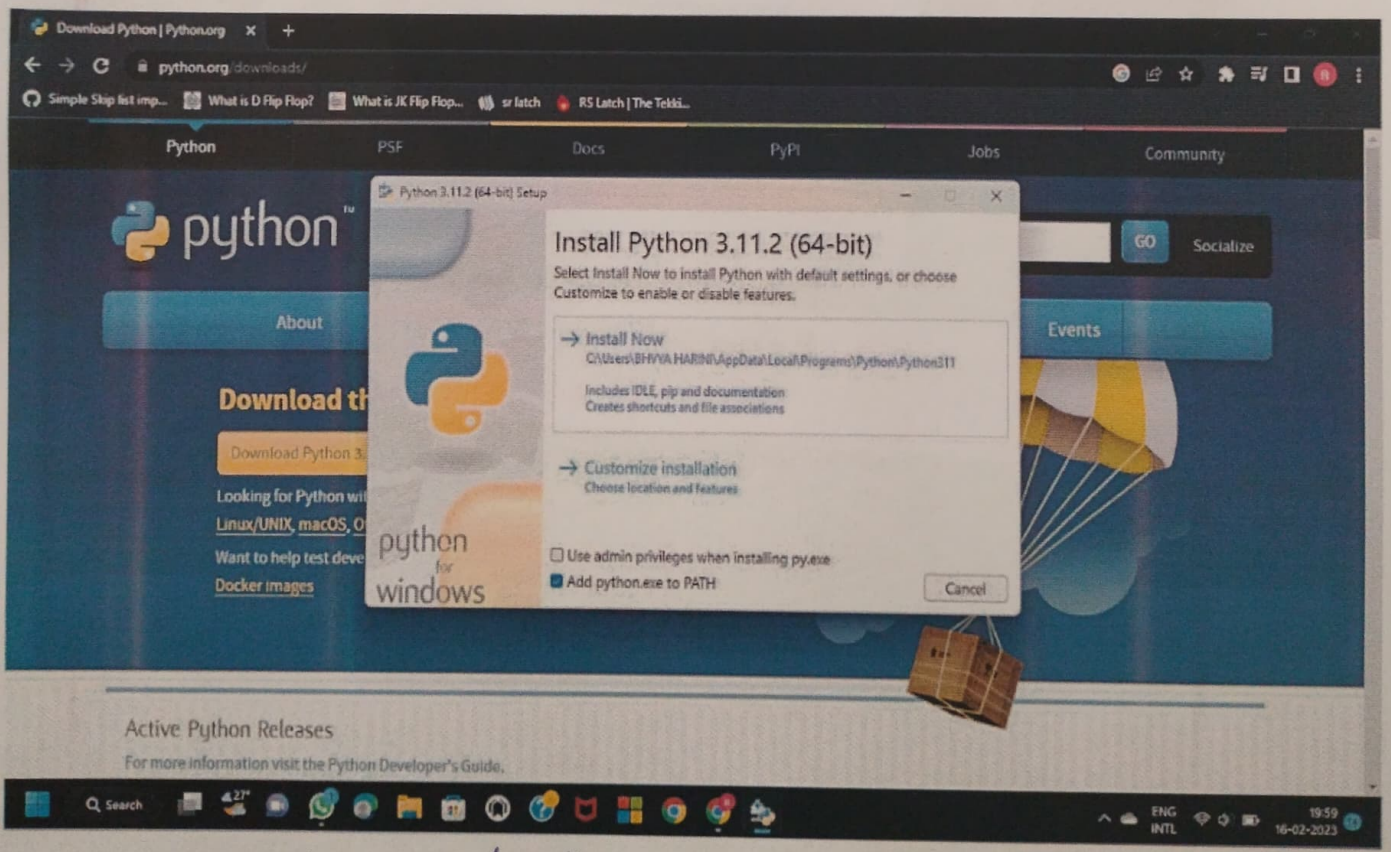
$ sudo apt -get install python 3.6

If you ar using different ubuntu environment try installing python by

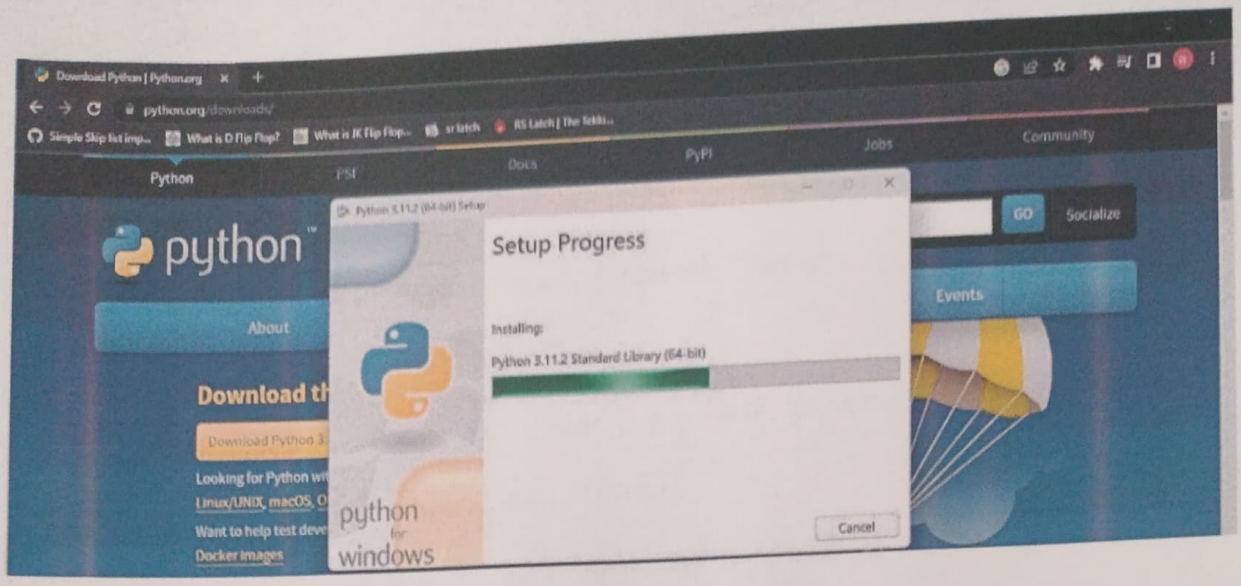
$ sudo dnf install python3



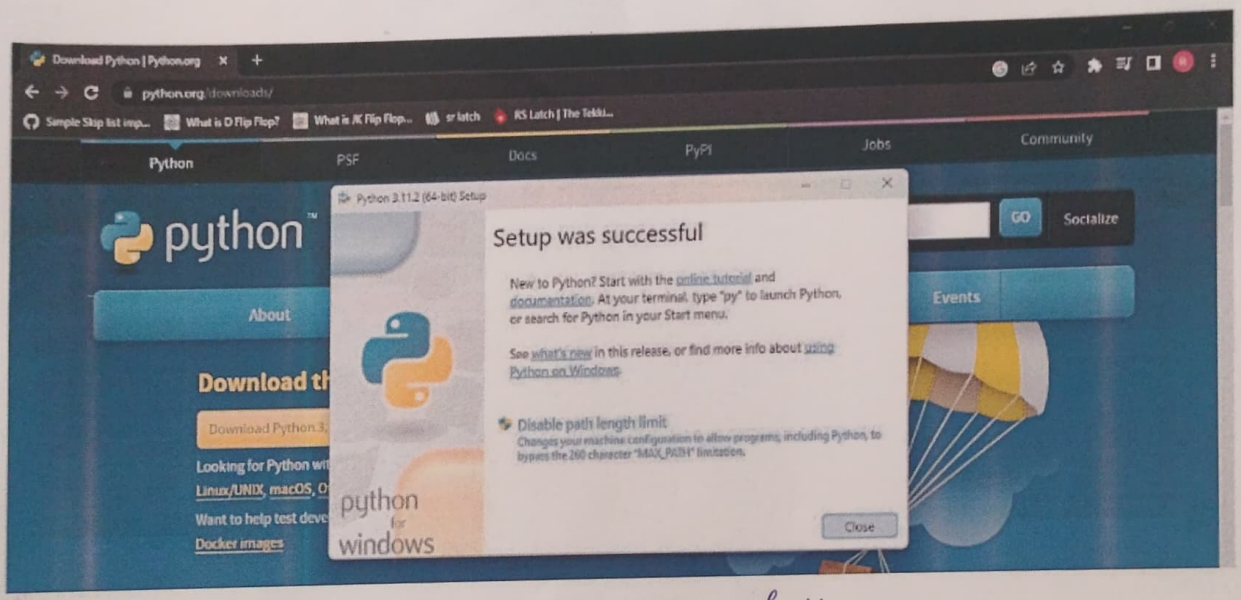
Selection of python version from official python website python.org.



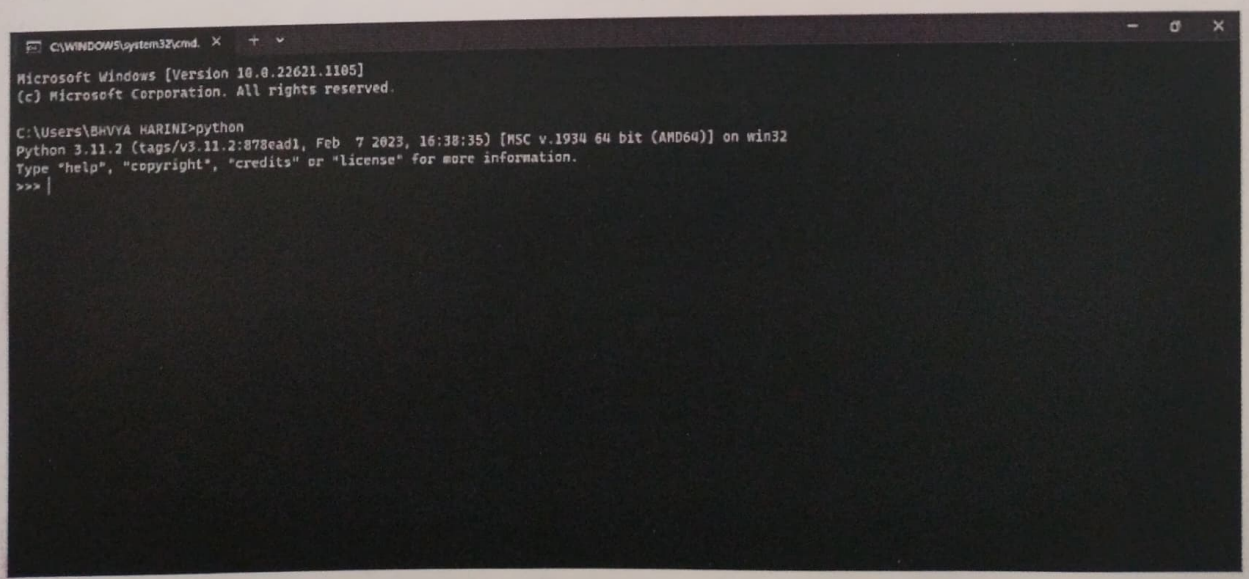
After downloading, open the install launcher checkbox add python.exe to path., click install now



Install process is taking place.



Python set up was successfull



open command prompt and type python, it gives the version of python installed.

AIM:

To perform basic operations on 1-D array using numpy

PROCEDURE:

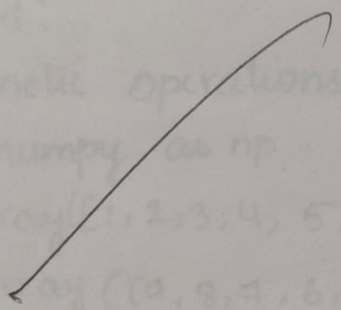
A numpy array is a grid of values, all of same type. In this program we created a 1D array. In this program we used type, size, shape commands. Type command gives the class of 1D array. Shape command gives the dimension of array whereas size gives the number of elements in the array. We changed the datatype of the elements using dtype attribute. We accessed the elements of the array using indexes. We performed slicing operation using a dot operator colon operator.

PROGRAM:

```
# Basic operations on 1D array
import numpy as np          # aliasing
a = np.array([2,4,6,8,10,12,14])   # Creation of 1D array
Print("One dimensional array")
print("array:",a)
print("Type of a:",type(a))   # Displays the class of array
print("Shape :" a.shape)      # Gives the dimensions rows, columns.
print("Size :", a.size)       # Gives no. of elements
a = np.array(a, dtype='float')   # changing the datatype
print(a)
print("Data type :",a.dtype)   # Accessing data type
print("Accessing elements:",a[3]," ",a[1])  # Accessing 2nd, 4th elements
print("Slicing :",a[1:4])   # Slicing from index 1 to 4
a[4]=1
print(a)                    # changing the value of element
```

OUTPUT:

```
//One dimension
One dimensional array
array : [2 4 6 8 10 12 14]
Type of a : <class 'numpy.ndarray'>
Shape : (7,)
Size : 7
After changing data type
[2. 4. 6. 8. 10. 12. 14.]
Data type : float64
Accessing elements : 8.0 4.0
Slicing : [4. 6. 8.]
[2. 4. 6. 8. 1. 12. 14.]
```



AIM:

To perform arithmetic operations on 1D array using numpy module.

PROCEDURE:

In this program we are going to perform arithmetic operations like addition, subtraction, multiplication, division of elements of array using arithmetic operators like '+', '-', '*', '/'. We performed sum of elements of array using np.sum() function. We performed square, square root of elements of the array using sqrt(), square functions. We also created an array using arrange operations which gives range of numbers from 0. by default or from range start to end with given step.

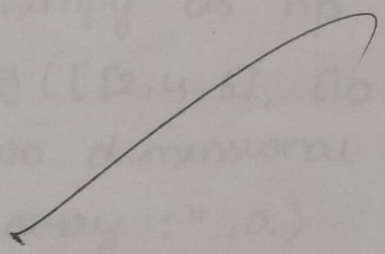
PROGRAM:

```
# Arithmetic operations on 1d array.
import numpy as np
a = np.array([1,2,3,4,5,6,7,8])
b = np.array([9,8,7,6,5,4,3,2])
print("arrays:", a, ",", b)
print("Sum of elements:", np.sum(a))
print("Addition:", a+b)  # addition of elements of arrays
print("Subtraction:", a-b) # Subtraction of elements of arrays
Print("Multiply:", a*b) # product of elements of array.
print("Divide:", a/b) # division of elements of arrays
print("squareroot of a:", np.sqrt(a))
# Square of elements of array
print("Square of a:", np.square(a))
```

```
print ("Arrange : ", np.arange(10))
#prints numbers from 0 to 9 with a step 1 by default
```

OUTPUT:

```
arrays : [1 2 3 4 5 6 7 8], [9 8 7 6 5 4 3 2]
Sum of elements: 36
Addition : [10 10 10 10 10 10 10 10]
Subtract : [-8 -6 -4 -2 0 2 4 6]
Multiply : [9 16 21 24 25 24 21 16]
Divide : [0.11111111 0.25 0.42857143 0.66666667 1. 1.5
2.33333333 4. ]
Squareroot of a: [1. 1.41421356 1.73205081 2. 2.23606798
2.44948974 2.64575131 2.82842712]
Square of a : [1 4 9 16 25 36 49 64]
Arange: [0 1 2 3 4 5 6 7 8 9]
```



AIM:

To perform basic operations on 2D array using numpy module.

PROCEDURE:

In this program we created a 2D array using np.array([[ ],[ ]]. We used functions like array.shape, type, size commands. Type command gives the class of 2D array. shape command gives the number of dimension of the array whereas size gives the number of elements in the array. We changed the datatype of the elements using dtype attribute. We accessed the elements of the array using indices. We performed slicing operation using (colon), comma operator.

PROGRAM:

```
# Basic operations on 2D array.
import numpy as np  # aliasing
a=np.array([[2,4,6], [10,12,14]])
print("Two dimensional array")
print ("array :",a)
print ("shape:", a.shape)  # Gives dimensions (rows, columns)
print ("size:", a.size)  # Gives number of elements of array
# Displays the class of the array.
print ("Type:", type(a))
# Changing data type of elements of array.
a=np.array(a, dtype='float')
print(a)
# Accessing data type of elements of array
```

```
print("Data type : ", a.dtype)
#Accessing a[1][0] and a[0][1] elements
print("Accessing elements :", a[1][0], "  " a[0][1])
#Slicing of 2D array
print("Slicing :", a[:1, :2])
 a[1][1] = 1  # changing element value
print("after changing element", a)
```

OUTPUT:

```
Two dimensional array.
array : [[2 4 6]
  [10 12 14]]
Shape : (2, 3)
Size : 6
Type : <class 'numpy.ndarray'>
[[2. 4. 6.]
  [10. 12. 14.]]
Data type : float 64
Accessing elements : 10.0  4.0
Slicing : [[2. 4.]]
after changing the element [[2. 4. 6]
[10. 1. 14.]]
```

AIM:

To perform arithmetic operations of 2-D array using numpy module.

PROCEDURE:

In this program we are going to perform arithmetic operations like addition, subtraction, multiplication, division of elements of array using arithmetic operators like '+', '-', '*', '/'. We performed sum of elements of the array using np.sum() function. We performed square, square root of elements of array using square(), sqrt() functions. We performed the dot product of two 2-D array. We performed transpose operation for one array.

PROGRAM:

```
#Arithmetic operations on 2d array.
import numpy as np  #aliasing
a=np.array([[1,2],[5,6]])
b=np.array([[9,8],[5,4]])
#Sum of elements of the array
print("sum of elements:",np.sum(a))
#Addition of elements of array.
print("Addition :",a+b)
# Subtraction of elements of array)
print("Subtract :",a-b)
# product of elements of array.
print("Divide:",a/b)
#Square root of elements of array
print("Square root of a:",np.sqrt(a))
```

```
print ("Squar of a:", np.Squar(a))
#Dot product of the arrays:
print("dot product:", np.dot(a,b))
#Transpose of array
print("Transpose:",a.T).
```

OUTPUT:

```
arrays: [[1 2]
[5, 6]], [[9 8]
[5 4]]
Sum of elements: 14.
Addition: [[10 10]
[10 10]]
Subtract: [[-8 -6]
[0 2]]
Multiply: [[9 16]
[25 24]]
Divide: [[0.11111111 0.25 ]
[1. 1.5 ]]
Squar root of a: [[1. 1.41421356]
[2.3606798 2.44948974]]
Squar of a: [[1 4]
[25 36]]
dot product: [[19 16]
[75 64]]
Transpose: [[1 5]
[2 6]].
```

AIM:

To perform slicing, indexing, dot operation, boolean indexing on 1d and 2D array using numpy.

PROCEDURE:

In this program, we created two arrays, a 1D and 2D array. We performed indexing of the elements in 1D and 2D array. We performed slicing operation by using indices. We even changed the datatype of the element. We changed the data type using dtype attribute into float, complex. We performed boolean indexing using a condition. We filtered the array according the given condition using boolean indexing. We done the dot product of two 2D arrays using dot operation.

PROGRAM:

```
# Slicing, indexing, boolean indexing on arrays
import numpy as np
a = np.array([1,3,6,7,9])    # Creating arrays
b = np.array([[2,4,6], [3,6,9], [4,8,12]])
# Indexing 1D and 2D arrays
print("Indexing 1D array 2nd element:", a[1])  # 2nd element
print("Indexing 2D array row1 column1 element " b[0,0])
# Slicing operations
# Slicing 1D array from second to 5th element
print("slicing 1d array:", a[1:5])
# Slicing 2D array row2 and column 1 to 3.
print("slicing on 2d array:", b[1:2, :3])
# Significance of data type in arrays
# Printing current data type
```

```
print ("Data type of array :", a.dtype)
#changing the data type to float
b= np.array (b, dtype = 'float')
print ("data type changed to float :", b.dtype)
print (b)
#changing the data type to complex
c= np.array (a, dtype = 'complex')
print ("data type changed to complex :", c.dtype)
print (c)
# Boolean indexing
x = (a%.2==0)  # condition
#Printing boolean values according to the condition
print ("Boolean indexing for even numbers :", a[x])
print ("Boolian indexing :", x)
d= np.array ([[1,2,3], [4,5,6], [7,8,9]])
#Dot product of b and d 2D array.
print ("Dot product of arrays:", d.dot (b))
```

OUTPUT:

```
Indexing 1d array 2nd element : 3
Indexing 2D array row1 column1 element : 2
Slicing 1d array : [3 6 7 9]
Slicing 2D array : [[3 6 9]]
Data type of array : int32
data type changed to float : float 64
[[2. 4. 6.]
 [3. 6. 9.]
 [4. 8. 12.]]
```

data type changed to Complex : Complex128

[1.+0.j 3+0.j 6+0.j 7.+0.j 9+0.j]

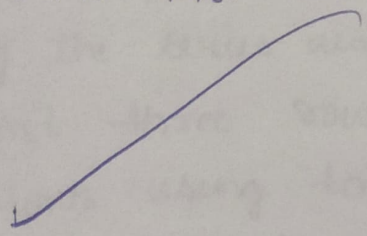
Boolean indexing of even numbers : [6]

Boolean indexing : [False False True False False]

Dot product of array : [[20. 40. 60.]

[47. 94. 141.]

[74. 148. 222.]]



AIM:

To perform basic operations on series using pandas.

PROCEDURE:

A panda series is like a column in a table. It is a one dimensional array holding data of any type. In this program, we created a panda series. We explicitly gave indices to the series. We printed the shape, size, class of the series using size, sha, class type(). We accessed the first three rows using head() command. We accessed the last rows using tail() command. Indexing is also performed to access the elements in series. We took 2 series and added the two series using '+' operator

PROGRAM:

```
# Basic operations on series
import pandas as pd
import numpy as np
l = ['Vaish', 'Unnathi', 'Chethana', 'Nikhitha', 'Neha']
#creating a series
s = pd.Series(l)
print(s)
k = ['s1', 's2', 's3', 's4', 's5']   # explicit indices
# Indexing the series
s1 = pd.Series(l, index = k)
print(s1)
print("class:", type(s))    # class of series
print("Head elements \n", s1.head(3))  #first 3 elements in series
print("Tail elements \n", s1.tail(3)) # last 3 elements in series
print("Datatypes :", s1.dtype)  #data type of elements
print("Shape:", s1.shape)  # Shape of the series
```

```
print ("Size:", S1.Size)  # Size of the series.
print ("1st element:", S1[0])  #indexing
AF pd.Series ({'apple': 20, 'banana':23, 'mango':2, 'strawberry':27})
b=pd.Series ({'banana':19, 'cherry':35, 'kiwi':7, 'avacado':9})
print ("Addition of series:", a+b)
```

OUTPUT:

```
0    Vaish
1    Unnathi
2    Chethana
3    Nikitha
4    Neha
dtype: Object
S1   Vaish
S2   Unnathi
S3   Chethana
S4   Nikhitha
S5   Neha
dtype: Object
class: <class 'pandas.core.series.Series'>
Head elements
S1   Vaish
S2   Unnathi
S3   Chethana
dtype: Object
Tail elements
S3   Chethana
S4   Nikhitha
S5   Neha
dtype: Object
Data type: Object
Shape: (5,)
Size 5
1st element Vaish
Addition of Series: apple  NaN.
Avacado    NaN
banana     42.0
cherry     NaN
Kiwi       NaN
mango      NaN
strawberry NaN
dtype: float64.
```

29/11/22

AIM :

To perform basic operations on dataframes.

PROCEDURE :

Pandas Dataframe is two dimensional size mutable, potentially heterogeneous tabular data structure with labeled axes. It consists of three components data, rows and columns. In this program, we created a DataFrame by importing pandas we set the index for the rows using index function. We performed slicing operation on the rows. We used loc and iloc which allows to select certain rows in the data set We even created a frequency distribution using value-counts() function We sorted the values of a column using sort_values() function. We tried to rename the columns by accessing the columns.

PROGRAM :

```
# Basic operations on dataframes
import pandas as pd
d = pd.DataFrame({'Name': ['a','b','c'], 'Class': ['CSE','ECE','EEE'],
        'Age': [21,23,22], 'Address': ["Hyderabad", 'Chennai', 'kolkata']})
d.index = ['S1', 'S2', 'S3']  # Setting indices
print(d)
# Slicing on rows print("slicing\n");
Print(d[0:2])
# Accessing rows using loc function
S = d.loc[['S2','S3']]
print("loc function :", S);
#Accessing rows using iloc function
;11 = d.iloc[1:, 2:3]
```

```
print('rows and columns\n', i11)
i12 = df1.iloc[1]
print("iloc function : ", i12)
#Creating another dataFrame
df2 = pd.DataFrame({'Fruit':['Apple', 'banana', 'cherry', 'pineapple'],
                    'Quantity' : [21, 21, 30, 2]})
#indexing
df2.index = ['a', 'd', 'c', 'b']
print(df2)
#Counting the index values
print(df2.index.value_counts())
#sorting values according to Quantity print("sort_values").
print(df2.sort_values('Quantity'))
#renaming the columns
df2.columns = ['Fruit Name', 'Quantities'], print("Renaming\n");
print(df2)
```

OUTPUT:

	Name	Class	Age	Address
S1	a	CSE	21	Hyderabad
S2	b	ECE	23	Chennai
S3	c	EEE	22	kolkata

Slicing

	Name	class	Age	Address
S1	a	CSE	21	Hyderabad
S2	b	ECE	23	Chennai

loc function:

	Name	Class	Age	Address
S2	b	ECE	23	Chennai
S3	c	EEE	22	kolkata

rows and columns

	Age
s_2	23
s_3	22

iloc function

Name	b
Class	ECE
Age	23
Address	Chennai

Name: s_2, dtype: Object

	Fruit	Quantity
a	Apple	21
d	banana	21
c	cherry	30
b	pineapple	2

value_counts()

a	1
d	1
c	1
b	1

dtype: int 64

Sort values

	Fruit	Quantity
b	pineapple	2
a	Apple	21
d	banana	21
c	cherry	30

Renaming

	Fruit Name	Quantities
a	Apple	21
d	banana	21
c	Cherry	30
b	pineapple	2

AIM:

To perform column and conditional operations on dataframes

PROCEDURE:

Dataframe is a two dimensional size, mutable, potentially heterogenous tabular data structure with labled axes It consists of three components data, rows and columns. In this program, we created a data frame using pandas we explicitly gave indices to the dataframe. We removed a column in the dataframe using pop(). We even created a new column from an existing column. We accessed the description of the data frame using description(). We accessed columns based on min(), agg(), max() of some columns in dataframe. We accessed the columns based on the condition provided. We checked if there are any null values. We even filled the null values. We ranked each variable according to the value.

PROGRAM:

```
import pandas as pd
import numpy as np
df. pd. DataFrame ({'Name': ['Joyce', 'William', 'Jonathan', 'Lucas',
'Billy'], 'Designation': ['Sr. Manager', 'CEO', 'Web Developer', 'Senior
HR', 'Data Analyst'], 'Salary': [200000, 300000, 50000, 50000, 20000],
'Wrong': [1,2,3,4,5], 'Experience': [12, 15, 2.3, np.nan]})
# explicit indexing
df. index = ['E1', 'E2', 'E3', 'E4', 'E5']
print ('DataFrame: \n', df)
# Deleting a column
df. pop ('Wrong')
```

```
print ('Drop:\n', df)
df ['Bonus'] = df.eval ('Salary * 0.1')
# Adding an extra column
print ("Adding:\n',df)
# Description of the dataframe
print ('Description', df.describe ())
# Grouping the column according to min() of salary
print ('Groupby :\n', df.groupby ('Name') .Salary.min ())
print ('Groupby 2:\n', df.groupby ('Name'). Salary .agg (['min', 'max',
                                                        'count']))
# Accessing column by a condition
print ('Filtering :\n' , df [df.Bonus >5000])
print ('Filtering2 :\n', df.loc [df.Bonus >5000 : 1])
# checking the null values.
print ('Isnull :\n', df. isnull() ).
# Counting the null values.
print ('Isnull Sum:\n', df.isnull .sum())
# Filling na value
print ("Fill Null value :\n', df.fillna (0))
print ('Rank :\n', df rank())
# Rank given according to values
```

OUTPUT:

DataFrame:

	Name	Designation	Salary	Wrong	Experience
E1	Joyce	Sr. Manager	200000	1	12.0
E2	William	CEO	300000	2	15.0
E3	Jonathan	Web Developer	50000	3	2.0
E4	Lucas	Senior HR	50000	4	3.0
E5	Billy	Data Analyst	20000	5	NAN.

Drop:

	Name	Designation	Salary	experience
E1	Joyce	Sr. Manager	200000	12.0
E2	William	CEO	300000	15.0
E3	Jonathan	Web Developer	50000	2.0
E4	Lucas	Senior HR	50000	3.0
E5	Billy	Data Analyst	20000	NaN

Adding:

	Name	Designation	Salary	Experience	Bonus
E1	Joyce	Sr. Manager	200000	12.0	40000.0
E2	William	CEO	300000	15.0	60000.0
E3	Jonathan	Web Developer	50000	2.0	10000.0
E4	Lucas	Senior HR	50000	3.0	10000.0
E5	Billy	Data Analyst	20000	NaN	4000.0

Description	Salary	Experience	Bonus
Count	5.000000	4.000000	5.000000
mean	124000.000000	8.000000	24800.000000
std	120954.000000	6.480741	24190.907383
min	20000.000000	2.000000	4000.000000
25%	50000.000000	2.750000	10000.000000
50%	50000.000000	7.500000	40000.000000
75%	200000.000000	12.750000	40000.000000
max	300000.000000	15.000000	60000.000000

Groupby.

Name	
Billy	20000
Jonathon	50000
Joyce	200000
Lucas	50000
William	300000

Name: Salary dtype: int64

Groupby 2 :

Experience	min	max	count
2.0	50000	50000	1
3.0	50000	50000	1
12.0	200000	200000	1
15.0	300000	300000	1

Filtering :

	Name	Designation	Salary	Experience	Bonus
E1	Joyce	Sr Manager	200000	12.0	400000
E2	William	CEO	300000	15.0	60000.0
E3	Jonathan	Web Developer	50000	2.0	10000.0
E4	Lucas	Senior HR	50000	3.0	10000.0

Filtering 2 :

	Name	Designation	Salary	Experience	Bonus
E1	Joyce	Sr. Manager	200000	12.0	40000.0
E2	William	CEO	300000	15.0	60000.0
E3	Jonathan	Web Developer	50000	2.0	10000.0
E4	Lucas	Senior HR	50000	3.0	10000.0

Isnull :

	Name	Designation	Salary	Experience	Bonus
E1	JoyceFalse	False	False	False	False
E2	False	False	False	False	False
E3	False	False	False	False	False
E4	False	False	False	False	False
E5	False	False	False	FalseTrue	False

Isnull sum :

Name	0
Designation	0
Salary	0
Experience	1

Bonus 0
dtype : int 64
Fillna value :

	Name	Designation	Salary	Experience	Bonus
E1	Joyce	Gr. Manager	200000	12.0	40000.0
E2	William	CEO	300000	15.0	60000.0
E3	Jonathan	Web Developer	50000	2.0	10000.0
E4	Lucas	Senior HR	50000	3.0	10000.0
E5	Billy	Data Analyst	20000	0.0	4000.0

Rank :

	Name	Designation	Salary	Experience	Bonus
E1	3.0	4.0	4.0	3.0	4.0
E2	5.0	1.0	5.0	4.0	5.0
E3	2.0	5.0	2.5	1.0	2.5
E4	4.0	3.0	2.5	2.0	2.5
E5	1.0	2.0	1.0	NaN	1.0

AIM:

To extract data from a csv file and to perform different operations and functions.

PROCEDURE:

In this program we make use of csv files. We can read csv files using pandas package. We use the function read_csv to read the file. The top elements and the last elements can be accessed by using head() and tail() functions. We accessed size, shape of the dataset. We found out the mean(), max() for finding mean of the values and maximum of the various. We can find variance and standard deviation using var() and std() functions. We can get description using describe(). We can check null values using isnull().

PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
#reading csv file
a = pd.read_csv('IRIS.csv')
# first 2 rows
print("Head:\n", a.head(2))
#last 2 rows
print("tail:\n", a.tail(2))
# shape of the dataset
print("shape: ", a.shape)
#size of the dataset
print("size: ", a.size)
#mean of sepal_length from dataset and maximum of it
print("sepal_length mean: ", a['sepal_length'].mean())
```

```
print("max of petal length: ", a['petal_length'].max())
#Variance and standard deviation of dataset
print("Variance: ", a.var())
print("Standard Deviation: ", a.std())
#Description of the dataset
print("Describe:", a.describe(include=object))
print("Null Values:", a.isnull().sum())
```

OUTPUT:

Head

	sepal_length	sepal_width	petal_length	petal width	species
0	5.1	3.5	1.4	0.2	Iris_setosa
1	4.9	3.0	1.4	0.2	Iris-setosa

Tail

	sepal_length	sepal_width	petal_length	petal_width	species
148	6.2	3.4	5.4	2.3	Iris_Virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

shape : (150, 5)

size : 750

Sepal ~~of~~ length mean : 5.84333334

Max of petal_length : 6.9

Variance:

Sepal_length	0.685694
Sepal_width	0.188004
petal_length	0.113179
petal_width	0.582414

dtype : float 64.

Standard deviation

Sepal_length	0.828066
sepal_width	0.433594
petal_length	1.764420
petal_width	0.763161

dtype : float 64

Describe

	species
count	150

AIM:

To plot different types of graphs using matplotlib

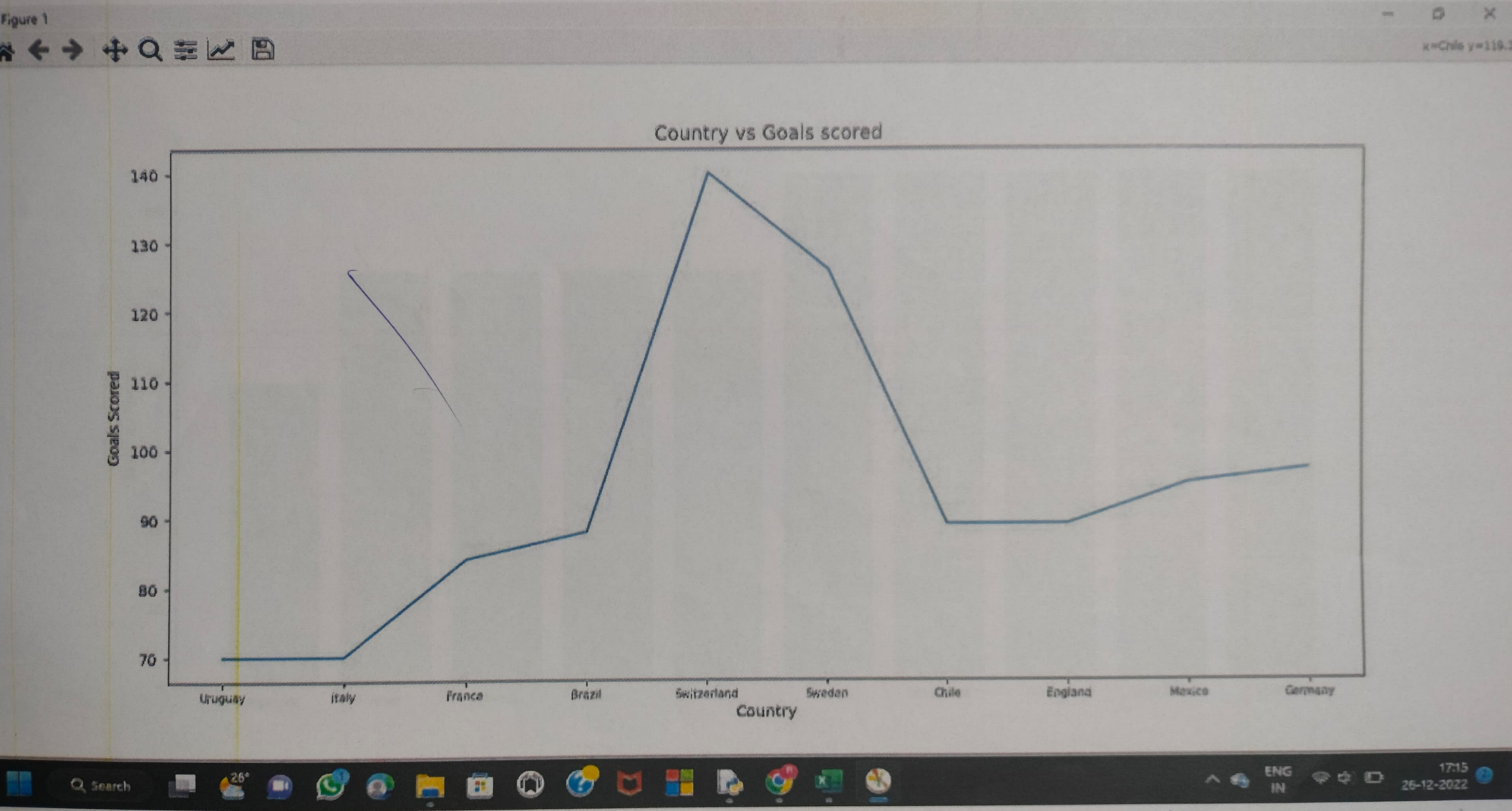
PROCEDURE:

Matplotlib is a python library used to create 2D graphs and plots. It has module pyplot which makes easy for plotting by providing features to control line styles, font, formatting axes. It has wide variety of graphs namely histogram, bar charts, power spectra, pie, boxplot, etc. In this program we used variety of graphs using a dataset. We used xlabel, ylabel for labelling the axes, title to provide a name for graph. We used hatching for bar graph. We used xticks, yticks and fontsize for styling the graphs.

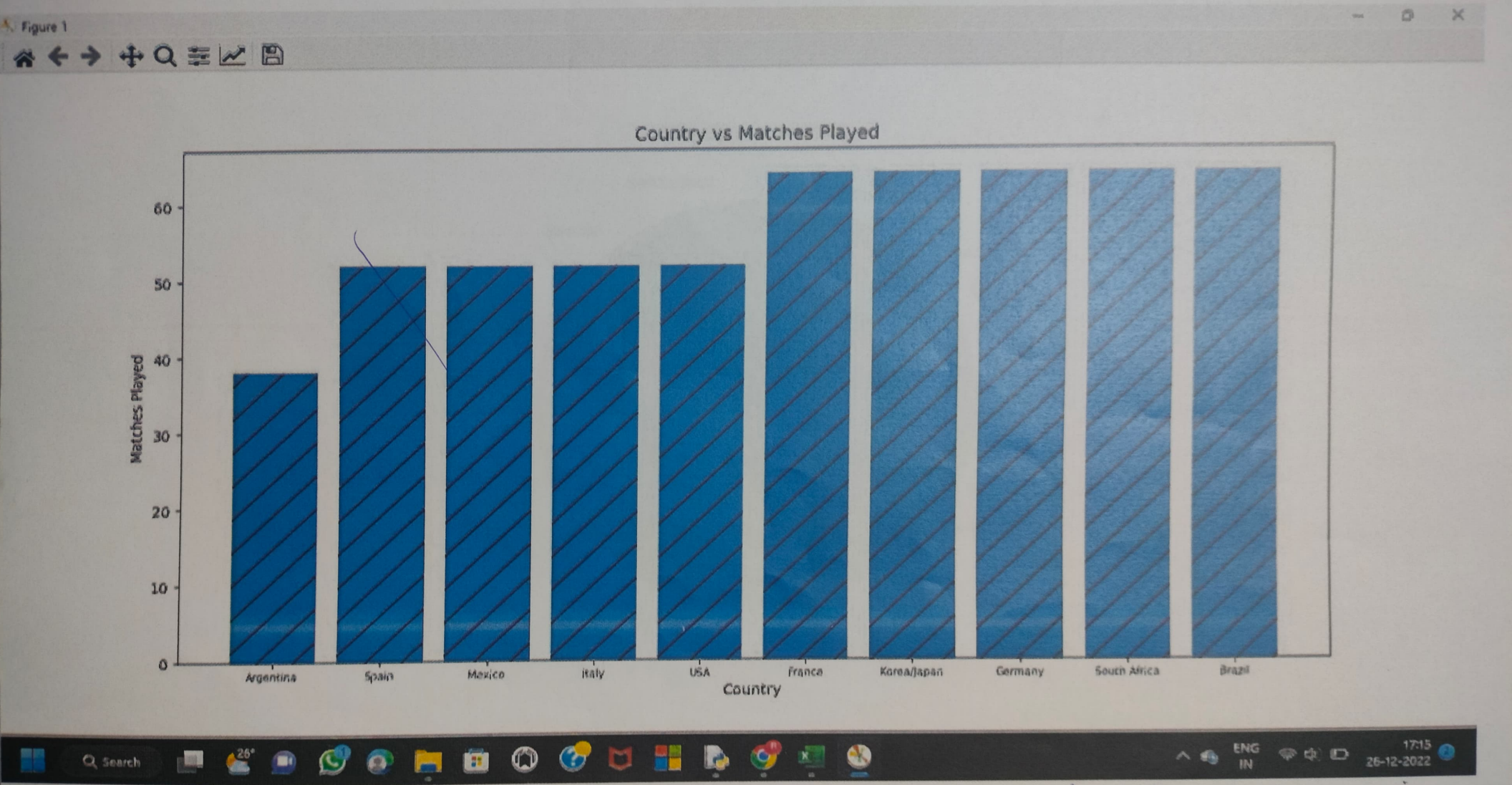
PROGRAM:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
#reading a dataset
a = pd.read_csv ('WorldCup.csv')
#Line plot
plt.plot (a.Country [0:10], a.GoalsScored [0:10])
# title to the graph
plt.title ("Country vs Goals scored")
#labeling axes
plt.xlabel ('Country')
plt.ylabel('Goals scored')
plt.xticks (fontsize = 7)  # Size of ticks
plt.show() # for providing the output graph.
# Bar graph
plt.bar (a.Country [10:20], a.MatchesPlayed [10:20], hatch= '/')
plt.title ('Country vs Matches Played')          #hatching
```

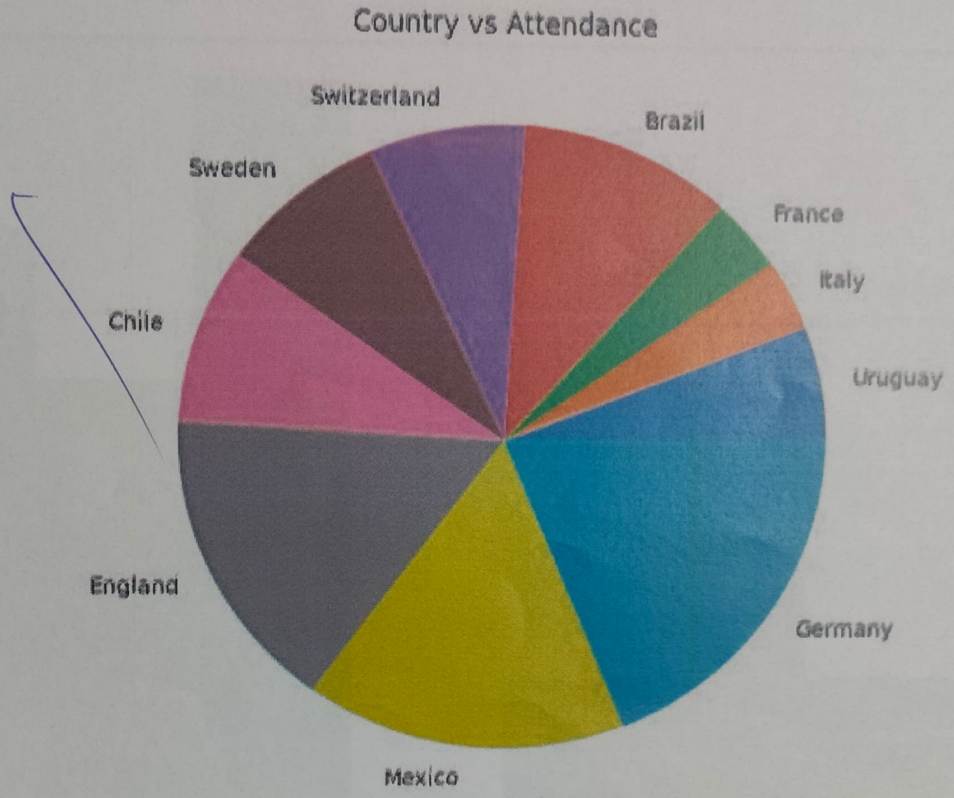
```
plt.xlabel("Country")
plt.ylabel("Matches played")
plt.xticks(fontsize=7)
plt.show()
b = a['Winner'].values.tolist()
#histogram
plt.title("Country Vs Winners")
plt.xlabel("Country")
plt.ylabel("Winner")
plt.xticks(fontsize=7)
plt.show()
#Scatter plot
plt.scatter(a['Winner'], a['Runners-Up'])
plt.title('Winners Vs Runners-Up')
plt.ylabel("Runners-Up")
plt.xlabel('Winners')
plt.xticks(fontsize=7)
plt.show()
#Pie chart
plt.pie(a.Attendance[0:10], labels=a.Country[0:10])
plt.title("Country Vs Attendance")
plt.show()
#Boxplot
plt.boxplot([a['Matchesplayed']])
plt.title('Matches Played')
plt.show()
#stacked bar
a.plot(kind='bar', stacked=True)
plt.title('Description of Countries')
plt.show()
```



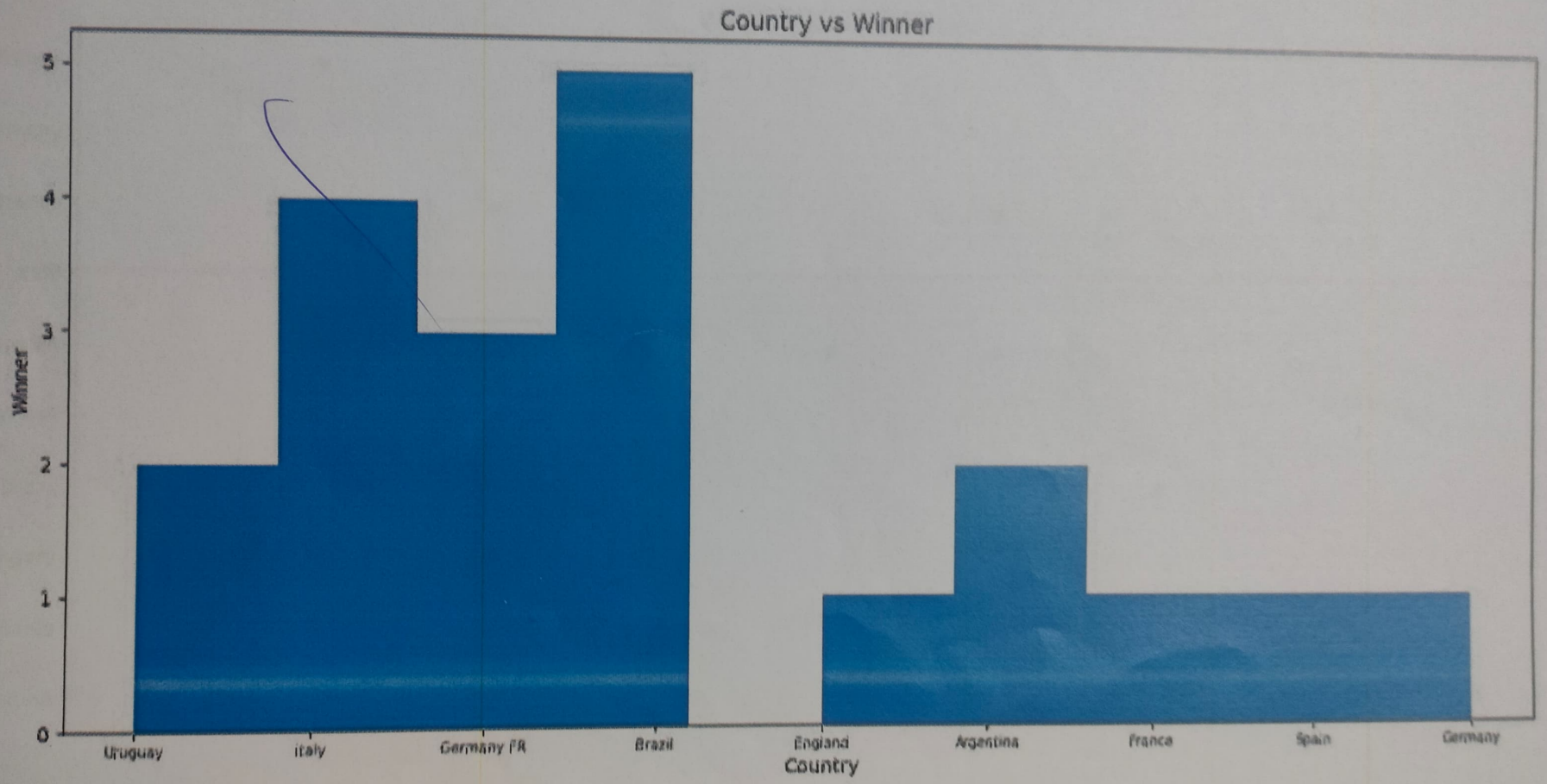
This a plot of country on horizontal axis and goals scored on vertical axis. This plot is done by using plot function.



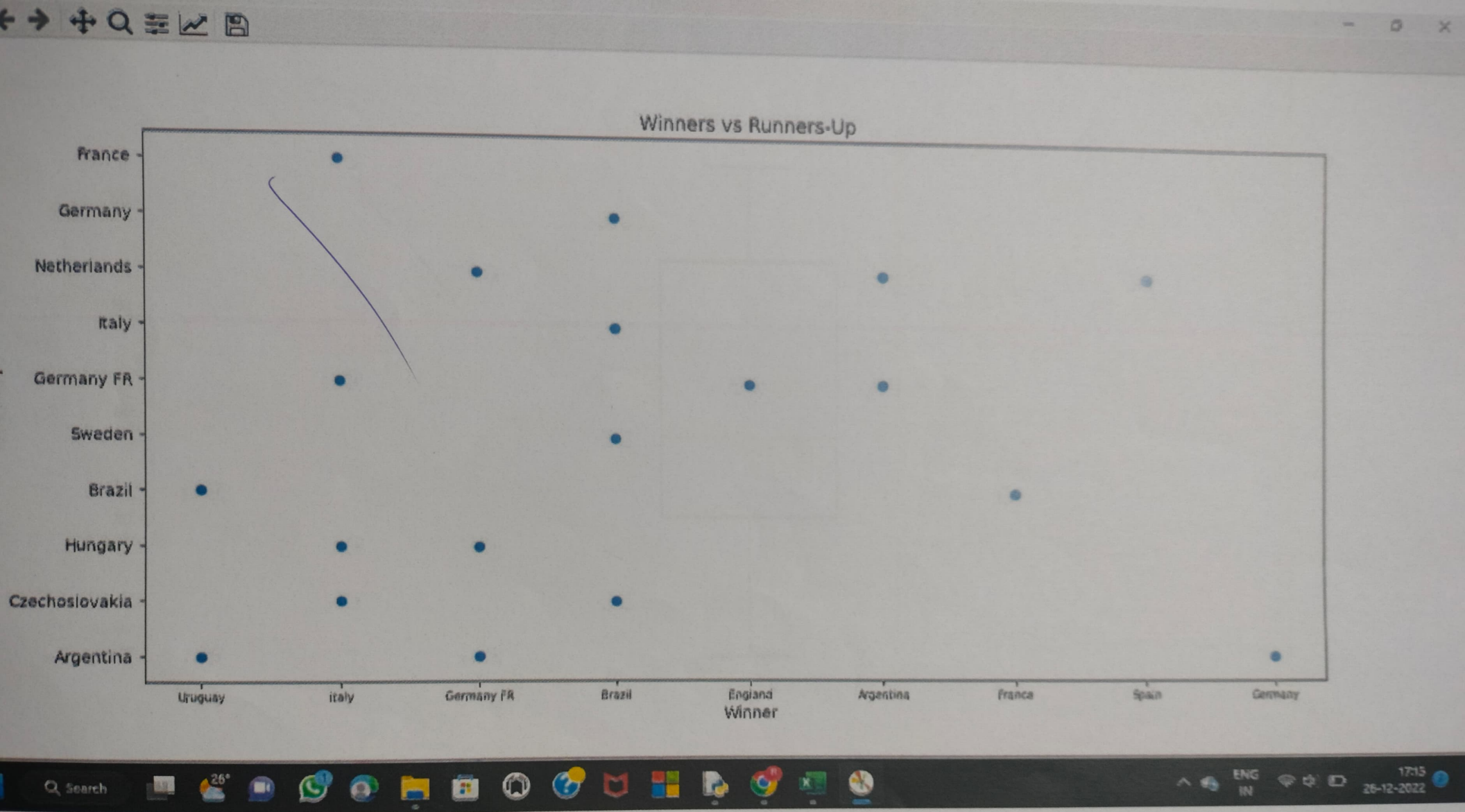
The bar graph is plotted with name of country on x axis and matches played on y-axis Hatch on the bars is done by using hatch function. Labelling is done using xlabel and y label.



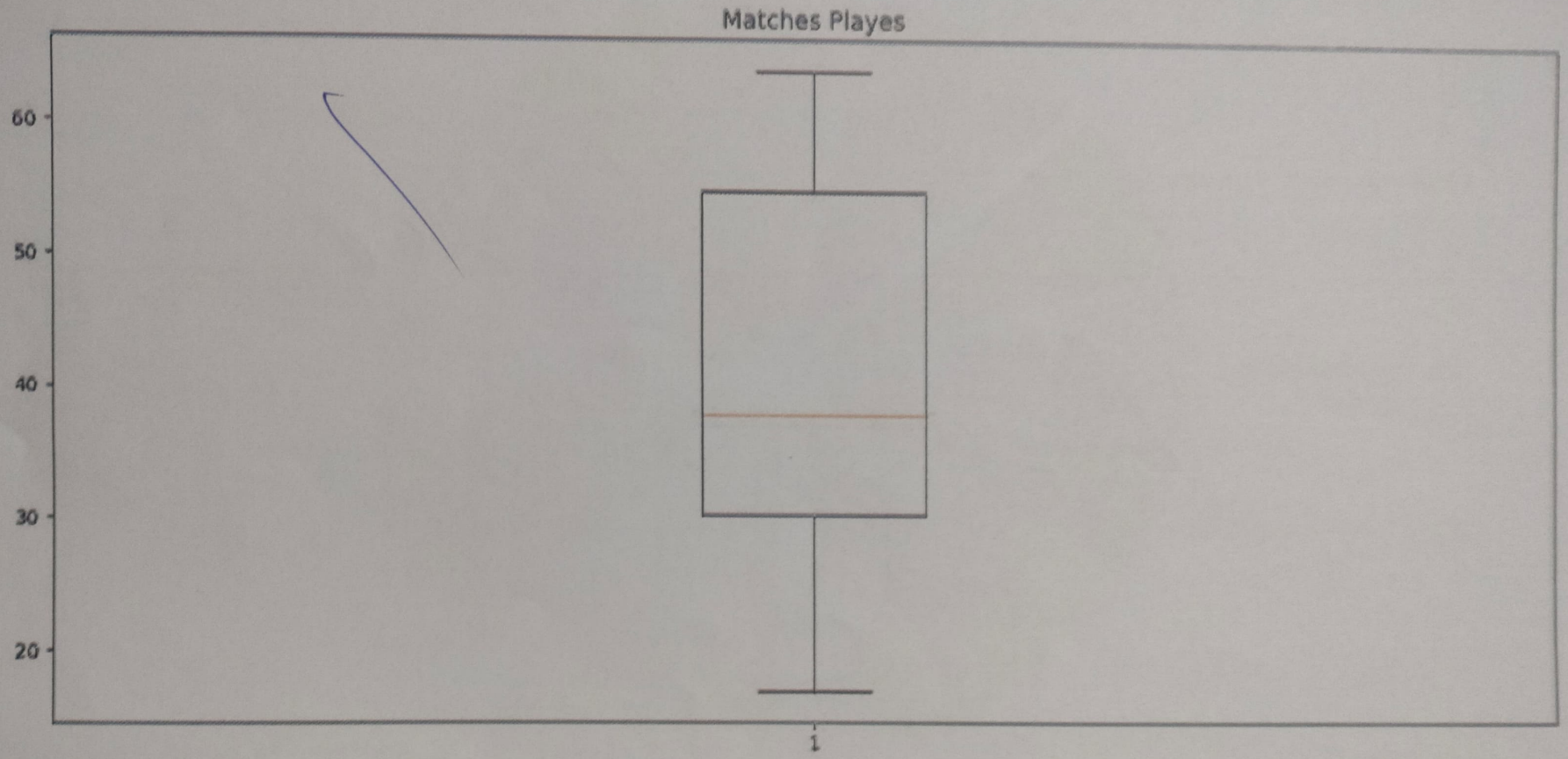
the graph is a pie chart depicting the attendence of each country in world cup. Different colours are shown by default. The labelling is done by labels function



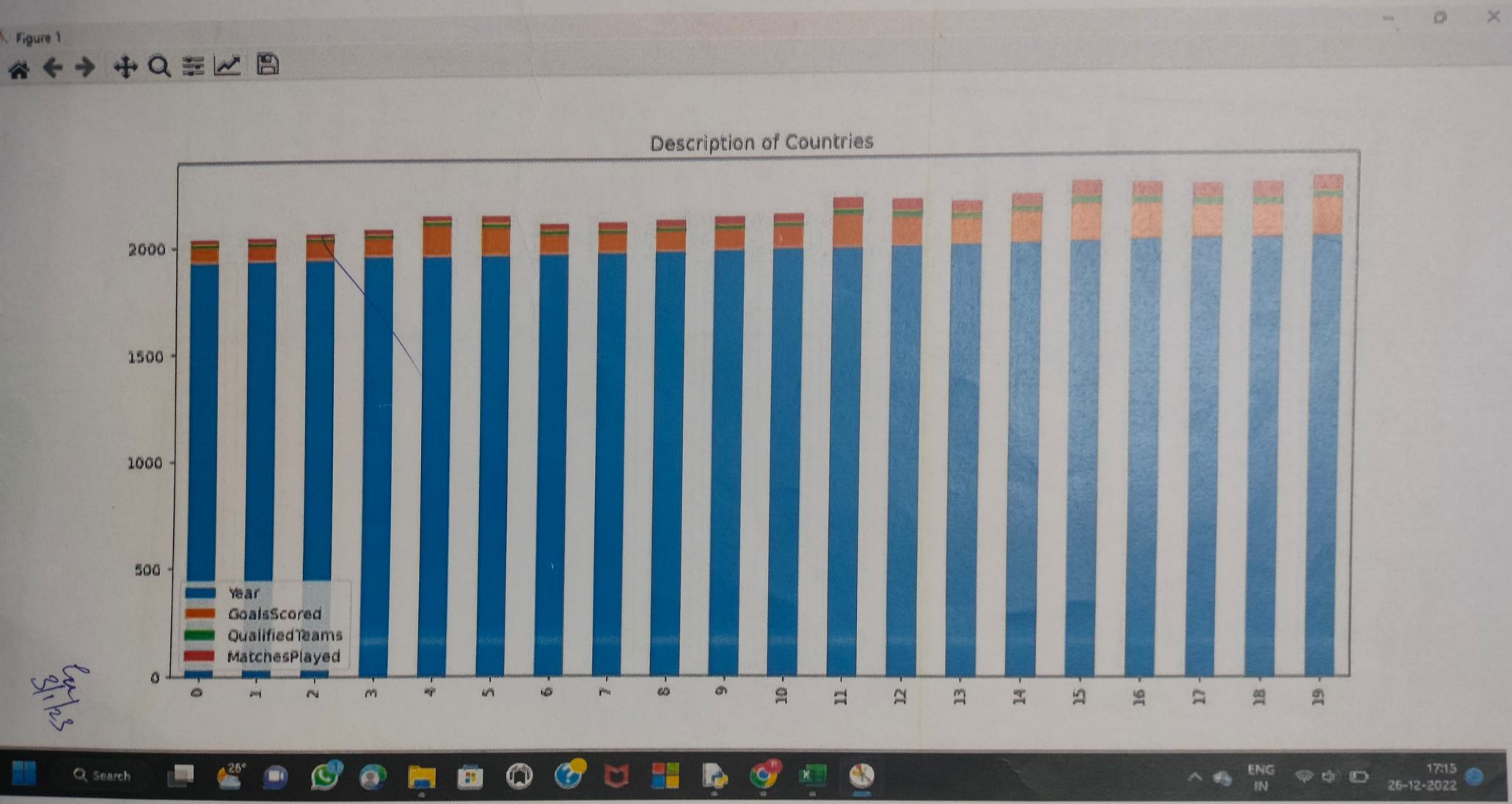
The graph is a histogram with name of country on horizontal axis and number of times a country won in world cup. Title is written using title function



This is a scatter plot showing different countries depicting winners vs runner ups. In this both axes have categorical data



This graph is box plot depicting the number of matches played. The average value is depicted by the orange line



This graph is Stacked Function in terms of bar graph. Stacked function plots data one above the other. The legend gives the information according to colour

AIM:

Demonstration of Influential Statistical sampling.

PROCEDURE:

Influential statistics is a branch of statistics that make use of various analytical tools to draw inferences about the population data from sample data. In this program, we import scipy which consists of all scientific operations. We found out the mean, variance and standard deviation using formula

$$\text{mean} = \frac{\text{Sum of elements of data}}{\text{no of elements of data}}$$

$$\text{Var} \quad \text{Stand Variance} = \frac{\Sigma (x_i - \bar{x})^2}{n-1}$$

$$\text{standard deviation} = \sqrt{\text{variance}}$$

We used scipy functions like mean(), std() and var() to find mean, standard deviation and variance respectively.

PROGRAM:

```
from scipy import stats
from scipy import mean
x = [1, 3, 5, 7, 11, 13, 15, 17]
print ("Data :", x)
s = sum(x)
n = len(x)
#Mean using formula
mean1 = s/n
print ("Mean without using scipy :", mean1)
print ("Mean using scipy :", mean(x))
# using scipy function
```

```
# calculating variance
var=0
for j in a:
    var = var + (j-mean)**2    # Σ(ai-x̄)^2
var = var/(n-1)
print("Standardard deviation using scipy:", stats.tstd(x))
# using std function
print("Standard deviation without using scipy:", var**0.5)
# using var function from scipy.
print("Variance using scipy:", stats.tvar(x))
print("Variance without using scipy:", var)
```

OUTPUT:

```
Data : [1, 3, 5, 7, 11, 13, 15, 17]
Mean without using scipy : 9.0
Mean using scipy : 9.0
Standard deviation using scipy : 5.855400437691199
Standard deviation without using scipy : 5.855400437691199
Variance using scipy : 34.285714285714285
Variance using without using scipy : 34.285714285714285
```

AIM:

To implement one sample ttest.

PROCEDURE:

The one sample t-test is a statistical hypothesis test used to determine whether an unknown population mean is different from specific values. We take a data sample. First we find mean of sample. Calculate the s value using

$$s = \sqrt{\frac{\sum(x_i - \bar{x})^2}{n-1}}$$

$\bar{x}$ = mean of sample

n = size of sample.

Then find degrees of freedom = n-1. Using that we calculate the t critical value. Then calculate p value:

If p value is less than alpha then accept null hypothesis. ~~OR accept null~~ reject null hypothesis. Where alpha is an assumed value.

PROGRAM:

```
from scipy import stats
z = [10,20, 30, 40, 50, 60]
s = sum(z)
n = len(z)
mean = s/n    #Calculating mean of sample.
b = 10
sd = 0
#Calculating s value.
for i in z:
    sd = sd + (i-mean)**2
s = (sd/(n-1))**(0.5)
#Calculating p value
p = (mean-b)/(s/(n**0.5))
print("Without using scipy")
```

```
print("p : ", p)
alpha = 2.3
print("alpha :", alpha)
if p<alpha:
    print ("Accept Null Hypothesis")
else:
    print ("Reject Null Hypothesis")
Print ("Using Scipy")
#Using Scipy function ttest_1samp()
tstat1, p1 = stats.ttest_1samp(z,10)
print ("p :", tstat1)
print ("alpha :", alpha)
if tstat1 < alpha:
    print ("Accept Null Hypothesis")
else:
    print ("Reject null Hypothesis")
```

OUTPUT:

Without using Scipy

P : 3.27326835 3539885

alpha : 2.3

Reject Null Hypothesis

Using Scipy

P : 3.27326835 3539885

alpha : 2.3

Reject Null Hypothesis

AIM:

Demonstration of 2-sample t-test

PROCEDURE:

The 2 sample t test is a method used to test whether the unknown population means of two groups are equal or not. Here we take 2 samples and identify Null hypothesis. Then calculate mean of 2 samples. Then calculate the standard deviations of two samples using

$$s = \sqrt{\frac{\sum (x - \bar{x})^2}{n-1}}$$

$\bar{x}$ be the mean

n be number of elements in sample.

t can be calculates using formula

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}}$$

s_1, s_2 be Standard deviations of 2 samples.

$\bar{x}_1, \bar{x}_2$ be means of 2 samples

n_1, N_2 be no. of elements samples.

If t is less than t critical then Null hypothesis is accept or else reject null hypothesis. where t critical is an assumed value. In this program it is calculated using formula and also by scipy function stats.ttest_ind (s_1, s_2).

PROGRAM:

```
from scipy import stats
a = [2, 4, 6, 8, 10, 12, 14, 16]
b = [3, 6, 9, 12, 15, 18, 21, 24]
print("Without using scipy")
na = len(a)
nb = len(b)
meana = sum(a)/na
meanb = sum(b)/nb
#function for standard deviation
def standard_deviation(s, mean):
        x = 0
        for j in s:
```

```
        x=x+(j-mean)**2
    sd = (x/(len(s)-1))**0.5
    return sd
sa =standard_deviation(a,meana)
sb=standard_deviation(b,meanb)
df =(sa**2/na + sb**2/nb)**2 /((sa**2/(na-1)) +(sb**2/
                    nb)**2/(nb-1))
tstat =(meana-meanb)/((sa**2/na) + (sb**2/nb))**0.5
print("tstat:", tstat)
print("Degree of freedom: ", df)
tcritic =1
if tstat< tcritic:
        print("Accept Null Hypothesis")
else:
    print("Reject Null Hypothesis")
print("Using scipy")
tstat1, p = stats.ttest_ind(a,b, equal_var = False)
print("tstat :", tstat)
print("p: ", p)
alpha = 0.1
if p > alpha:
        print("Accept Null Hypothesis")
else:
    print("Reject Null Hypothesis").
```

OUTPUT:

Without using scipy

tstat = -1.4411533842457842

Degree of freedom : 12.195876288659795

Accept Null Hypothesis

Using scipy

tstat : -1.4411533842457842.

p = 0.17472061882778517

Accept Null Hypothesis

AIM:

Demonstration of paired t-test

PROCEDURE:

Paired t-test is a procedure used to determine whether the mean difference between two set of observations is zero. The two set of observations are separated by time. Two set of samples are taken. The difference and square of difference of two samples is calculated. The degree of freedom is calculated as n-1. Using degrees of freedom t critical value is found. t value is found by using the formulala.

$$t = \frac{\Sigma d}{\sqrt{\frac{n(\Sigma d^2) - (\Sigma d)^2}{n-1}}}$$

d = difference per paired value.

n = number of elements in sample.

If t is less than t critical, Null Hypothesis is accepted or it is rejected. Using formula this can be perform. We can also find by using scipy function stats.ttest_rel(s_1, s_2). if P is less than alpha, reject Null hypothesis or accept null hypothesis.

PROGRAM:

```
from scipy import stats
a = [20, 27, 19, 40, 29, 53, 41]
b = [25, 24, 27, 42, 33, 50, 44]
n = len(a)
print("without using scipy").
D1 = [a[i] - b[i] for i in range(n)]
D2 = [(a[i] - b[i])**2 for i in range(n)]
S1 = sum(D1)
```

```
S2 = sum(D2)
den = ((n*S2 - S1**2) // (n-1)**0.5
t_cntic = 1.77
tstat = S1/den
print("tstat :", tstat)
if tstat < t_cntic :
    print("Accept Null Hypothesis")
else:
    print("Reject Null Hypothesis")
print("Using scipy")
tstat1, p1 = stats.ttest_rel(a,b)
print("tstat :",tstat1)
print("p :", p1)
if p1<0.05:
    print("Reject Null Hypothesis")
else:
    print("Accept Null Hypothesis")
```

OUTPUT:

without using scipy
tstat : -1.4855627054164149
Accept Null Hypothesis
Using scipy
tstat : -1.4855627054164149
p = 0.18793956652624283
Accept Null Hypothesis

AIM:

To perform Wilcoxon Rank Sum Test

PROCEDURE:

The wilcoxon rank-sum test is a non parametric test that uses ranks of sample data from two independent populations. It is used to test the null hypothesis that the two independent samples come from population having same distribution. In this program two samples are taken and ranks are assigned to the samples according to the elements in sample.

W_1 = Sum of ranks of sample 1

W_2 = Sum of ranks of sample 2

$U_1 = W_1 - n_1(n_1+1)/2$

$U_2 = W_2 - n_2(n_2+1)/2$

U = minimum (U_1, U_2).

If $U <$ critical value, accept null hypothesis. By using inbuilt function which is imported from scipy.stats i.e ranksum(sample1, sample2). If $p < 0.05$ then reject Null Hypothesis else accept it.

PROGRAM:

```
import numpy as np
from scipy.stats import ranksums
a = [22, 33, 44, 55, 66, 77, 88]
b = [10, 20, 30, 40, 50, 60, 70, 80]
n1=len(a)
n2=len(b)
c = a+b
c = sorted(c)
w1, w2 =0,0
for i in a:
    w1 = w1+c.index(i)+1
```

```
for j in b:
    w2 = w2 + c.index(j)+1
u1 = w1 - n1 * (n1+1)/2
u2 = w2 - n2 * (n2+1)/2
u = min(u1, u2)
print("without using function")
print("tstat value :", u)
tcritic = 25
if u>tcritic:
    print("Reject Null Hypothesis")
else:
    print("Accept Null Hypothesis").
fstat, p = ranksums(a, b)
print("Using function")
print("p value:", p)
if p<0.05:
    print("Reject Null Hypothesis")
else:
    print("Accept Null Hypothesis").
```

OUTPUT:

```
without using function
tstat value : 21.0
Accept Null Hypothesis
Using function
pvalue : 0.41788694317506
Accept Null Hypothesis
```

Aim:- Demonstration of anova test

Procedure:-

—Anova test is a statistical test used to determine if there is a statistically significant difference between two or more categorical groups by testing for difference of means using variance. In this program Anova is demonstrated by taking 2 samples of different sizes.

$SS_{within} = \Sigma(x_{1j} - \bar{x}_1)^2 + \Sigma(x_{2j} - \bar{x}_2)^2 \ldots \Sigma(x_{ik} - \bar{x}_k)^2$

$= \Sigma(x_{ij} - \bar{x}_j)^2$

$M_{swithin} = \Sigma(x_{ij} - \bar{x}_j)^2 / (N-k)$

$SS_{within} = n_1(\bar{x}_1 - \bar{x}_G)^2 + n_2(\bar{x}_2 - \bar{x}_G)^2 + n_3(\bar{x}_3 - \bar{x}_G)^2 + \ldots n_k(\bar{x}_k - \bar{x}_G)^2$

$M_{sbetween} = \dfrac{n_1(\bar{x}_1 - \bar{x}_G)^2 + n_2(\bar{x}_2 - \bar{x}_G)^2 + n_3(\bar{x}_3 - \bar{x}_G)^2 \ldots n_k(\bar{x}_k - \bar{x}_G)^2}{k-1}$

$f_{stat} = MSSb / MSSw$

Program:-

```
import scipy.
from scipy.stats import f_oneway.
import numpy as np.

group1 = [85, 86, 88, 75, 78, 94, 98, 79, 71, 80]
group2 = [91, 92, 93, 85, 87, 84, 82, 88, 95, 96]
```

Roll No.

CBIT

Output :-

means of two samples are: 83.4 89.3

F_onewayResult (statistic=3.6922804949911 6, pvalue=0.0706432)

Accept Null Hypothesis

MANUAL

Fstat = 3.692280494991505

Accept NH.

AIM:

Time Series Forecasting with ARIMA Model.

PROCEDURE:

Time series analysis comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of data. It is used to predict future values based on previously observed values. In this program we took a csv file. ARIMA is present in module statsmodels. In this program ARIMA function is used ARIMA(p,d,q) p is lag order, d is degree of differencing, q is order of moving average. We even plot a graph for csv file between Sales and months, we also printed the description the model is prepared on training data by calling fit() function. Predictions can be made by calling predict() function and specifying time or times to be predicted.

PROGRAM:

```
from statsmodels.tsa.arima.model import ARIMA
import pandas as pd
from matplotlib import pyplot
Series = pd.read_csv('shampoo_sales.csv', header=0, parse_dates=[0],
                     index_col=0, squeeze=True)
model = ARIMA(Series, order=(5,1,0))
model_fit=model.fit()
print(model_fit.summary())
residuals = pd.DataFrame(model_fit.resid)
residuals.plot()
pyplot.ylabel("Sales")
pyplot.show()
```

Date

```
residuals.plot(kind='kde')
pyplot.show()
print(residuals.describe())
```

OUTPUT:

SARIMAX RESULTS

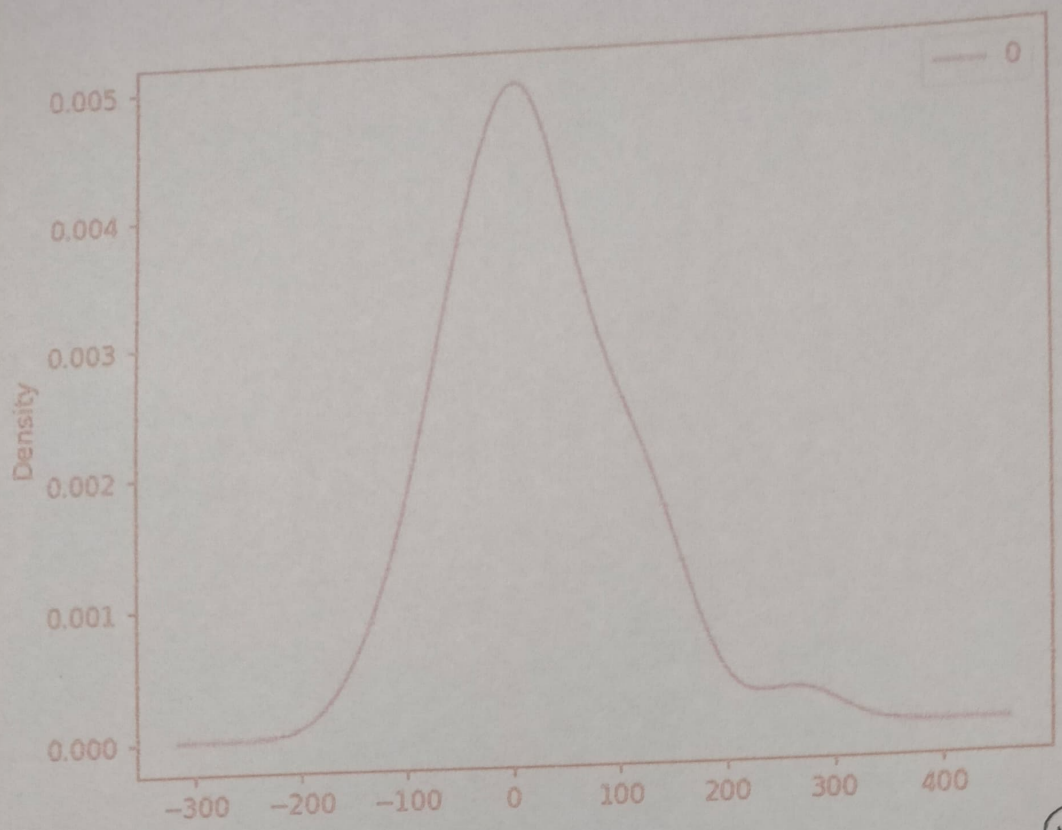
Dep. Variable:	Sales	No. observations	36
Model:	ARIMA(5,1,0)	log likelihood	-198.485
Date:	Mon, 6 Feb 2023	AIC	408.969
Time:	21:14:11	BIC	418.301
Sample:	0 - 36	HQIC	412.191
Covariance Type:	opg		

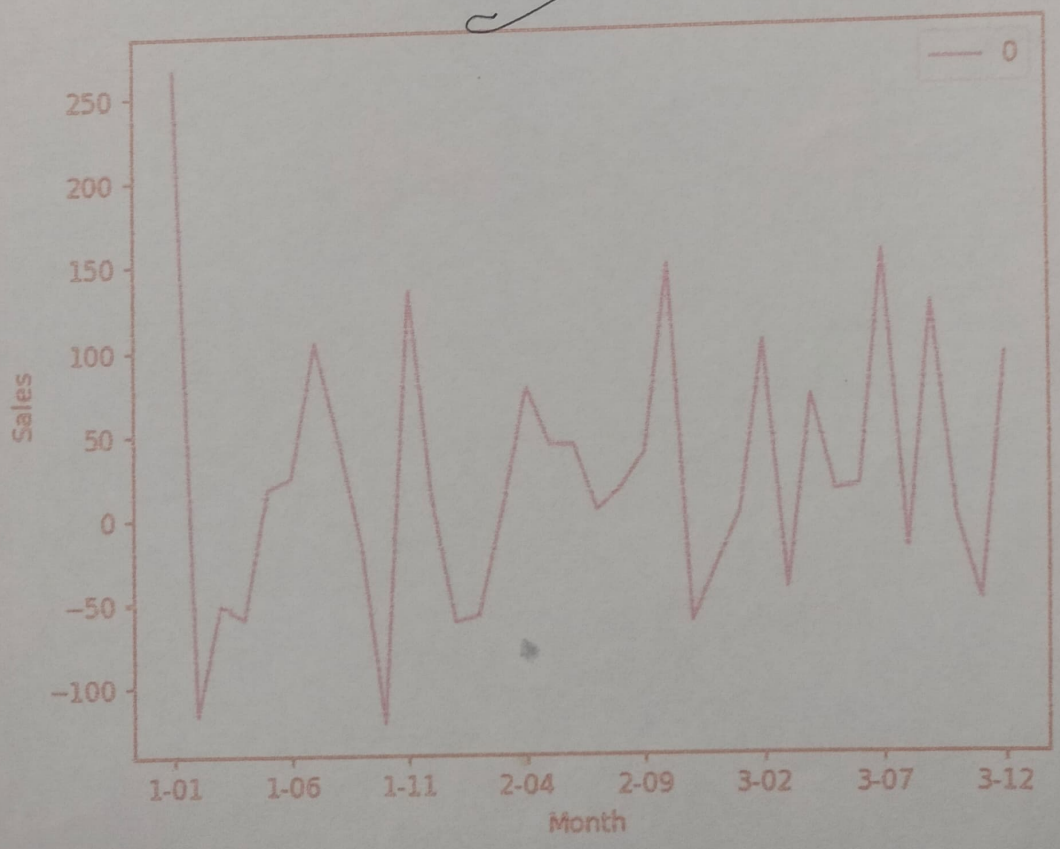
	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.9014	0.247	-3.647	0.000	-1.386	-0.417
ar.L2	-0.2284	0.268	-0.851	0.395	-0.754	0.298
ar.L3	0.0747	0.291	0.256	0.798	-0.497	0.646
ar.L4	0.2519	0.340	0.742	0.458	-0.414	0.918
ar.L5	0.3344	0.210	1.593	0.111	-0.077	0.746
sigma2	4728.9608	1316.021	3.593	0.000	2149.607	7308.315

Ljung-Box L1(Q):	0.61	Jarque-Bera (JB):	0.96
Prob(Q):	0.44	Prob(JB):	0.62
Heteroskedasticity	1.07	Skew:	0.28
Prob(H) (two-sided):	0.90	Kurtosis:	2.41

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

count	36.000000
mean	21.936145
std	80.774430
min	-122.292030
25%	-35.040859
50%	13.147219
75%	68.848286
max	266.000000


0
0.005
0.004
0.003
0.002
0.001
0.000
Density
−300
−200
−100
0
100
200
300
400


0
250
200
150
100
50
0
−50
−100
Sales
1-01
1-06
1-11
2-04
2-09
3-02
3-07
3-12
Month