**Practical – 1**

**AIM:** Study of numpy library of Python. Write a program to demonstrate use of various functions.

**Numpy:**

* In Python we have lists that serve the purpose of arrays, but they are slow to process.
* NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.
* Arrays are very frequently used in data science, where speed and resources are very important.
* **Installation of NumPy:**
* If you have Python and PIP already installed on a system, it will install the numpy.
* Command:
* **pip install numpy**
* **Import NumPy:**
* Once NumPy is installed, import it in your applications by adding the import keyword.
* Create an alias with the as keyword while importing:
* Command:
* **import numpy as np**

**Programs:**

1. **Create an Array:**

* We can create a NumPy ndarray object by using the array() function.
* We can create multi-dimensional array by using numpy.

**CODE:**

##create an 1D array

a = np.array([1,2,3,4,5])

a

**OUTPUT:**

****

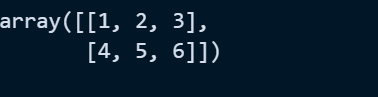
**CODE:**

# create an 2D array of multidimensional

b = np.array([[1,2,3],[4,5,6]])

b

**OUTPUT:**

****

1. **Shape of an Array:**

* The shape of an array is the number of elements in each dimension.
* NumPy arrays have an attribute called shape that returns a tuple with each index having the number of corresponding elements.

**CODE:**

# return the row and columns of the array

arr3.shape

**OUTPUT:**

****

1. **Size of an Array:**

* Numpy arrays have attribute called size that returns a total no. of elements are available in the array.

**CODE:**

# return the total no. of elements in the array

a.size

**OUTPUT:**

****

**CODE:**

x = np.size(a)

x

**OUTPUT:**

****

1. **Dimension of an Array:**

* NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

**CODE:**

arr3.ndim

**OUTPUT:**

****

1. **Sorting of numpy array:**

* The NumPy ndarray object has a function called sort(), that will sort a specified array.
* You can also sort arrays of strings, or any other data type.

**CODE:**

# create an 1D array

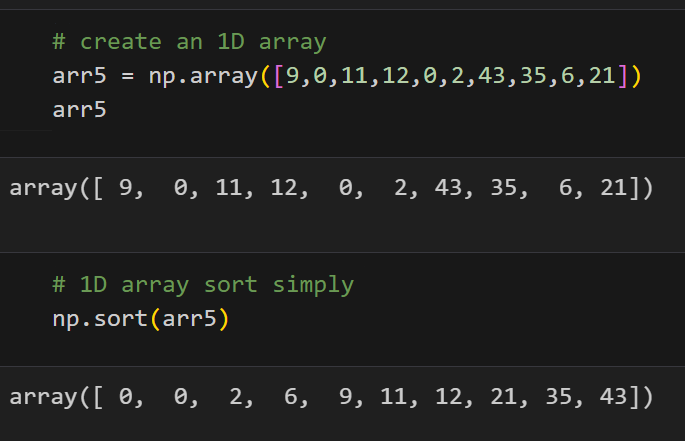
arr5 = np.array([9,0,11,12,0,2,43,35,6,21])

arr5

# 1D array sort simply

np.sort(arr5)

**OUTPUT:**

****

1. **Zeros function of numpy:**

* The numpy.zeros() function returns a new array of given shape and type, with zeros.

**CODE:**

x = np.zeros(2)

x

**OUTPUT:**

****

1. **Ones function of numpy:**

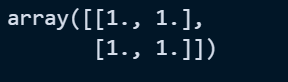
* The numpy.ones() function returns a new array of given shape and type, with ones.

**CODE:**

y = np.ones((2,3))

y

**OUTPUT:**

****

1. **Arange function of numpy:**

* The arange([start,] stop[, step,][, dtype]) : Returns an array with evenly spaced elements as per the interval. The interval mentioned is half-opened i.e. [Start, Stop)

**CODE:**

z = np.arange(10)

print(z)

p = np.arange(2,8)

print(p)

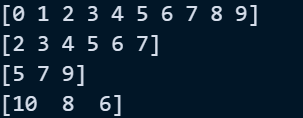
q = np.arange(5,10,2)

print(q)

r = np.arange(10,5,-2)

print(r)

**OUTPUT:**

****

1. **Concatenate of numpy arrays:**

* For concatenate the dimension or shape should be same for the both the arrays.

**CODE:**

np.concatenate((np.array([1,2,3]),a),axis=0)

**OUTPUT:**

****

1. **Random class of numpy:**

* The random is a module present in the NumPy library. This module contains the functions which are used for generating random numbers.
* This module contains some simple random data generation methods, some permutation and distribution functions, and random generator functions.

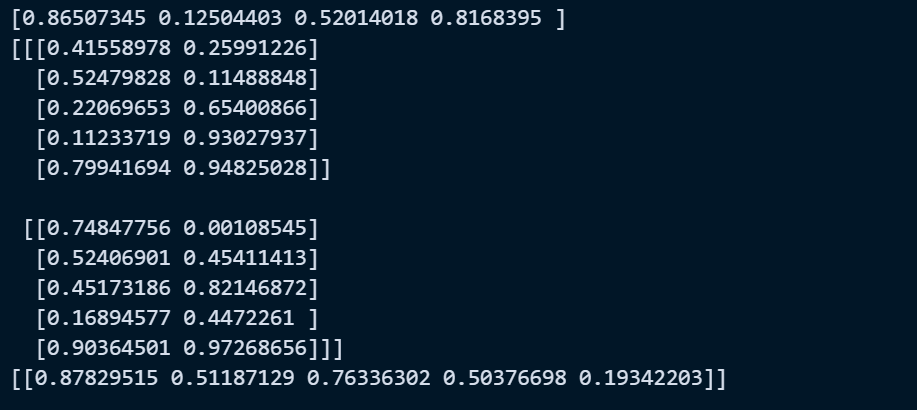
**CODE:**

print(np.random.rand(4))

print(np.random.rand(2,5,2))

print(np.random.rand(1,5))

**OUTPUT:**

****

1. **Linespace function of numpy:**

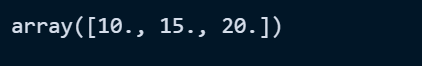
* The NumPy.linspace() function returns an array of evenly spaced values within the specified interval [start, stop].
* It is similar to NumPy.arange() function but instead of a step, it uses a sample number.

**CODE:**

c = np.linspace(10,20,num=3)

c

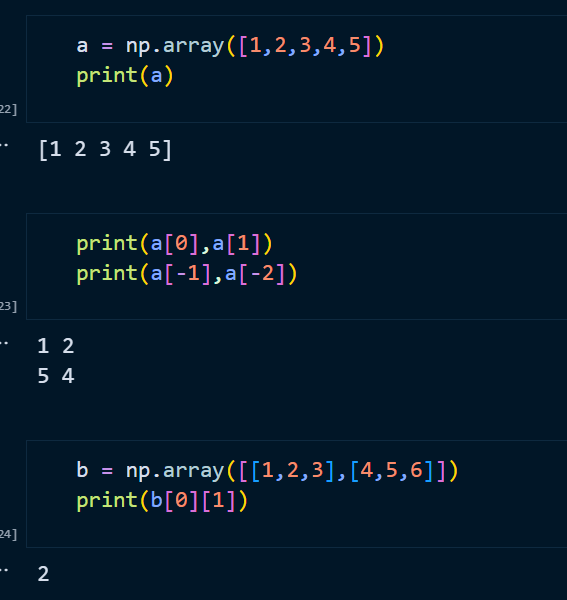
**OUTPUT:**

****

1. **Indexing in array:**

* Array indexing is the same as accessing an array element.
* You can access an array element by referring to its index number.
* The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

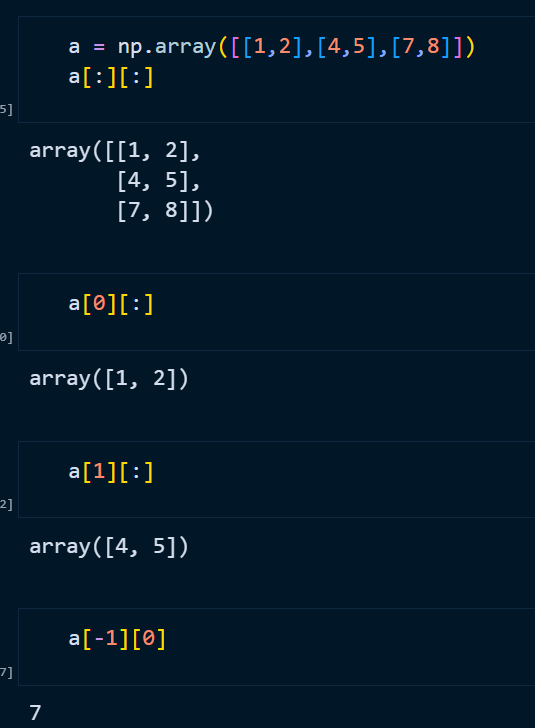
**CODE + OUTPUT:**

****

1. **Slicing in array:**

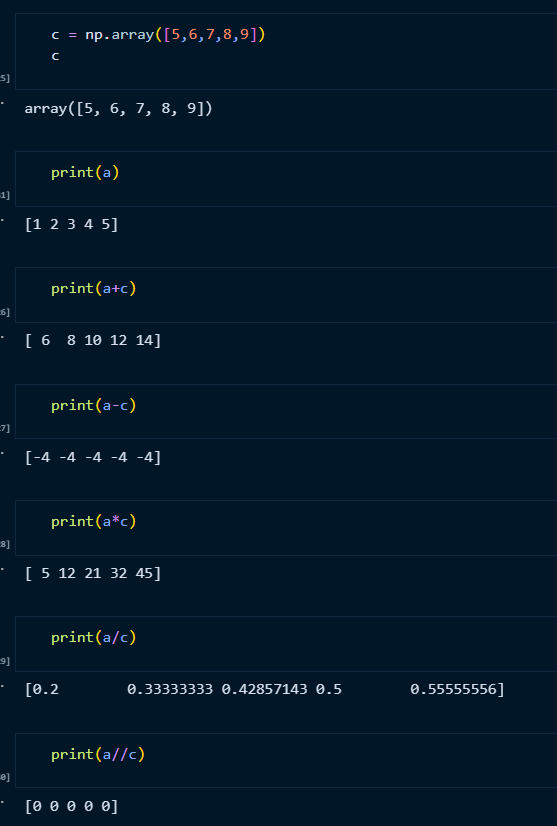
* Slicing in python means taking elements from one given index to another given index.
* We pass slice instead of index like this: [start:end].
* We can also define the step, like this: [start:end:step].
* If we don't pass start its considered 0
* If we don't pass end its considered length of array in that dimension
* If we don't pass step its considered 1

**CODE + OUTPUT:**

****

1. **Arithmetic Operation on array:**

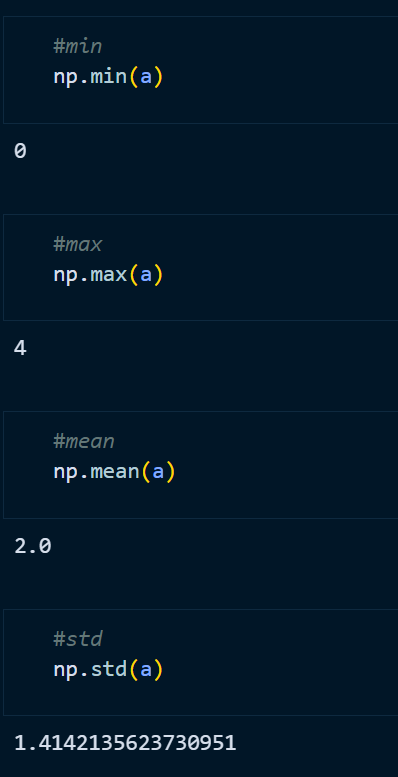
**CODE + OUTPUT:**

****

1. **Max,Min,Mean and Std function of array:**

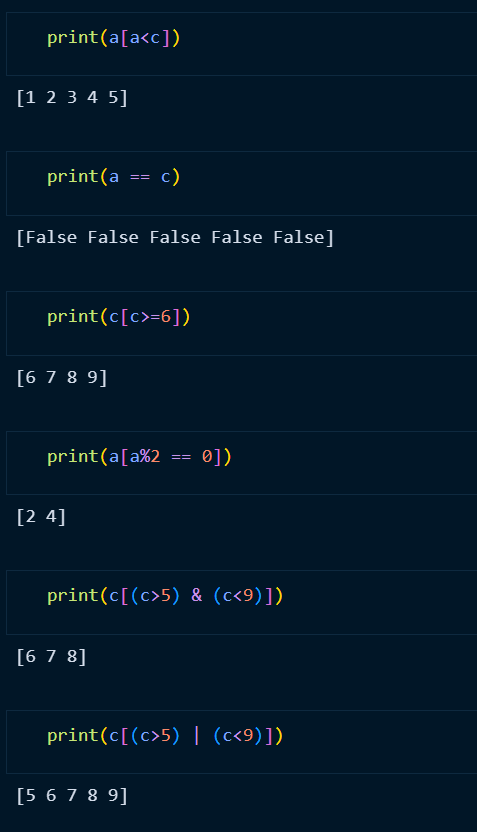
* NumPy has both a package-level function and an ndarray method named max(). They work in the same way, though the package function np.max() requires the target array name as its first parameter.

**CODE + OUTPUT:**

****

1. **Comparison of array:**

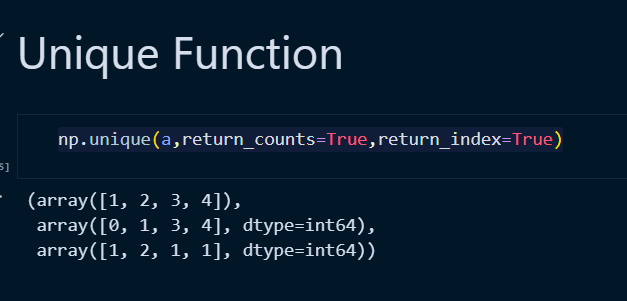
**CODE + OUTPUT:**

****

1. **Unique in array:**

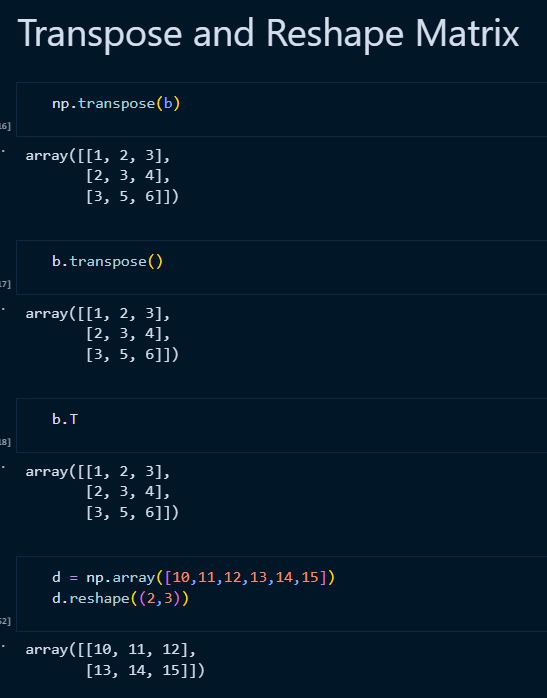
* The transpose of a matrix is found by interchanging its rows into columns or columns into rows.

**CODE + OUTPUT:**

****

1. **Transpose of matrix:**

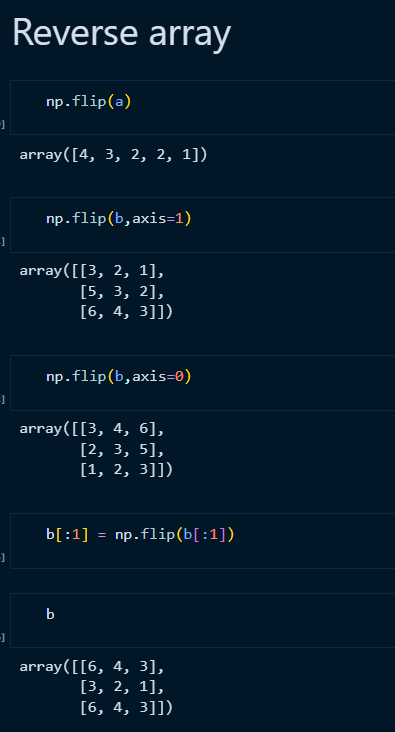
**CODE + OUTPUT:**

****

1. **Flip method of array:**

* The numpy.flip() function reverses the order of array elements along the specified axis, preserving the shape of the array.

**CODE + OUTPUT:**

****

**Practical – 2**

**AIM:** Study of matplotlib library of Python. Write a program to demonstrate use of various functions.

**Matplotlib:**

* Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
* Matplotlib was created by John D. Hunter.
* Matplotlib is open source and we can use it freely.
* **Installation of Matplotlib:**
* If you have Python and PIP already installed on a system, it will install the Matplotlib.
* Command:
* **pip install matplotlib**
* **Import Matplotlib:**
* Once Matplotlib is installed, import it in your applications by adding the import keyword.
* Create an alias with the as keyword while importing:
* Command:
* **import matplotlib**
* **PyPlot:**
* Import Matplotlib:
* Once Matplotlib is installed, import it in your applications by adding the import keyword.
* Create an alias with the as keyword while importing:
* Command:
* **import matplotlib.pyplot**

**Programs:**

1. **Line Graph:**

**CODE:**

import matplotlib.pyplot as plt

x = [1,2,3]

y = [4,5,6]

x1 = [4.5,8,9]

y1 = [10,5.1,12]

plt.plot(x,y,"--",x1,y1,"--")

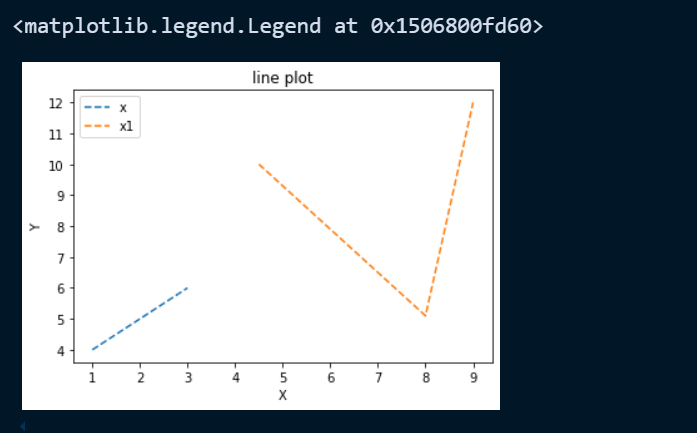
plt.title("line plot")

plt.xlabel("X")

plt.ylabel("Y")

plt.legend(["x","x1"])

**OUTPUT:**

****

1. **Bars:**

**CODE:**

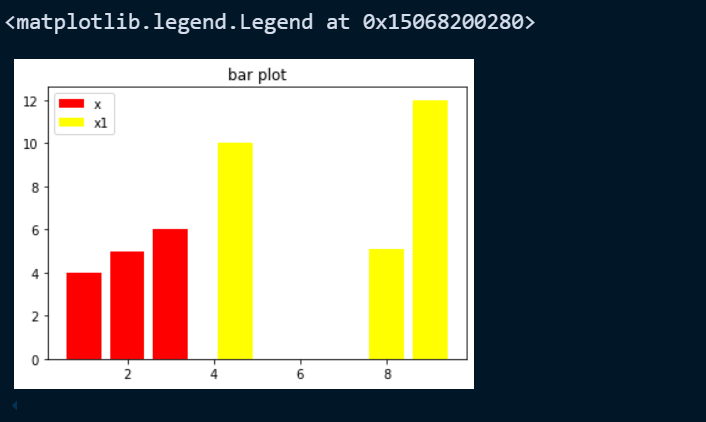
plt.bar(x,y,width=.8,color="red")

plt.bar(x1,y1,width=0.8,color='yellow')

plt.title("bar plot")

plt.legend(["x","x1"])

**OUTPUT:**

****

1. **Histogram:**

* A histogram is a graph showing frequency distributions.
* It is a graph showing the number of observations within each given interval.

**CODE:**

# histogram char in python

data = np.random.randn(1000)

plt.hist(data, bins=30, color='skyblue', edgecolor='black')

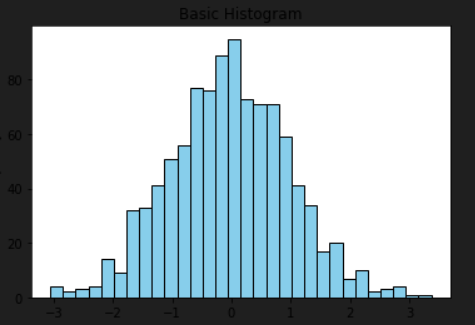
plt.xlabel('Values')

plt.ylabel('Frequency')

plt.title('Basic Histogram')

plt.show()

**OUTPUT:**

****

1. **Scatter Plot:**

* With Pyplot, you can use the scatter() function to draw a scatter plot.
* The scatter() function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis:

**CODE:**

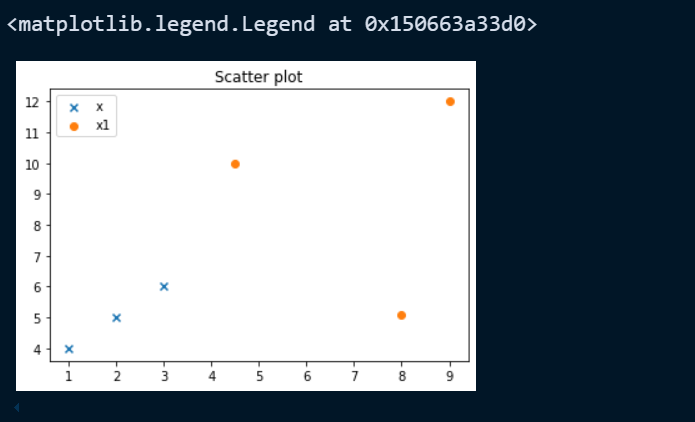
plt.scatter(x,y,marker="x")

plt.scatter(x1,y1)

plt.title("Scatter plot")

plt.legend(["x","x1"])

**OUTPUT:**

****

1. **Area Chart:**

**CODE:**

days = [1,2,3,4]

study = [8,5,9,10]

play = [7,4,6,2]

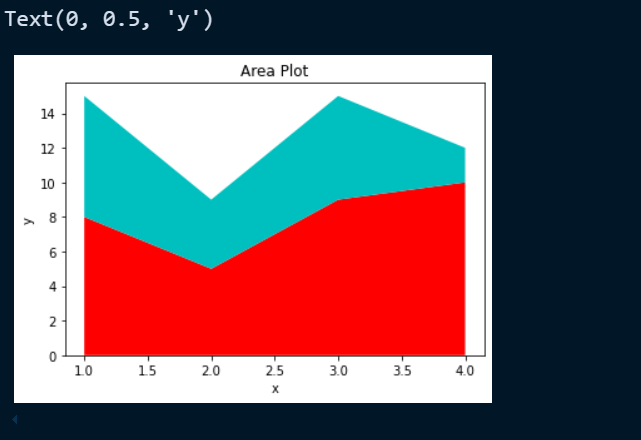
plt.stackplot(days,study,play,colors=['r','c'])

plt.title("Area Plot")

plt.xlabel("x")

plt.ylabel("y")

**OUTPUT:**

****

1. **Pie Chart:**

* With Pyplot, you can use the pie() function to draw pie charts:

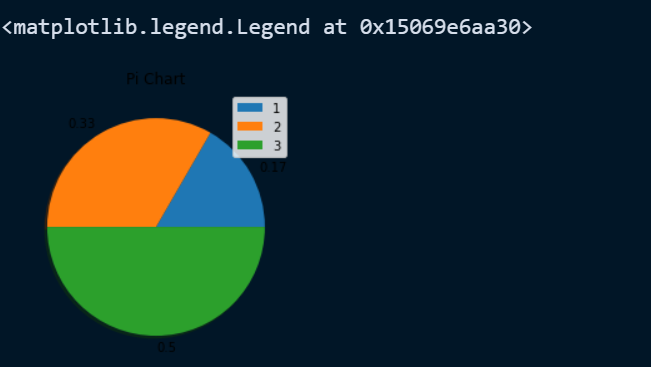
**CODE:**

plt.pie(x,shadow=True,labels=[round(1/sum(x),2),round(2/sum(x),2),round(3/sum(x),2)])

plt.title("Pi Chart")

plt.legend([1,2,3])

**OUTPUT:**

****

**Practical – 3**

**AIM:** Study of Pandas library of Python. Write a program to demonstrate its use.

**Pandas:**

* Pandas is a Python library used for working with data sets.
* It has functions for analyzing, cleaning, exploring, and manipulating data.
* Pandas allows us to analyze big data and make conclusions based on statistical theories.
* Pandas can clean messy data sets, and make them readable and relevant.
* **Installation of Pandas:**
* If you have Python and PIP already installed on a system, it will install the pandas.
* Command:
* **pip install pandas**
* **Import Pandas:**
* Once Pandas is installed, import it in your applications by adding the import keyword.
* Create an alias with the as keyword while importing:
* Command:
* **import pandas as pd**

**Programs:**

1. **Read CSV file using pandas:**

* A simple way to store big data sets is to use CSV files (comma separated files).
* CSV files contains plain text and is a well know format that can be read by everyone including Pandas.

**CODE:**

# read csv file in python

import pandas as pd

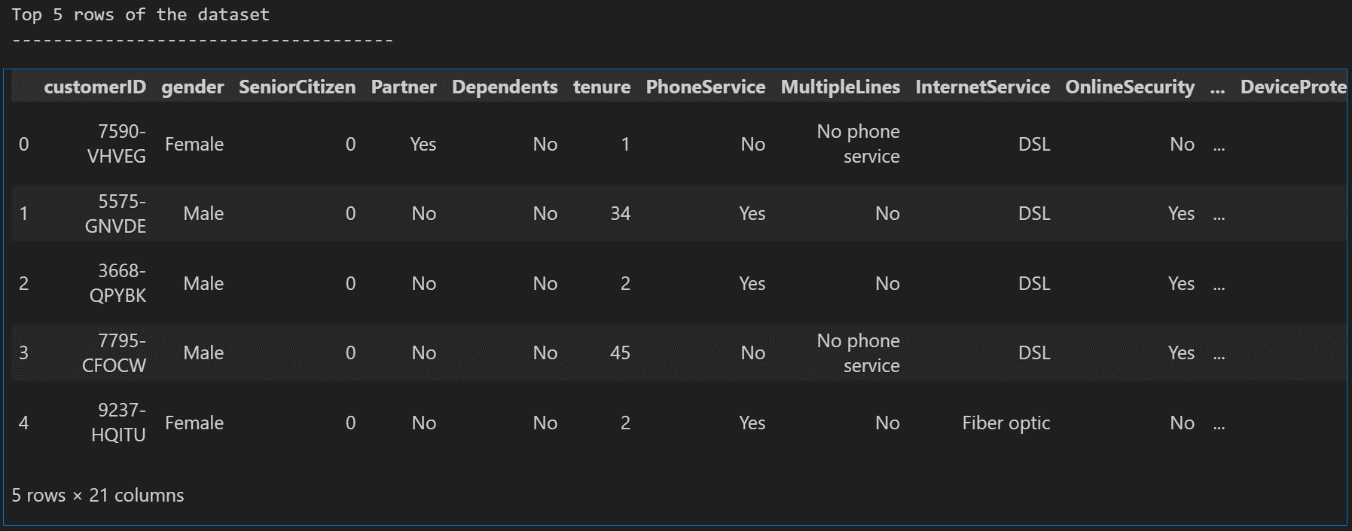
df = pd.read\_csv('./LAB2CSV.csv')

# read top 5 samples

print("Top 5 rows of the dataset\n-------------------------------------")

df.head()

**OUTPUT:**

****

* Read\_csv function take one argument as **nrows** by which we can read the specific no. of rows(samples) from the dataset(CSV file).

**CODE:**

print("Get DataFrame just of 50 columns\n-------------------------------------")

spe\_df = pd.read\_csv('./LAB2CSV.csv', nrows=50)

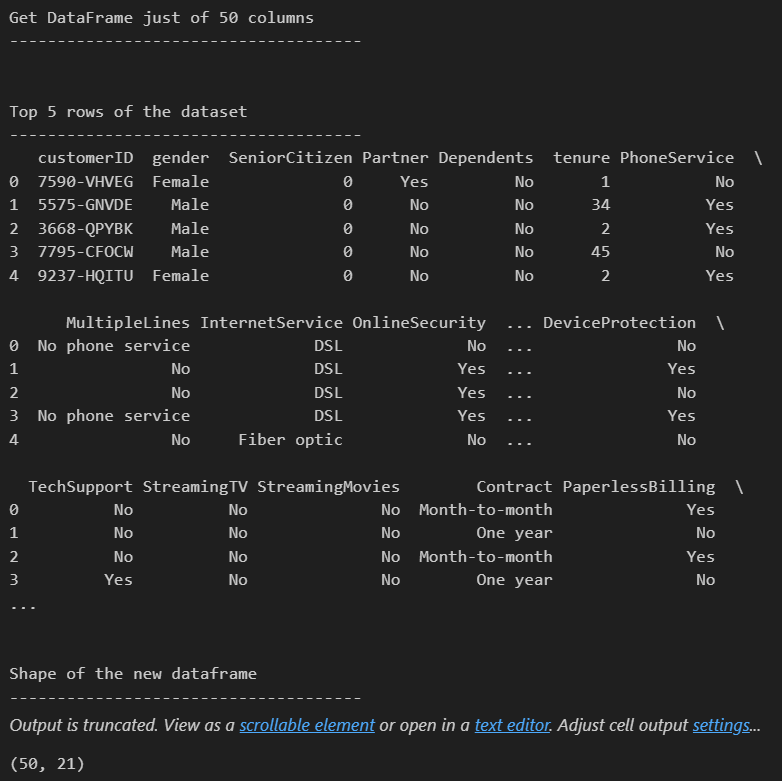
print("\n\nTop 5 rows of the dataset\n-------------------------------------")

print(spe\_df.head())

print("\n\nShape of the new dataframe\n-------------------------------------")

spe\_df.shape

**OUTPUT:**

****

* Read\_csv function take one argument as **usecols** by which we can just read the specific columns from the dataset(CSV file).

**CODE:**

print("Get DataFrame just of 50 columns & 5 columns\n-------------------------------------")

spe\_df = pd.read\_csv('./LAB2CSV.csv', nrows=50, usecols=['customerID', 'gender', 'Partner', 'InternetService', 'Contract'])

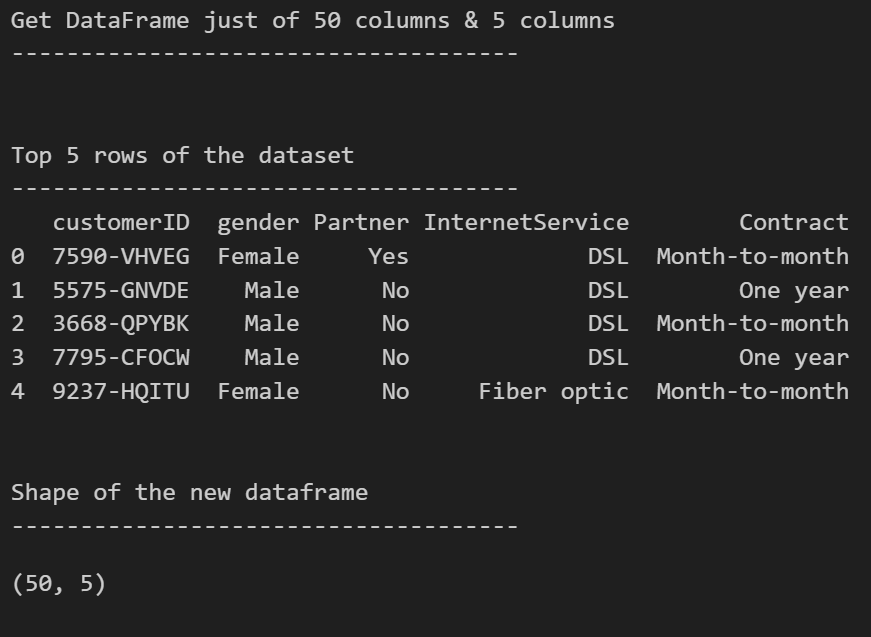
print("\n\nTop 5 rows of the dataset\n-------------------------------------")

print(spe\_df.head())

print("\n\nShape of the new dataframe\n-------------------------------------")

spe\_df.shape

**OUTPUT:**

****

1. **Shape of dataset(dataframe):**

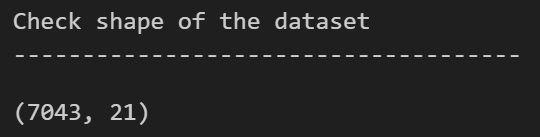
* The shape property returns a tuple containing the shape of the DataFrame.
* The shape is the number of rows and columns of the DataFrame

**CODE:**

print("Check shape of the dataset\n-------------------------------------")

df.shape

**OUTPUT:**

****

1. **Get Columns of dataset(dataframe):**

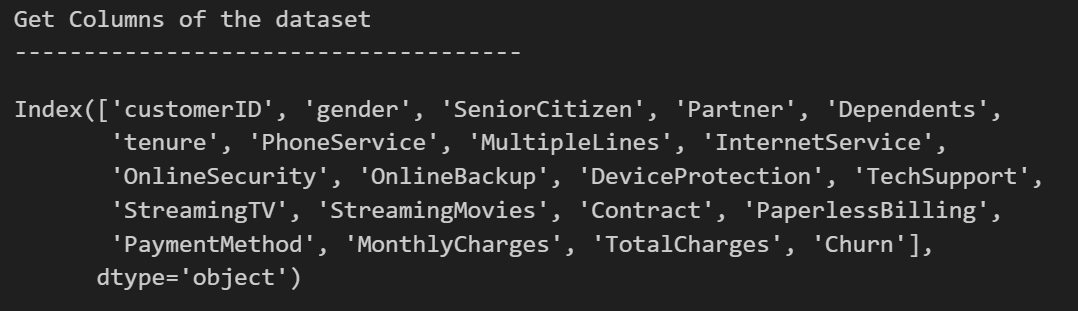
* This attribute does not require any parameters to be passed. When called on a data frame using the syntax **DataFrame.columns**, it returns the names of the columns present in that data frame.

**CODE:**

print("Get Columns of the dataset\n-------------------------------------")

df.columns

**OUTPUT:**

****

1. **Head function on dataset(dataframe):**

* The head() method returns a specified number of rows, string from the top.
* The head() method returns the first 5 rows if a number is not specified.

**CODE:**

print("Get DataFrame just of 5 columns")

spe\_df = pd.read\_csv('./LAB2CSV.csv', usecols=['customerID', 'gender', 'Partner', 'InternetService', 'Contract'])

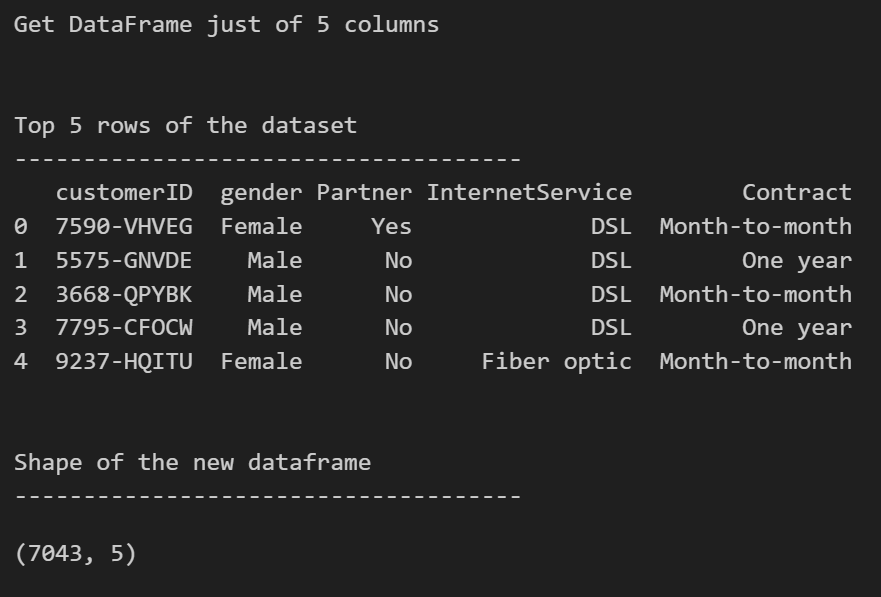
print("\n\nTop 5 rows of the dataset\n-------------------------------------")

print(spe\_df.head())

print("\n\nShape of the new dataframe\n-------------------------------------")

spe\_df.shape

**OUTPUT:**

****

1. **Drop function of pandas:**

* The drop() method removes the specified row or column.
* By specifying the column axis (axis='columns'), the drop() method removes the specified column.
* By specifying the row axis (axis='index'), the drop() method removes the specified row.

**CODE:**

print("Get shape of th data frame\n-------------------------------------------")

new\_df = df.drop(['gender', 'Partner'], axis = 1) # axis 1 : for delete column

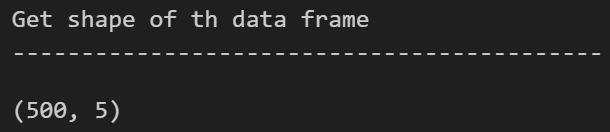
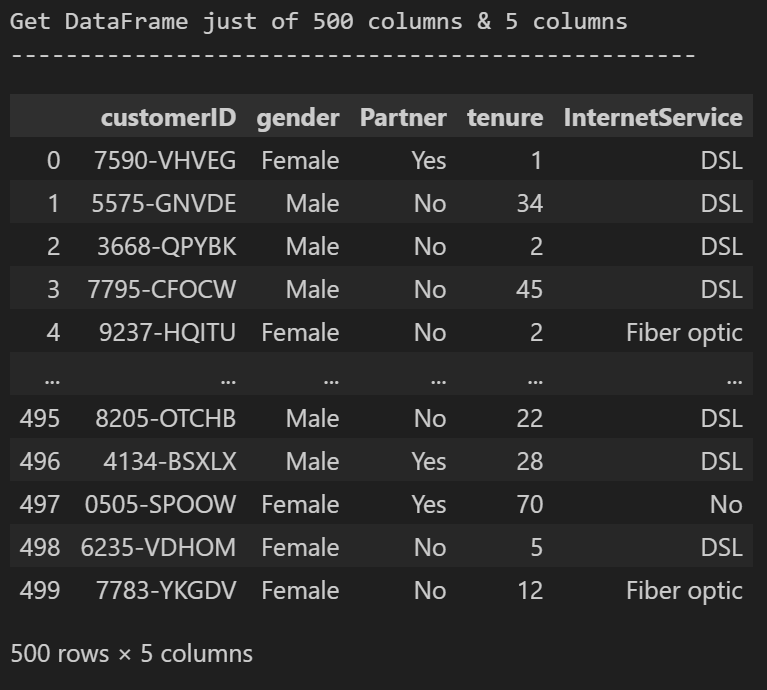
# new\_df = df.drop(['gender', 'Partner'], axis = 0) # axis 0 : for delete rows

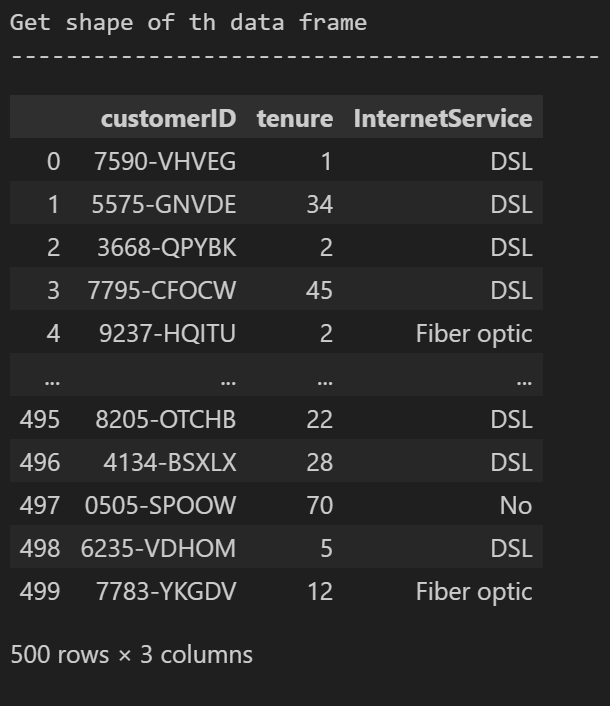
# new\_df = df.drop(Partner', axis = 1) # delete the single column for multiple column can pass list

# new\_df = df.drop(['gender', 'Partner'], axis = 1, inplace = True) # inplace True : means original data frame update by default False

new\_df

**OUTPUT:**

****

****

1. **Sample function of pandas:**

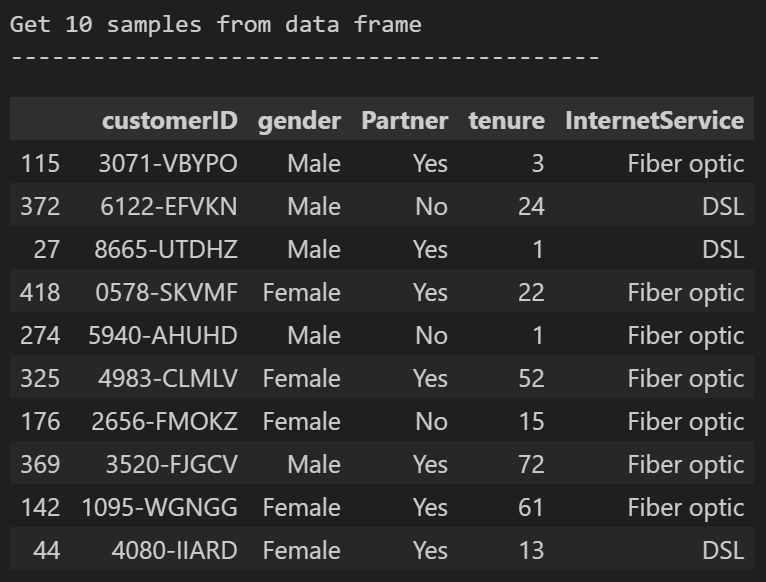
* Pandas sample() is used to generate a sample random row or column from the function caller data frame.

**CODE:**

print("Get 10 samples from data frame\n-------------------------------------------")

df.sample(n = 10)

**OUTPUT:**

****

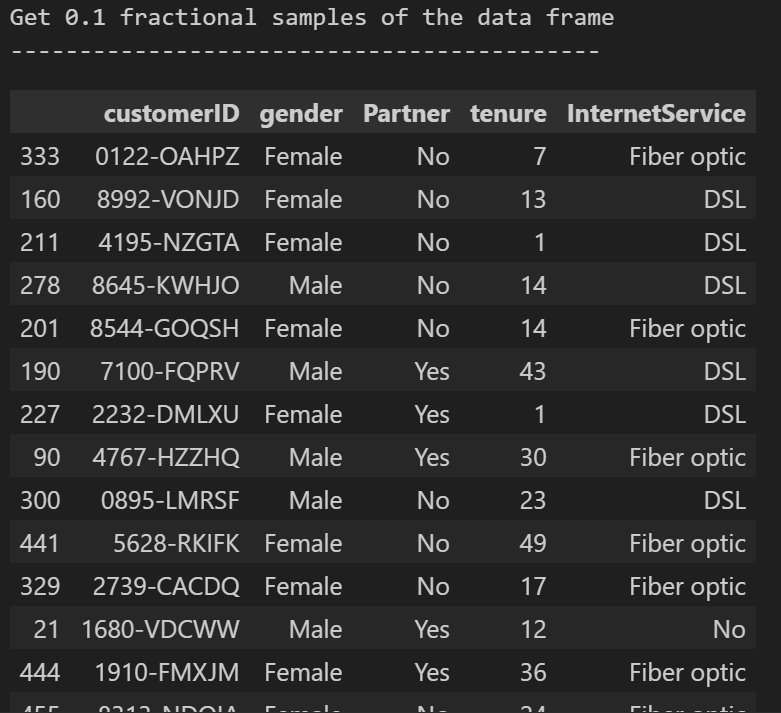
* Here one more argument can be pass as frac it is optional argument in .sample().
* It return fraction of rows, like 0.5 for 50% of the rows

**CODE:**

print("Get 0.1 fractional samples of the data frame\n-------------------------------------------")

df.sample(frac = 0.1)

**OUTPUT:**



1. **Count function of the pandas:**

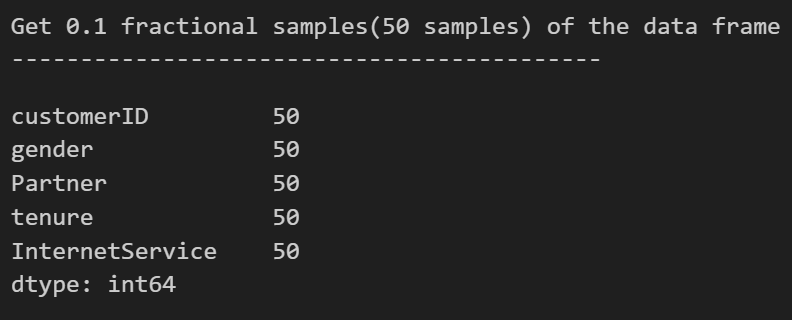
* The count() method counts the number of not empty values for each row, or column if you specify the axis parameter as axis='columns', and returns a Series object with the result for each row (or column).

**CODE:**

print("Get 0.1 fractional samples(50 samples) of the data frame\n-------------------------------------------")

df.sample(frac = 0.1).count()

**OUTPUT:**

****

1. **Isna function of the pandas:**

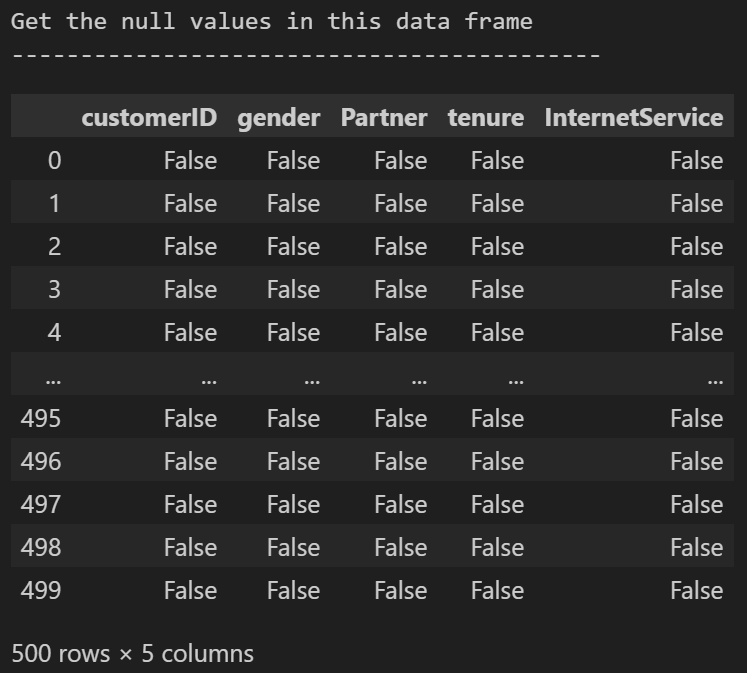
* Pandas dataframe.isna() function is used to detect missing values. It return a boolean same-sized object indicating if the values are NA. NA values, such as None or numpy.NaN, gets mapped to True values.
* Everything else gets mapped to False values. Characters such as empty strings ” or numpy.inf are not considered NA values.

**CODE:**

print("Get the null values in this data frame\n-------------------------------------------")

df.isna()

**OUTPUT:**

****

1. **Sum function of pandas:**

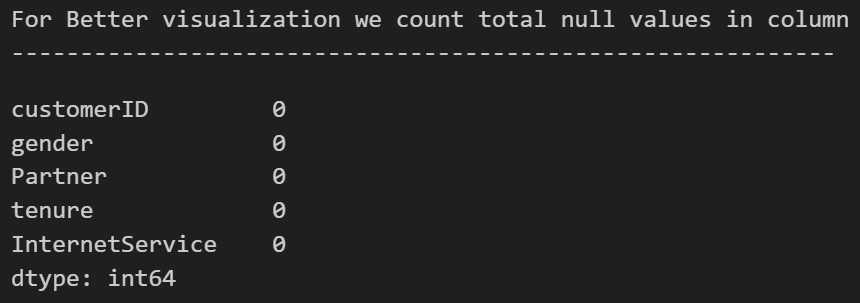
* The sum() method adds all values in each column and returns the sum for each column.
* By specifying the column axis (axis='columns'), the sum() method searches column-wise and returns the sum of each row.

**CODE:**

print("For Better visualization we count total null values in column\n------------------------------------------------------------")

df.isna().sum()

**OUTPUT:**

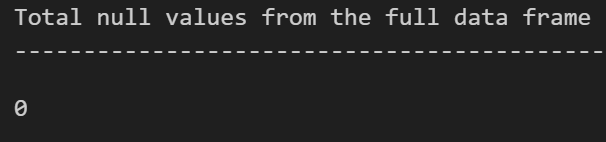
****

**CODE:**

print("Total null values from the full data frame\n------------------------------------------------------------")

df.isna().sum().sum()

**OUTPUT:**

****

1. **Loc[] of pandas:**

* Pandas DataFrame.loc attribute accesses a group of rows and columns by label(s) or a boolean array in the given DataFrame.

**CODE:**

import numpy as np

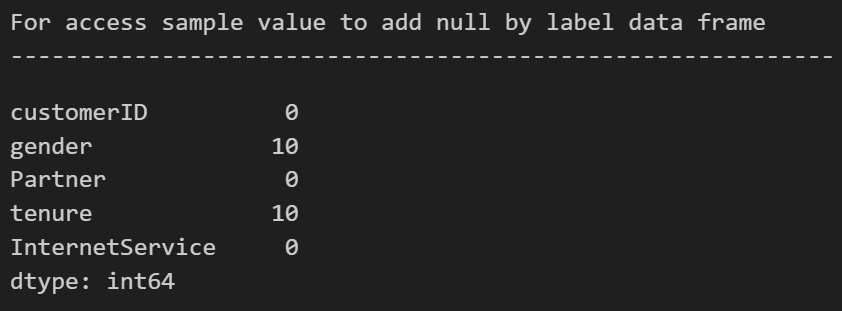
print("For access sample value to add null by label data frame\n------------------------------------------------------------")

missing\_val = np.random.randint(500, size=10)

df.loc[missing\_val, ['gender', 'tenure']] = np.nan

df.isna().sum()

**OUTPUT:**

****

1. **Iloc[] of pandas:**

* The iloc property gets, or sets, the value(s) of the specified indexes.
* Specify both row and column with an index.
* To access more than one row, use double brackets and specify the indexes, separated by commas:
* df.iloc[[0, 2]]
* Specify columns by including their indexes in another list:
* df.iloc[[0, 2], [0, 1]]
* You can also specify a slice of the DataFrame with from and to indexes, separated by a colon:
* df.iloc[0:2]

**CODE:**

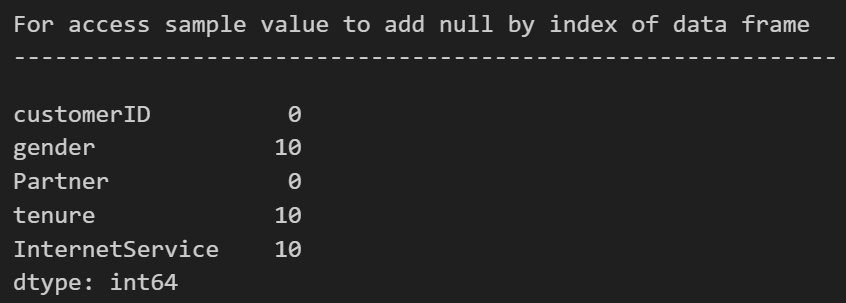
print("For access sample value to add null by index of data frame\n------------------------------------------------------------")

missing\_val = np.random.randint(500, size=10)

df.iloc[missing\_val, -1] = np.nan

df.isna().sum()

**OUTPUT:**

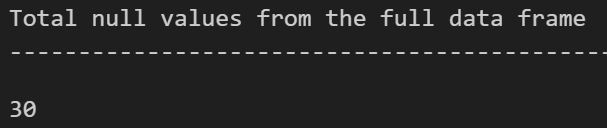
****

**CODE:**

print("Total null values from the full data frame\n------------------------------------------------------------")

df.isna().sum().sum()

**OUTPUT:**

****

1. **Value\_counts of pandas:**

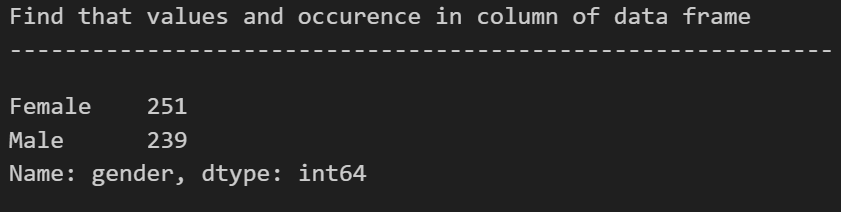
* Pandas Index.value\_counts() function returns object containing counts of unique values. The resulting object will be in descending order so that the first element is the most frequently-occurring element. Excludes NA values by default.

**CODE:**

print("Find that values and occurence in column of data frame\n------------------------------------------------------------")

df['gender'].value\_counts()

**OUTPUT:**

****

1. **Fillna function of pandas:**

* The fillna() method replaces the NULL values with a specified value.
* The fillna() method returns a new DataFrame object unless the inplace parameter is set to True, in that case the fillna() method does the replacing in the original DataFrame instead.

**CODE:**

print("Replace the null values of column with specific value in data frame\n------------------------------------------------------------")

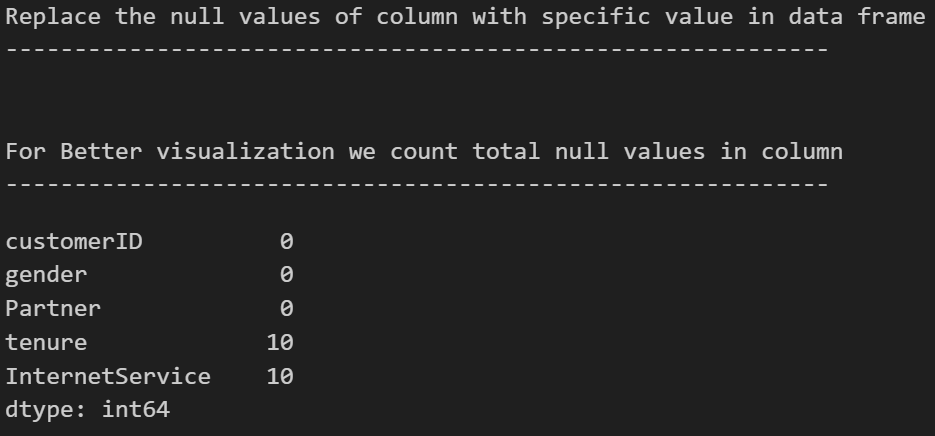
mode = df['gender'].value\_counts()[0]

df['gender'].fillna(mode, inplace = True)

print("\n\nFor Better visualization we count total null values in column\n------------------------------------------------------------")

df.isna().sum()

**OUTPUT:**

****

1. **Mean function of pandas:**

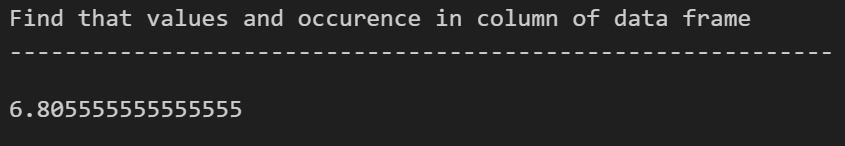
* The mean() method returns a Series with the mean value of each column.
* By specifying the column axis (axis='columns'), the mean() method searches column-wise and returns the mean value for each row.

**CODE:**

print("Find that values and occurence in column of data frame\n------------------------------------------------------------")

df['tenure'].value\_counts().mean()

**OUTPUT:**

****

**CODE:**

print("Replace the null values of column with specific value in data frame\n------------------------------------------------------------")

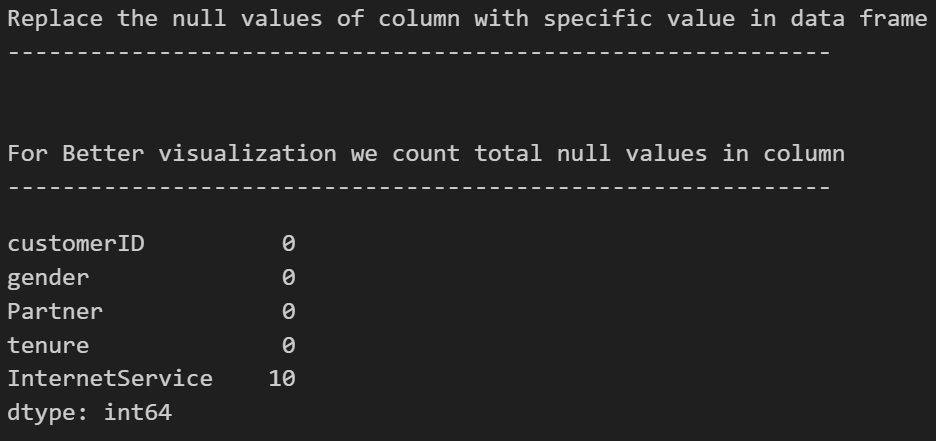
mode = df['tenure'].value\_counts().mean()

df['tenure'].fillna(mode, inplace = True)

print("\n\nFor Better visualization we count total null values in column\n------------------------------------------------------------")

df.isna().sum()

**OUTPUT:**

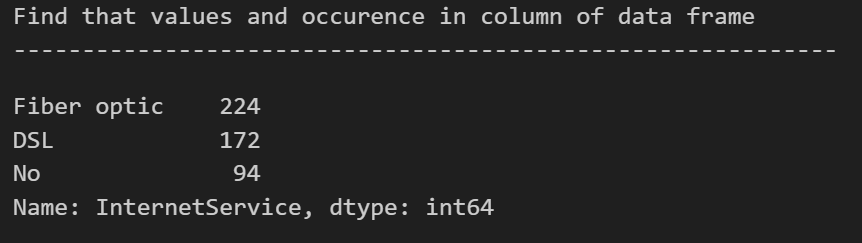
****

**CODE:**

print("Find that values and occurence in column of data frame\n------------------------------------------------------------")

df['InternetService'].value\_counts()

**OUTPUT:**

****

1. **Dropna function of pandas:**

* Pandas is one of the packages that makes importing and analyzing data much easier. Sometimes CSV file has null values, which are later displayed as NaN in Pandas DataFrame. Pandas dropna() method allows the user to analyze and drop Rows/Columns with Null values in different ways.

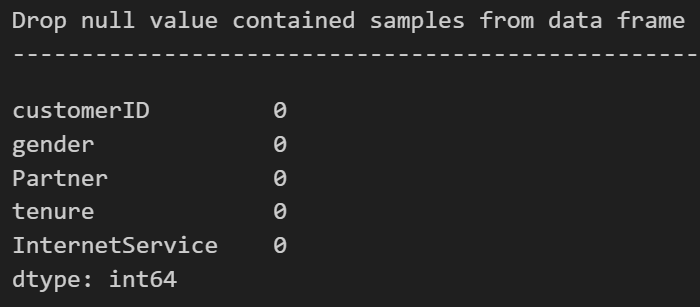
**CODE:**

print("Drop null value contained samples from data frame\n------------------------------------------------------------")

df.dropna(axis = 0, how = 'any', inplace = True)

df.isna().sum()

**OUTPUT:**

****

1. **Condition access in dataframe of pandas:**

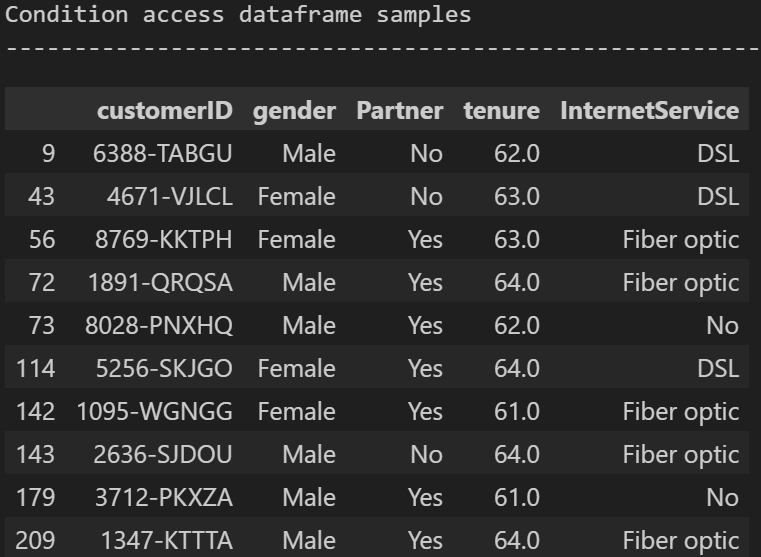
**CODE:**

print("Condition access dataframe samples\n------------------------------------------------------------")

new\_val = df[(df['tenure'] > 60) & (df['tenure'] < 65)]

new\_val

**OUTPUT:**

****

1. **Groupby function of pandas:**

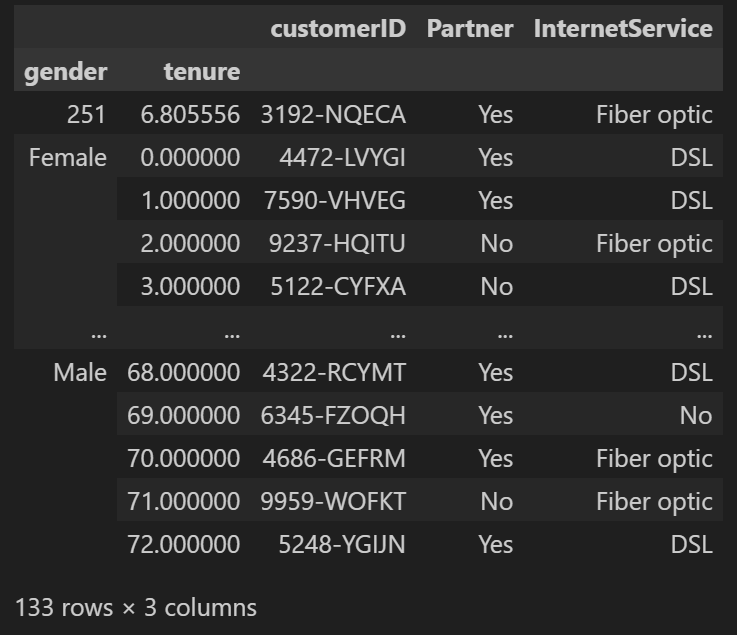
* Pandas dataframe.groupby() function is used to split the data into groups based on some criteria. Pandas objects can be split on any of their axes. The abstract definition of grouping is to provide a mapping of labels to group names.

**CODE:**

new\_val = df.groupby(['gender', 'tenure'])

new\_val.first()

**OUTPUT:**

****

**Practical – 3**

**AIM:** Write a program to demonstrate working of Linear Regression Algorithm without using scikit learn library. Implement the same algorithm using scikit learn and compare the results.

**Linear Regression Algorithm:**

* Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between the dependent variable and one or more independent features by fitting a linear equation to observed data.
* When there is only one independent feature, it is known as Simple Linear Regression, and when there are more than one feature, it is known as Multiple Linear Regression.
* **Import Libraries:**
* Command:
* **import numpy as np**
* **import pandas as pd**
* **import matplotlib.pyplot as plt**

**Programs:**

1. **Without Scikit learn:**

**CODE:**

def estimate\_coe(x, y):

n = np.size(x)

mean\_x = np.mean(x)

mean\_y = np.mean(y)

print("Mean of the X values : ", mean\_x)

print("Mean of the Y values : ", mean\_y)

sum\_xy = 0

sum\_x = 0

for i in range(n):

sum\_xy += ((x[i] - mean\_x) \* (y[i] - mean\_y))

sum\_x += (x[i] - mean\_x)\*\*2

print("\n(x - x\')(y - y\') : ", sum\_xy)

print("(x - x\') : ", sum\_x)

slop = (sum\_xy / sum\_x)

constant = mean\_y - slop \* mean\_x

return (slop, constant)

def plot\_graph(x, y, ans):

plt.scatter(x, y, color="m", marker = 'o', s = 30)

predicted\_y = ans[1] + ans[0]\*x

plt.plot(x, predicted\_y, color='g')

plt.xlabel('X Values')

plt.ylabel('Y Values')

plt.show()

def main():

print("Linear Regression\n---------------------\n\n")

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

print("Input Values\n---------------------")

print("x : ", x)

print("y : ", y)

print("\n\nCorrelation Estimation\n-------------------------")

ans = estimate\_coe(x, y)

print("\nSlop of the input : ", ans[0])

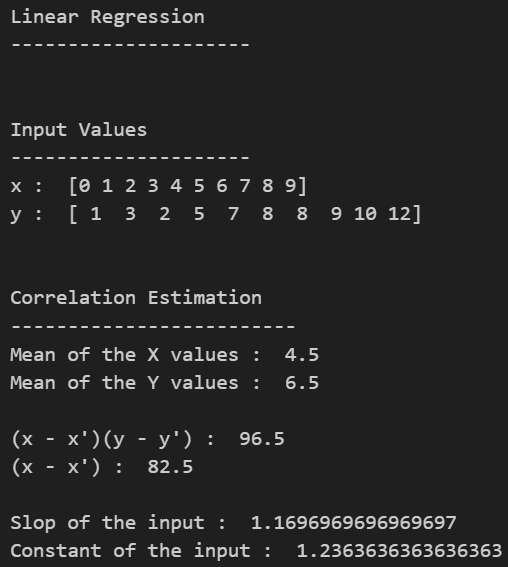
print("Constant of the input : ", ans[1])

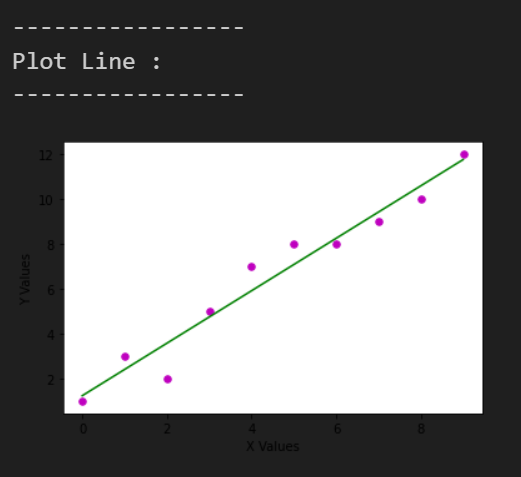
print("\n-----------------\nPlot Line :\n-----------------")

plot\_graph(x, y, ans)

main()

**OUTPUT:**

****



1. **With Scikit learn built in methods:**

**CODE:**

# linear regression with using built-in function of sklearn

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

print("Linear Regression\n---------------------\n\n")

df = pd.read\_csv("./salary\_data.csv")

print("Data From CSV File\n---------------------")

print(df.head(3))

x = df.iloc[:, :-1].values

y = df.iloc[:, 1].values

print("\n\nSplit data into 2 parts(Train Data, Test Data)\n--------------------------------------------")

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=1/3, random\_state=0)

regressor = LinearRegression()

regressor.fit(x\_train, y\_train)

y\_pred = regressor.predict(x\_test)

y\_train\_pred = regressor.predict(x\_train)

plt.scatter(x\_train, y\_train, color="g")

plt.scatter(x\_train, y\_test\_pred, color="red")

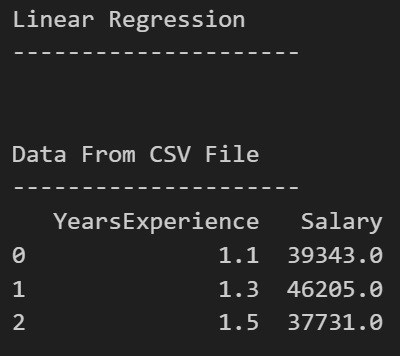
plt.title("Training Dataset")

plt.xlabel('Years Of Experience')

plt.ylabel('Salary(In Rypees)')

plt.show()

**OUTPUT:**

****

