ML LAB  
  
1) CENTRAL TENDENCY  
  
import statistics

# Taking user input

data\_input = input("Enter numbers separated by spaces: ")

data = list(map(float, data\_input.split()))

# Central Tendency Measures

mean = statistics.mean(data)

median = statistics.median(data)

mode = statistics.mode(data)

# Measures of Dispersion

variance = statistics.variance(data)

standard\_deviation = statistics.stdev(data)

# Display the results

print("\nData:", data)

print("Mean:", mean)

print("Median:", median)

print("Mode:", mode)

print("Variance:", variance)

print("Standard Deviation:", standard\_deviation)  
  
  
  
2) STATISTICS   
  
import statistics

# User Input

data\_input = input("Enter numbers separated by spaces: ")

data = list(map(float, data\_input.split()))

# Using statistics library

mean = statistics.mean(data)

median = statistics.median(data)

mode = statistics.mode(data)

variance = statistics.variance(data)

standard\_deviation = statistics.stdev(data)

# Output

print("\nStatistics Results:")

print("Mean:", mean)

print("Median:", median)

print("Mode:", mode)

print("Variance:", variance)

print("Standard Deviation:", standard\_deviation)  
  
  
  
  
  
  
  
  
3) MATH  
  
import math

# User Input

num = float(input("Enter a number: "))

# Using math library

sqrt\_num = math.sqrt(num)

power\_num = math.pow(num, 2) # num^2

log\_num = math.log(num)

factorial\_num = math.factorial(int(num)) # Only integer

# Output

print("\nMath Results:")

print("Square Root:", sqrt\_num)

print("Power (number^2):", power\_num)

print("Natural Log:", log\_num)

print("Factorial:", factorial\_num)  
  
  
4) NUMPY  
  
import numpy as np

# User Input

data\_input = input("Enter numbers separated by spaces: ")

data = np.array(list(map(float, data\_input.split())))

# Using numpy library

mean = np.mean(data)

median = np.median(data)

std\_dev = np.std(data)

sum\_all = np.sum(data)

# Output

print("\nNumpy Results:")

print("Array:", data)

print("Mean:", mean)

print("Median:", median)

print("Standard Deviation:", std\_dev)

print("Sum of all elements:", sum\_all)  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
5) SCIPY   
  
from scipy import special

import math

# User Input

a = int(input("Enter first number: "))

b = int(input("Enter second number: "))

# Using scipy.special (Gamma function, factorial etc.)

gamma\_a = special.gamma(a) # (a-1)!

gamma\_b = special.gamma(b)

# Using math.gcd (greatest common divisor)

gcd = math.gcd(a, b)

# Output

print("\nScipy Results:")

print(f"Gamma({a}) (which is {a-1}!):", gamma\_a)

print(f"Gamma({b}) (which is {b-1}!):", gamma\_b)

print(f"GCD of {a} and {b}:", gcd)  
  
  
6) PANDAS Library   
  
import pandas as pd

# User Input

names = input("Enter names separated by commas: ").split(',')

marks = input("Enter marks separated by commas: ").split(',')

# Create DataFrame

data = {'Name': names, 'Marks': list(map(int, marks))}

df = pd.DataFrame(data)

# Output

print("\nPandas DataFrame:")

print(df)

# Some basic operations

print("\nBasic Operations:")

print("Average Marks:", df['Marks'].mean())

print("Highest Marks:", df['Marks'].max())

print("Lowest Marks:", df['Marks'].min())  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
7) MATPLOTLIB  
  
import matplotlib.pyplot as plt

# User Input

subjects = input("Enter subjects separated by commas: ").split(',')

scores = input("Enter scores separated by commas: ").split(',')

# Converting scores to integers

scores = list(map(int, scores))

# Create Bar Chart

plt.bar(subjects, scores, color='skyblue')

plt.xlabel('Subjects')

plt.ylabel('Scores')

plt.title('Scores in Different Subjects')

plt.show()  
  
  
  
8) SIMPLE LINEAR REGRESSION  
  
import numpy as np

from sklearn.linear\_model import LinearRegression

X = np.array(list(map(float, input("Enter X values separated by spaces: ").split()))).reshape(-1, 1)

y = np.array(list(map(float, input("Enter Y values separated by spaces: ").split())))

model = LinearRegression()

model.fit(X, y)

print("\nSimple Linear Regression Model:")

print(f"Coefficient (slope): {model.coef\_[0]}")

print(f"Intercept: {model.intercept\_}")

x\_new = float(input("\nEnter a new X value to predict Y: "))

print(f"Predicted Y value: {model.predict([[x\_new]])[0]}")

9) Multiple Linear Regression  
  
import numpy as np

from sklearn.linear\_model import LinearRegression

X\_input = []

y\_input = []

n = int(input("Enter number of houses: "))

for i in range(n):

X\_input.append(list(map(float, input(f"Features for house {i+1}: ").split())))

y\_input.append(float(input(f"Price for house {i+1}: ")))

X = np.array(X\_input)

y = np.array(y\_input)

model = LinearRegression()

model.fit(X, y)

print("\nMultiple Linear Regression Model:")

print(f"Coefficients: {model.coef\_}")

print(f"Intercept: {model.intercept\_}")

new\_features = list(map(float, input("\nEnter new house features: ").split()))

predicted\_price = model.predict([new\_features])

print(f"Predicted House Price: {predicted\_price[0]}")

10) Decision tree using sklearn  
  
from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

X\_input = []

y\_input = []

n = int(input("Enter number of data points: "))

for i in range(n):

X\_input.append(list(map(float, input(f"Features for point {i+1}: ").split())))

y\_input.append(input(f"Label for point {i+1}: "))

model = DecisionTreeClassifier(criterion='entropy', max\_depth=3)

model.fit(X\_input, y\_input)

new\_features = list(map(float, input("\nEnter new features to predict label: ").split()))

print(f"\nPredicted Label: {model.predict([new\_features])[0]}")

tree.plot\_tree(model, filled=True)

11) KNN using sklearn  
  
from sklearn.neighbors import KNeighborsClassifier

X\_input = []

y\_input = []

n = int(input("Enter number of data points: "))

for i in range(n):

X\_input.append(list(map(float, input(f"Features for point {i+1}: ").split())))

y\_input.append(input(f"Label for point {i+1}: "))

k = int(input("Enter value of k: "))

model = KNeighborsClassifier(n\_neighbors=k)

model.fit(X\_input, y\_input)

new\_features = list(map(float, input("\nEnter new features to predict label: ").split()))

print(f"\nPredicted Label: {model.predict([new\_features])[0]}")

12) Logistic Regression using sklearn  
  
from sklearn.linear\_model import LogisticRegression

X\_input = []

y\_input = []

n = int(input("Enter number of data points: "))

for i in range(n):

features = list(map(float, input(f"Features for point {i+1} (e.g., Height Weight Age): ").split()))

label = int(input(f"Label for point {i+1} (0 or 1): "))

X\_input.append(features)

y\_input.append(label)

model = LogisticRegression()

model.fit(X\_input, y\_input)

new\_features = list(map(float, input("\nEnter new features to predict label: ").split()))

prediction = model.predict([new\_features])

probability = model.predict\_proba([new\_features])

print(f"\nPredicted Label: {prediction[0]}")

print(f"Probability of each class: {probability[0]}")

12) K-Means Clustering  
  
from sklearn.cluster import KMeans

X\_input = []

n = int(input("Enter number of data points: "))

for i in range(n):

features = list(map(float, input(f"Enter features for point {i+1}: ").split()))

X\_input.append(features)

k = int(input("Enter the number of clusters (k): "))

model = KMeans(n\_clusters=k, random\_state=0)

model.fit(X\_input)

predictions = model.predict(X\_input)

print("\nCluster assignments for each data point:")

for i in range(n):

print(f"Point {X\_input[i]} -> Cluster {predictions[i]}")

print("\nCluster Centers:")

print(model.cluster\_centers\_)