first program

```
def linearSearch(array,n,x):
  #Going through array sequentially
   for i in range(0,n):
       if array[i] == x:
           return i

array = [2,4,0,1,9]
x=eval(input("enter the element to be searched:"))
n=len(array)
result = linearSearch(array,n,x)

if result == -1:
   print("Element not found")
else:
   print("Element found at index:", result)
```

second program

```
import bisect
def insert(list,n):
       bisect.insort(list,n)
       return list
list=[1,2,4]
```

```
n=eval(input("enter the value to be inserted"))
print(insert(list,n))
```

third program

```
class Base:
    def __init__(self):
        self.a=10
        self._b=20
    def display(self):
        print("the values are:")
        print(f"a={self.a} b={self._b}")

class Derived(Base):
    def __init__(self):
        Base.__init__(self)
        self.d=30

    def display(self):
        Base.display(self)
        print(f"d={self.d}")

    def __add__(self,ob):
        return self.a + ob.a + self.d + ob.d
```

```
obj1=Base()
obj2=Derived()
obj3=Derived()

obj2.display()
obj3.display()

print("\n Sum of two objects:", obj2 + obj3)
```

4th

```
import pandas as pd

na_values = ["??","????"]

# Read the CSV file and specify index_col and na_values parameters
cars_data = pd.read_csv("Toyota.csv", index_col=0, na_values = na_values)

# Display the first 5 rows of the DataFrame
print("First 5 rows:")
print(cars_data.head())

# Display the last 3 rows of the DataFrame
```

```
print("\nLast 3 rows:")
print(cars_data.tail(3))

# Display the index of the DataFrame
print("\nIndex:")
print(cars_data.index)

# Display the columns of the DataFrame
print("\nColumns:")
print(cars_data.columns)

# Display the shape of the DataFrame (number of rows and columns)
print("\nShape:")
print(cars_data.shape)



#ii) cleaning the data
#Drop features that are not required
cars_data2=cars_data.copy()

cars_data2=cars_data.drop(['Doors','Weight'],axis='columns')
print("Shape= ",cars_data2.shape)

#Dealing with Missing values
#identifying missing values(NaN -> Not a Number)
sum_na=cars_data2.isna().sum()
print("Sum of null values:\n",sum_na)
#subsetting the rows that have one or more missing values
missing=cars_data2[cars_data2.isnull().any(axis=1)]
print("Missing values:\n",missing)
```

```
#ii) cleaning the data
#Drop features that are not required
cars_data2=cars_data.copy()

cars_data2=cars_data.drop(['Doors','Weight'],axis='columns')
print("Shape= ",cars_data2.shape)

#Dealing with Missing values
#identifying missing values(NaN -> Not a Number)
sum_na=cars_data2.isna().sum()
print("Sum of null values:\n",sum_na)
#subsetting the rows that have one or more missing values
missing=cars_data2[cars_data2.isnull().any(axis=1)]
print("Missing values:\n",missing)
```

5th

```
import numpy as np
#Concatinatin of arrays
a=np.array([[1,2],[3,4]])
b=np.array([[5,6]])
print("concate with axis=0:",np.concatenate((a,b),axis=0))
print("concate with axis=1:",np.concatenate((a,b.T),axis=1))
print(np.concatenate((a,b),axis=None))


a=np.array([[12,4,5],[23,45,66],[45,34,23]])

b=np.array([[1,40,50],[2,4,6],[4,3,2]])


#Verticle stacking

print(np.vstack((a,b)))


#Horizontal stacking

print(np.hstack((a,b)))


print(a.reshape(3,3))




#Sorting

#importing numpy package

import numpy as np

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

print(a)

print("Sorted array:\n",np.sort(a))

print("\n sorted flattened array:\n",np.sort(a,axis=0))


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

print("\n indices that would sort an array",np.argsort(x))

print("\n Sorting complex number:",np.sort_complex([[3 + 4j, 1 - 2j,

                                            5 + 1j, 2 + 2j]]))




#Searching

import numpy as np

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

```
x=np.where(arr==4)
print(x)

arr=np.array([6,7,8,9])
x=np.searchsorted(arr,5)
print(x)

arr=np.array([1,3,5,7])
x=np.searchsorted(arr,[2,4,6])
print(x)


#Splitting
import numpy as np
x=np.arange(9.0)
print(x)
print(np.split(x,3))
print(np.split(x,[3,5,6,10]))

x=np.arange(9)
print(np.array_split(x,4))
a=np.array([[1,3,5,7,9,11],
       [2,4,6,8,10,12]])
print("Splitting along horizontal axis into 2 parts:\n",np.hsplit(a,2))
print("\n Splitting along vertical axis into 2 parts:\n",np.vsplit(a,2))


# 5b)Broadcasting and plotting numpy arrays

import numpy as np
x=np.arange(4)
```

```
print(x)
y=np.ones(5)
print(y)
xx=x.reshape(2,2)
print(xx)
z=np.ones((3,4))
print(z)
print(x.shape)

a=np.array([0.0,10.0,20.0,30.0])
b=np.array([1.0,2.0,3.0])
a[:,np.newaxis]+b



#plotting
import numpy as np
import matplotlib.pyplot as plt

x=np.arange(0,3*np.pi,0.1)
print("x=",x)

y_sin=np.sin(x)
y_cos=np.cos(x)

plt.plot(x,y_sin)
plt.plot(x,y_cos)
plt.xlabel('x values')
plt.ylabel('y sine and cosine values')
plt.title('Sine and Cosine')
plt.legend(['Sine','Cosine'])
plt.show()
```

6th

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

x=np.arange(0,3*np.pi,0.1)
print("x=",x)

y_sin=np.sin(x)
y_cos=np.cos(x)

plt.plot(x,y_sin)
plt.plot(x,y_cos)
plt.xlabel('x values')
plt.ylabel('y sine and cosine values')
plt.title('Sine and Cosine')
plt.legend(['Sine','Cosine'])

plt.show()



import numpy as np
import matplotlib.pyplot as plt
counts=[979,120,12]
fuelType=('Petrol','Diesel','CNG')
index=np.arange(len(fuelType))

plt.bar(index,counts,color=['red','blue','cyan'])
plt.title('Bar plot of fuel types')
plt.xlabel('Fuel Types')
plt.ylabel('frequency')
plt.xticks(index,fuelType,rotation=0)
Plt.show()
```

```
import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)
plt.show()
```

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

# Generate random data for the histogram
data = np.random.randn(1000)
#print(data)
# Plotting a basic histogram
```

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

# Adding labels and title
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Basic Histogram')

# Display the plot
plt.show()
```

7th

```
import numpy as np
a=np.array([[12,4,5],[23,45,66],[45,34,23]])
print("Printing Array\n")
print(a,"\n")

print("Printing numpy array Attributes")
print("1>. Array Shape is: ",a.shape)
print("2>. Array dimensions are ",a.ndim)
print("3>. Datatype of array is ",a.dtype)
print("4>. Length of each element of array in bytes is ",a.itemsize)
print("5>. Number of elements in array are ",a.size)

#seed with a set value in order to ensure that the same random arrays are generated every time this code is run
np.random.seed(0)
```

```
x1=np.random.randint(10,size=6)#one-dimensional array
x2=np.random.randint(10,size=(3,4))#two dimensional array
x3=np.random.randint(10,size=(3,4,5))#three dimensional array
print("One dimensional array\n",x1)
print("Two dimensional array\n",x2)
print("Three dimensional array\n",x3)
```

8th

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt


# Load the dataset from the CSV file
df = pd.read_csv('sample_dataset.csv')

# Extract 'x' and 'y' columns
X = df[['x']]
y = df[['y']]

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Linear Regression model
```

```
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)  # RMSE is the square root of MSE
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

cor_coe=df['x'].corr(df['y'])
# Print evaluation metrics
print(f'Mean Squared Error (MSE): {mse:.2f}')
print(f'Root Mean Squared Error (RMSE): {rmse:.2f}')
print(f'Mean Absolute Error (MAE): {mae:.2f}')
print(f'R-squared (R2): {r2:.2f}')
print(f'Coefficient_coefficent:{cor_coe:.2f}')

# Visualize the regression line
plt.scatter(X_test, y_test, color='black', label='Actual data')
plt.plot(X_test, y_pred, color='blue', linewidth=3, label='Regression line')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Simple Linear Regression')
plt.legend()
plt.show()
```

9th

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,
precision_score, recall_score, f1_score

import matplotlib.pyplot as plt


# Load dataset

dataset = pd.read_csv('purchase.csv')

dataset.head(5)


# Extracting features (X) and labels (Y)

X = dataset.iloc[:, :-1].values

X

Y = dataset.iloc[:, -1].values

Y


# Splitting the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, random_state=1)


# Initializing Logistic Regression model

lr = LogisticRegression(random_state=123)
```

```
# Training the model
lr.fit(X_train, y_train)

# Making predictions on the test set
predictions = lr.predict(X_test)



# Confusion Matrix
cm = confusion_matrix(y_test, predictions)
print("Confusion Matrix: ")
print(cm)

# Classification Report
cr = classification_report(y_test, predictions)
print("Classification Report: ")
print(cr)

# Calculate metrics
lra = accuracy_score(predictions, y_test) * 100
print('Accuracy:', lra)
lrp = precision_score(predictions, y_test) * 100
print('Precision:', lrp)
lrr = recall_score(predictions, y_test) * 100
print('Recall:', lrr)
lrf = f1_score(predictions, y_test) * 100
print('F1 Score:', lrf)
```

10th

```
import pandas as pd
import matplotlib.pyplot as plt

# Load sample time series data (historical stock prices)
# You can replace this with your own time series data
data = {
    'Date': ['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-04', '2022-01-05'],
    'StockPrice': [100, 105, 98, 102, 110]
}

df = pd.DataFrame(data)
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)

# Display the time series data
print("Time Series Data:")
print(df)

# Plot the time series data
df['StockPrice'].plot(title='Stock Price Over Time', xlabel='Date', ylabel='Stock Price')
plt.show()

# Perform basic time series operations
print("\nBasic Time Series Operations:")
print("Average Stock Price:", df['StockPrice'].mean())
print("Maximum Stock Price:", df['StockPrice'].max())
print("Minimum Stock Price:", df['StockPrice'].min())
```

```
# Resample the time series data to monthly frequency and calculate the mean
monthly_mean = df['StockPrice'].resample('M').mean()
print("\nMonthly Mean Stock Price:")
print(monthly_mean)
```

11th

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Sample data
data = {
    'Category': ['A', 'B', 'C', 'D'],
    'Values': [10, 20, 15, 25]
}

# Convert the data into a DataFrame
df = pd.DataFrame(data)

# Create a bar plot using Seaborn
sns.barplot(x='Category', y='Values', data=df)

# Add titles and labels
plt.title('Bar Graph using Seaborn', fontsize=16)
plt.xlabel('Categories', fontsize=14)
plt.ylabel('Values', fontsize=14)

# Show the plot
```

```
plt.show()
```

```
###### import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Sample data
data = {
    'X': [5, 10, 15, 20, 25],
    'Y': [10, 15, 20, 25, 30]
}

# Convert the data into a DataFrame
df = pd.DataFrame(data)
# Create a scatter plot using Seaborn
sns.scatterplot(x='X', y='Y', data=df, color='skyblue',
                edgecolor='black')

# Add titles and labels
plt.title('Scatter Plot using Seaborn', fontsize=16)
plt.xlabel('X-axis', fontsize=14)
plt.ylabel('Y-axis', fontsize=14)
plt.show()
```

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


# Generate random data (for demonstration purposes)
np.random.seed(0)
data = np.random.randn(1000)  # Generate 1000 random numbers from a standard normal distribution


# Create a histogram using Seaborn
sns.histplot(data,color='skyblue', bins=30)


# Add titles and labels
plt.title('Histogram using Seaborn', fontsize=16)
plt.xlabel('Values', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.show()
```