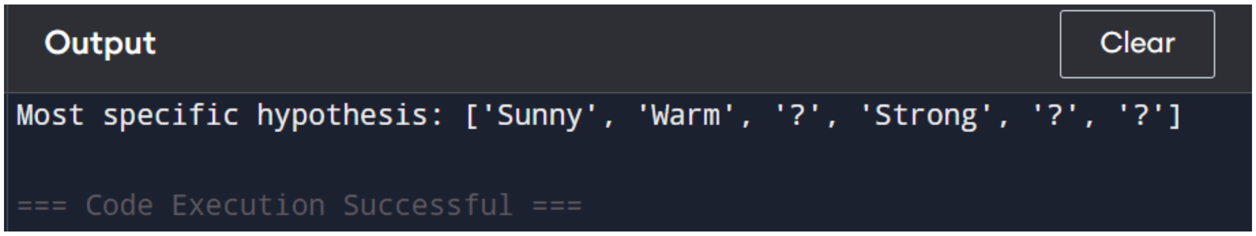
Exp-01

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

program

```
# Training data
data = [
    ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'],
    ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'],
    ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
]

# Initialize hypothesis
h = ['0'] * (len(data[0]) - 1)

# FIND-S algorithm
for sample in data:
    if sample[-1] == 'Yes':
        for i in range(len(h)):
            if h[i] == '0':
                h[i] = sample[i]
            elif h[i] != sample[i]:
                h[i] = '?'


print("Most specific hypothesis:", h)
```

output



Output

Clear

```
Most specific hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']

=== Code Execution Successful ===
```

Exp-02

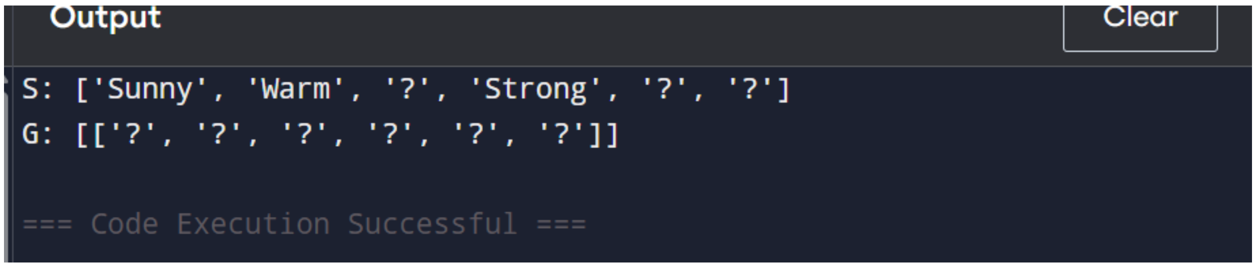
For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm in python to output a description of the set of all hypotheses consistent with the training examples

Program

```
# Training data

data = [

 ['Sunny','Warm','Normal','Strong','Warm','Same','Yes'],

 ['Sunny','Warm','High','Strong','Warm','Same','Yes'],

 ['Rainy','Cold','High','Strong','Warm','Change','No'],

 ['Sunny','Warm','High','Strong','Cool','Change','Yes']

]


X = [row[:-1] for row in data]

Y = [row[-1] for row in data]
```

```
S = X[0][:]
G = [['?'] * len(S)]


for i in range(len(X)):
    if Y[i] == 'Yes':
        for j in range(len(S)):
            if S[j] != X[i][j]:
                S[j] = '?'


print("S:", S)
print("G:", G)
```

output



```
S: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
G: [['?', '?', '?', '?', '?', '?']]

=== Code Execution Successful ===
```

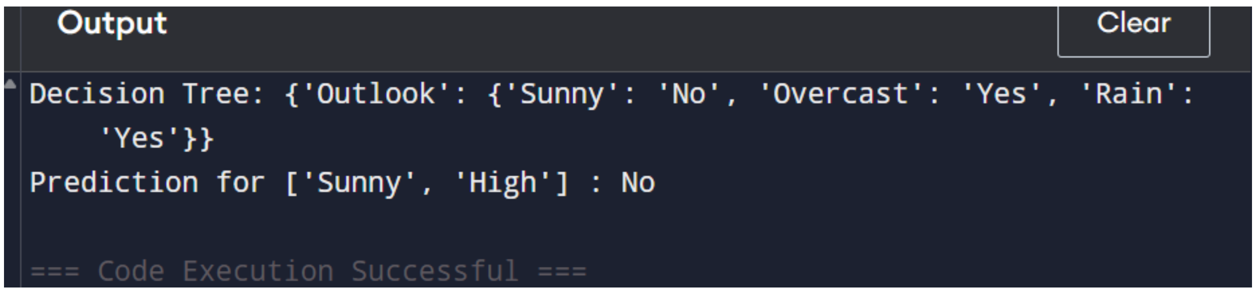
Exp-03

Demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Program

```
import math
```

```
# Dataset (last column = class label)
data = [
    ['Sunny','High','No'],
    ['Sunny','High','No'],
    ['Overcast','High','Yes'],
    ['Rain','Normal','Yes'],
    ['Rain','Normal','Yes']
]
labels = ['Outlook','Humidity']

# Entropy
def entropy(d):
    c={};[c.update({r[-1]:c.get(r[-1],0)+1}) for r in d]
    return -sum((v/len(d))*math.log2(v/len(d)) for v in c.values())

# Information Gain
def gain(d,col):
    return entropy(d)-sum((len(s)/len(d))*entropy(s)
        for v in set(r[col] for r in d)
        for s in [[r for r in d if r[col]==v]])

# ID3 Tree
def id3(d,l):
    cls=[r[-1] for r in d]
    if cls.count(cls[0])==len(cls): return cls[0]
    best=max(range(len(l)), key=lambda i:gain(d,i));
t={l[best]:{}}
    for v in set(r[best] for r in d):
        s=[r[:best]+r[best+1:] for r in d if r[best]==v]
        nl=l[:best]+l[best+1:]
        t[l[best]][v]=id3(s,nl)
    return t

tree=id3(data,labels)
print("Decision Tree:", tree)

# Classification
def classify(t,l,s):
    r=list(t.keys())[0]; v=s[l.index(r)]; res=t[r][v]
    return res if type(res)==str else classify(res,l,s)

new_sample = ['Sunny','High']
print("Prediction for", new_sample, ":", classify(tree,labels,new_sample))
```



Output Clear

```
Decision Tree: {'Outlook': {'Sunny': 'No', 'Overcast': 'Yes', 'Rain':
    'Yes'}}
Prediction for ['Sunny', 'High'] : No

=== Code Execution Successful ===
```

Exp-04

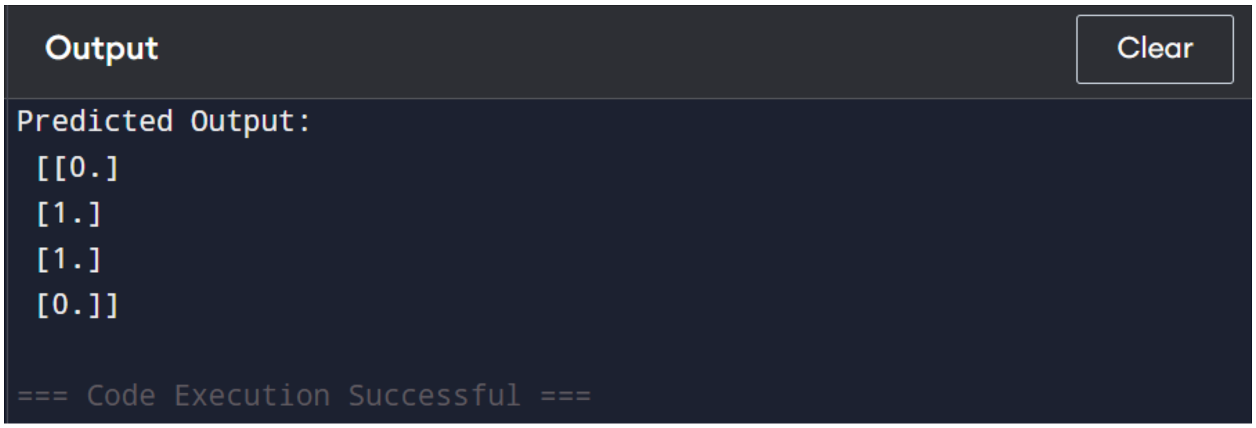
Build an Artificial Neural Network by implementing the Backpropagation algorithm and test the same using appropriate data sets.

Program

```
import numpy as np


# Dataset: XOR example
X = np.array([[0,0],[0,1],[1,0],[1,1]])
Y = np.array([[0],[1],[1],[0]])


# Sigmoid activation
sigmoid = lambda x: 1/(1+np.exp(-x))
sigmoid_deriv = lambda x: x*(1-x)
```

```
# Initialize weights
np.random.seed(1)
wh = np.random.rand(X.shape[1],2)
bh = np.random.rand(1,2)
wout = np.random.rand(2,Y.shape[1])
bout = np.random.rand(1,Y.shape[1])
lr = 0.5

# Training
for _ in range(10000):
    hidden = sigmoid(np.dot(X,wh)+bh)
    output = sigmoid(np.dot(hidden,wout)+bout)
    e = Y - output
    d_output = e * sigmoid_deriv(output)
    d_hidden = d_output.dot(wout.T) * sigmoid_deriv(hidden)
    wout += hidden.T.dot(d_output)*lr
    wh += X.T.dot(d_hidden)*lr
    bout += np.sum(d_output,axis=0,keepdims=True)*lr
    bh += np.sum(d_hidden,axis=0,keepdims=True)*lr
```

```
# Test
hidden = sigmoid(np.dot(X,wh)+bh)
output = sigmoid(np.dot(hidden,wout)+bout)
print("Predicted Output:\n", np.round(output))
```



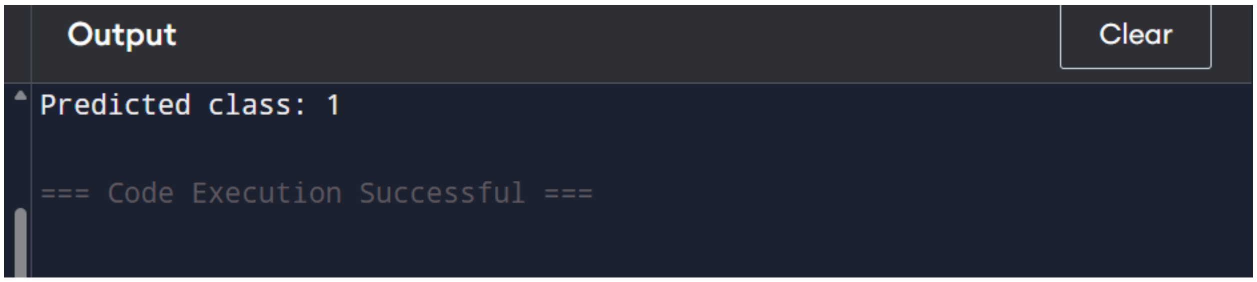
```
Predicted Output:
 [[0.]
 [1.]
 [1.]
 [0.]]

=== Code Execution Successful ===
```

Exp-05

Write a program for Implementation of K-Nearest Neighbours (K-NN) in Python

Program

```
import numpy as np
from collections import Counter

# Dataset: points (x, y) and their class
X = np.array([[1,2],[2,3],[3,1],[6,5],[7,7],[8,6]])
Y = np.array([0,0,0,1,1,1])

# K-NN prediction
def knn(x, X, Y, k=3):
    distances = np.sqrt(np.sum((X - x)**2, axis=1))
    nearest = Y[np.argsort(distances)[:k]]
    return Counter(nearest).most_common(1)[0][0]

# Test new sample
sample = np.array([5,5])
print("Predicted class:", knn(sample, X, Y, k=3))
```



```
Predicted class: 1

=== Code Execution Successful ===
```

Exp-06

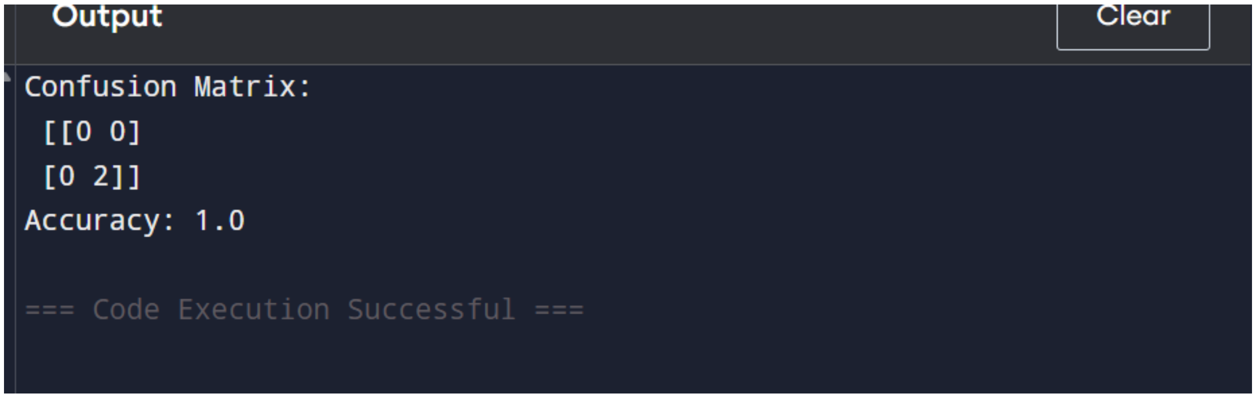
Write a program to implement Naïve Bayes algorithm in python and to display the results using confusion matrix and accuracy.

Program

```
import numpy as np


# Dataset: [feature1, feature2], class
X = np.array([[1,1],[2,1],[1,2],[6,6],[7,5],[8,6]])
y = np.array([0,0,0,1,1,1])


# Train/test split
X_train, y_train = X[:4], y[:4]
X_test, y_test = X[4:], y[4:]


# Class stats
classes = np.unique(y_train)
```

```
mean_std = {c:(X_train[y_train==c].mean(0),
X_train[y_train==c].std(0)) for c in classes}
priors = {c:np.mean(y_train==c) for c in classes}


# Gaussian probability with small epsilon to avoid divide by
zero
def g(x,m,s):
    s = s if s>0 else 1e-6
    return (1/(np.sqrt(2*np.pi)*s)) * np.exp(-((x-
m)**2)/(2*s**2))


# Predict
def predict(x):
    return
max({c:priors[c]*np.prod([g(x[i],mean_std[c][0][i],mean_std[c]
[1][i]) for i in range(len(x))]) for c in classes}, key=lambda k:k)


# Predictions
y_pred = np.array([predict(x) for x in X_test])


# Confusion matrix & accuracy
```

```
cm = np.zeros((len(classes),len(classes)),int)
for t,p in zip(y_test,y_pred): cm[t][p]+=1
accuracy = np.sum(np.diag(cm))/len(y_test)

print("Confusion Matrix:\n", cm)
print("Accuracy:", accuracy)
```



```
Confusion Matrix:
 [[0 0]
 [0 2]]
Accuracy: 1.0

=== Code Execution Successful ===
```

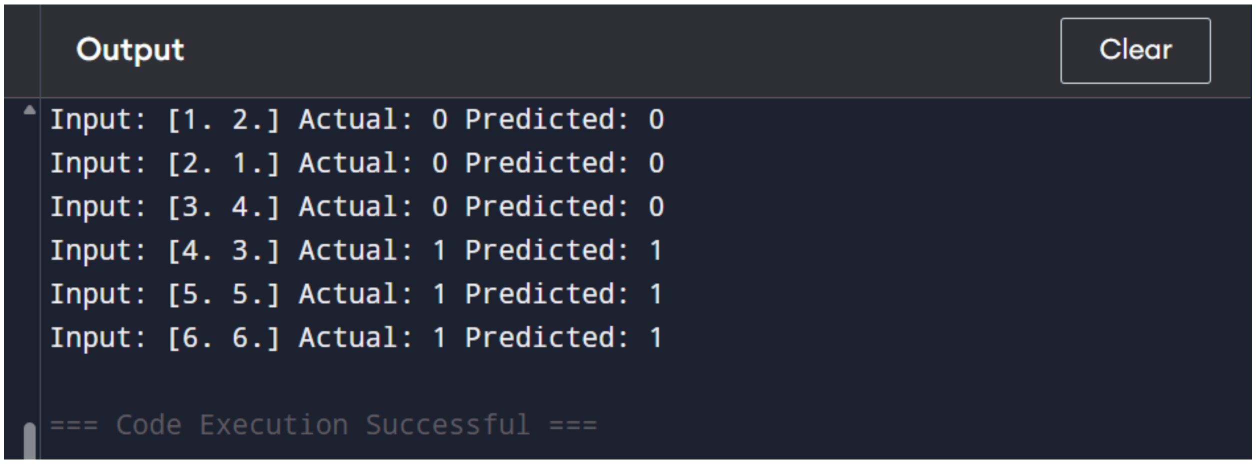
Exp-07

Write a program to implement Logistic Regression (LR) algorithm in python

Program

```
import numpy as np


# Dataset: [feature1, feature2], class
X = np.array([[1,2],[2,1],[3,4],[4,3],[5,5],[6,6]])
y = np.array([0,0,0,1,1,1]).reshape(-1,1)
```

```
# Add bias
X = np.hstack([np.ones((X.shape[0],1)), X])
w = np.random.rand(X.shape[1],1)
lr = 0.1

sigmoid = lambda z: 1/(1+np.exp(-z))

# Training
for _ in range(1000):
    w -= lr * X.T @ (sigmoid(X@w)-y) / y.size

# Predict function
predict = lambda x: 1 if sigmoid(np.dot([1,*x], w))>=0.5 else 0

# Test
for xi, yi in zip(X[:,1:], y):
    print("Input:", xi, "Actual:", yi[0], "Predicted:", predict(xi))
```



```
Input: [1. 2.] Actual: 0 Predicted: 0
Input: [2. 1.] Actual: 0 Predicted: 0
Input: [3. 4.] Actual: 0 Predicted: 0
Input: [4. 3.] Actual: 1 Predicted: 1
Input: [5. 5.] Actual: 1 Predicted: 1
Input: [6. 6.] Actual: 1 Predicted: 1

=== Code Execution Successful ===
```

Exp-08

Write a program to implement Linear Regression (LR) algorithm in python

Program

```
import numpy as np


# Dataset: X = input, y = output
X = np.array([[1],[2],[3],[4],[5]])
y = np.array([[2],[4],[6],[8],[10]])


# Add bias term
X_b = np.hstack([np.ones((X.shape[0],1)), X])


# Compute weights using Normal Equation: w = (X^T X)^-1
X^T y
w = np.linalg.inv(X_b.T @ X_b) @ X_b.T @ y
```

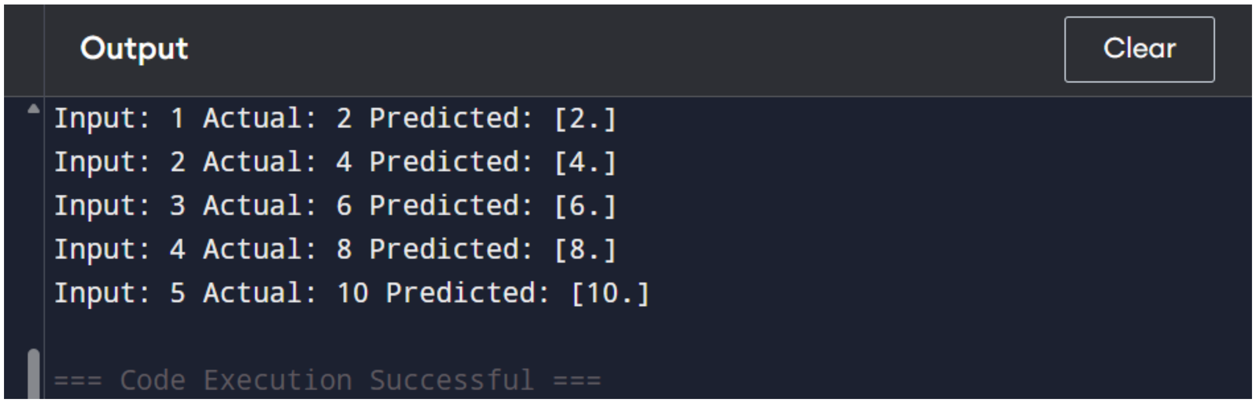
# Predict function

predict = lambda x: np.dot([1, x], w)

# Test

for xi, yi in zip(X, y):

    print("Input:", xi[0], "Actual:", yi[0], "Predicted:", predict(xi[0]))



```
Input: 1 Actual: 2 Predicted: [2.]
Input: 2 Actual: 4 Predicted: [4.]
Input: 3 Actual: 6 Predicted: [6.]
Input: 4 Actual: 8 Predicted: [8.]
Input: 5 Actual: 10 Predicted: [10.]

=== Code Execution Successful ===
```

Exp-09

Compare Linear and Polynomial Regression using Python

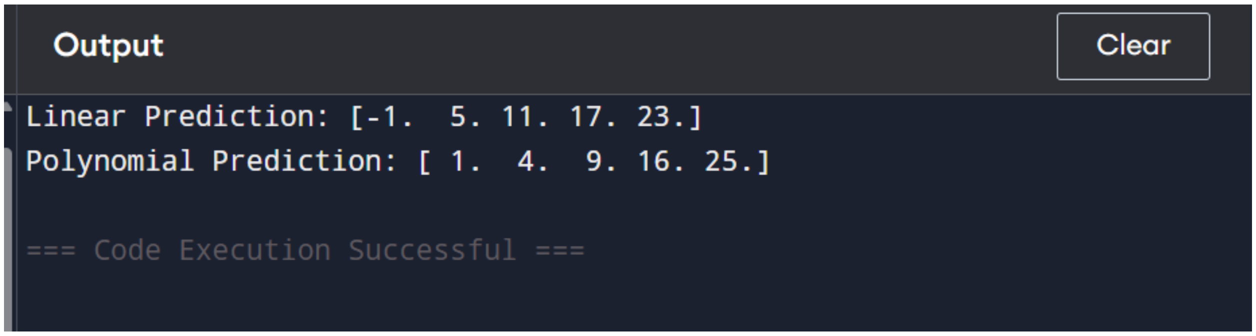
Program

import numpy as np

# Data

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

```
y = np.array([1, 4, 9, 16, 25])

# ---- Linear Regression (y = a*X + b) ----
A = np.vstack([X, np.ones(len(X))]).T
a, b = np.linalg.lstsq(A, y, rcond=None)[0]
linear_pred = a*X + b
print("Linear Prediction:", linear_pred)

# ---- Polynomial Regression (degree 2: y = a*X^2 + b*X + c) --
--
poly_A = np.vstack([X**2, X, np.ones(len(X))]).T
a2, b2, c2 = np.linalg.lstsq(poly_A, y, rcond=None)[0]
poly_pred = a2*X**2 + b2*X + c2
print("Polynomial Prediction:", poly_pred)
```



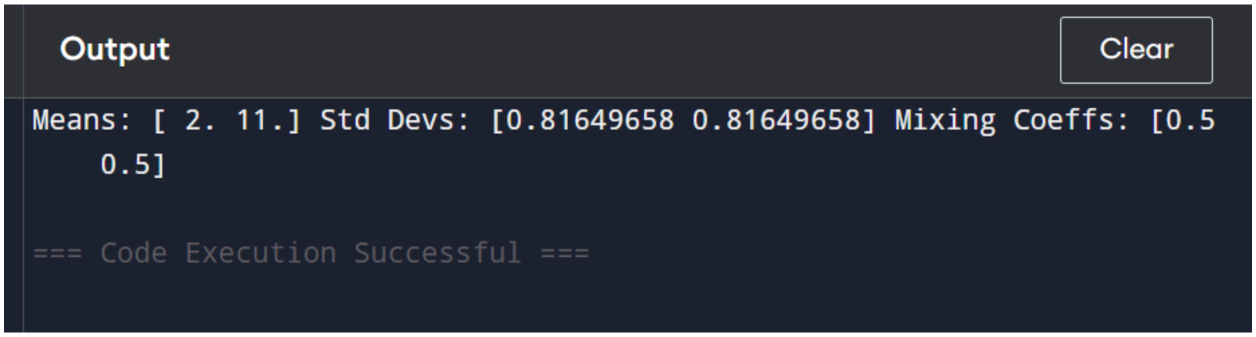
```
Linear Prediction: [-1.  5. 11. 17. 23.]
Polynomial Prediction: [ 1.  4.  9. 16. 25.]

=== Code Execution Successful ===
```

Exp-10

Write a Python Program to Implement Expectation & Maximization Algorithm

Program

```
import numpy as np


X = np.array([1,2,3,10,11,12])
mu, sigma, pi = np.array([2.0,11.0]), np.array([1.0,1.0]),
np.array([0.5,0.5])


for _ in range(5):
    gamma =
np.array([pi[k]*(1/(np.sqrt(2*np.pi)*sigma[k]))*np.exp(-(X-
mu[k])**2/(2*sigma[k]**2)) for k in range(2)]).T
    gamma /= gamma.sum(axis=1, keepdims=True)
    for k in range(2):
        Nk = gamma[:,k].sum()
        mu[k] = (gamma[:,k] @ X)/Nk
        sigma[k] = np.sqrt((gamma[:,k] @ (X-mu[k])**2)/Nk)
        pi[k] = Nk/len(X)


print("Means:", mu, "Std Devs:", sigma, "Mixing Coeffs:", pi)
```



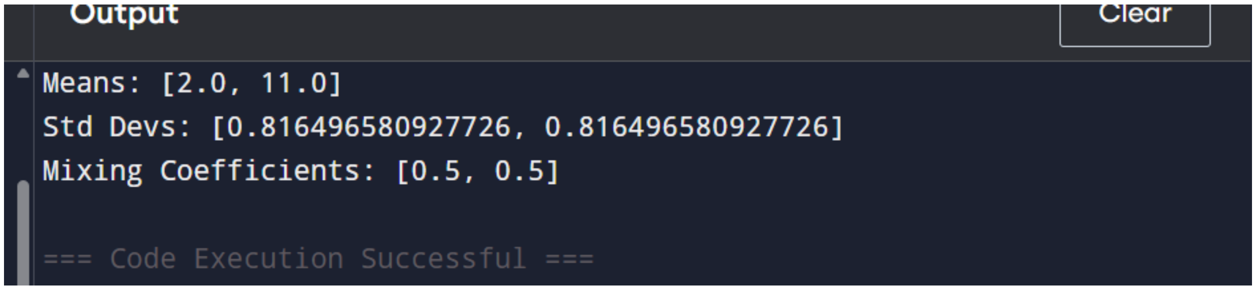
```
Means: [ 2. 11.] Std Devs: [0.81649658 0.81649658] Mixing Coeffs: [0.5
    0.5]

=== Code Execution Successful ===
```

Exp-11

Write a program for the task of Credit Score Classification

Program

```
import numpy as np


X = np.array([1,2,3,10,11,12])

mu, sigma, pi = [2,11], [1,1], [0.5,0.5]


for _ in range(5):

    gamma =
np.array([pi[k]/(sigma[k]*np.sqrt(2*np.pi))*np.exp(-(X-
mu[k])**2/(2*sigma[k]**2)) for k in range(2)]).T

    gamma /= gamma.sum(axis=1, keepdims=True)

    for k in range(2):

        Nk = gamma[:,k].sum()

        mu[k] = float((gamma[:,k] @ X)/Nk)
        sigma[k] = float(np.sqrt((gamma[:,k] @ (X-mu[k])**2)/Nk))
        pi[k] = float(Nk/len(X))


print(f"Means: {mu}")
print(f"Std Devs: {sigma}")
print(f"Mixing Coefficients: {pi}")
```



```
Means: [2.0, 11.0]
Std Devs: [0.816496580927726, 0.816496580927726]
Mixing Coefficients: [0.5, 0.5]


=== Code Execution Successful ===
```

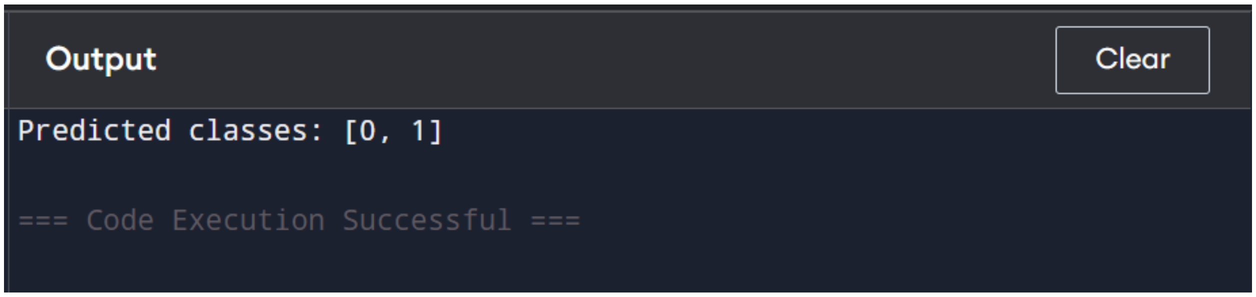
Exp-12

Implement Iris Flower Classification using KNN

Program

```
import numpy as np


# Small Iris dataset: [sepal_len, sepal_wid, petal_len, petal_wid]
X = np.array([
    [5.1,3.5,1.4,0.2],[4.9,3.0,1.4,0.2],[5.0,3.6,1.4,0.2],  # Class 0 (Setosa)
    [6.5,3.0,5.2,2.0],[6.2,3.4,5.4,2.3],[5.9,3.0,5.1,1.8]   # Class 1
(Versicolor)
])
y = np.array([0,0,0,1,1,1])

# KNN prediction function
def knn_predict(x, k=3):
    distances = np.sqrt(((X - x)**2).sum(axis=1))  # Euclidean
distance
    idx = distances.argsort()[:k]                # indices of k nearest
    vals, counts = np.unique(y[idx], return_counts=True)
    return int(vals[counts.argmax()])

# Test samples
test_samples = np.array([[5.0,3.4,1.5,0.2],[6.0,3.0,5.0,1.8]])
predictions = [knn_predict(x) for x in test_samples]
```

print("Predicted classes:", predictions)



```
Predicted classes: [0, 1]

=== Code Execution Successful ===
```

Exp-13

Implement the Car Price Prediction Model using Python

Program

import numpy as np

# Sample car data: [mileage in 1000 km, age in years]

X = np.array([[10, 1], [20, 2], [30, 3], [40, 4], [50, 5]], dtype=float)

y = np.array([20, 18, 15, 12, 10], dtype=float) # Prices in 1000 $

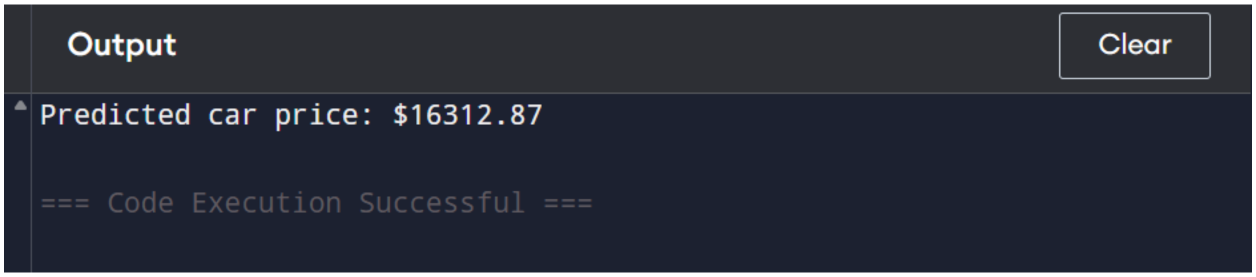
# Add bias column for intercept

X_b = np.c_[np.ones((X.shape[0],1)), X] # np.c_ stacks column

# Compute weights using pseudo-inverse (robust)

w = np.linalg.pinv(X_b) @ y

```
# Predict price for a new car: 25,000 km mileage, 2 years old
new_car = np.array([1, 25, 2], dtype=float)  # Add bias term
pred_price = new_car @ w

print(f"Predicted car price: ${pred_price*1000:.2f}")
```



Output Clear

```
Predicted car price: $16312.87

=== Code Execution Successful ===
```

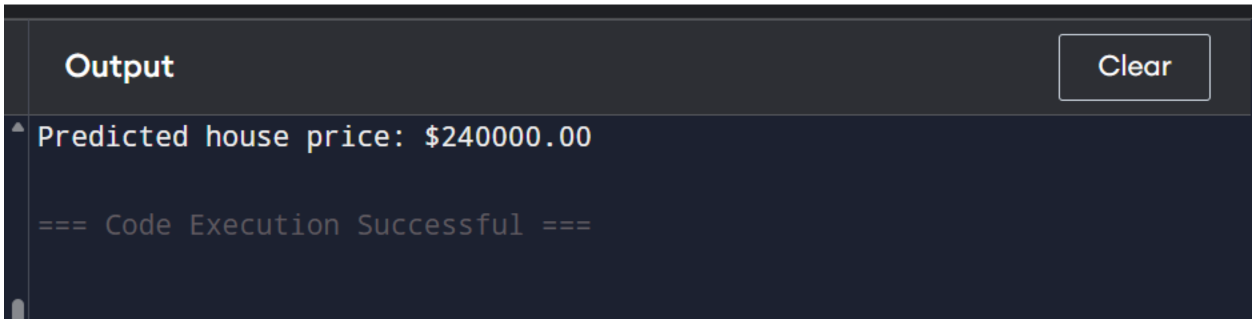
Exp-14

Implement House price Prediction using appropriate machine learning algorithm

Program

```
import numpy as np

# Adjusted data (scaled features)
X = np.array([[1.0, 2.0], [1.5, 3.0], [2.0, 3.0], [2.5, 4.0], [3.0,
4.0]], dtype=float)  # Size in 1000 sq.ft
y = np.array([200, 220, 240, 260, 280], dtype=float)  # Prices in
1000 $
```

```
# Add bias
X_b = np.c_[np.ones((X.shape[0],1)), X]

# Compute weights
w = np.linalg.pinv(X_b) @ y

# Predict for 2.0 (2000 sq.ft), 3 bedrooms
new_house = np.array([1, 2.0, 3.0])
pred_price = new_house @ w

print(f"Predicted house price: ${pred_price*1000:.2f}")
```



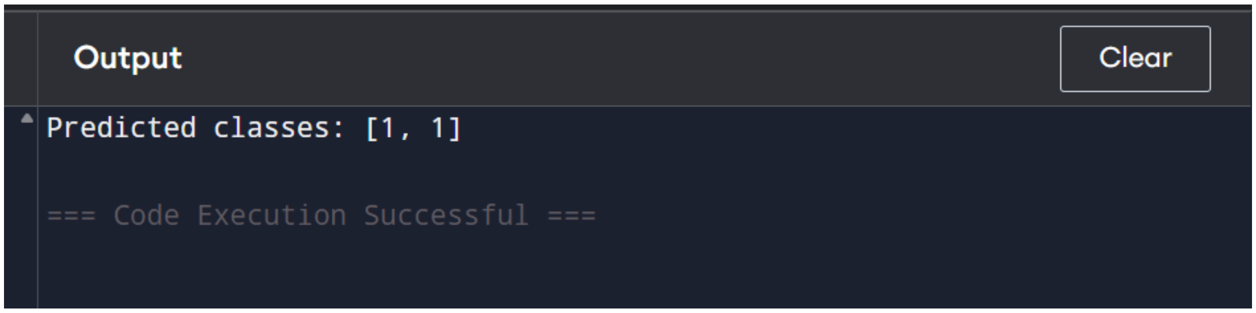
```
Predicted house price: $240000.00

=== Code Execution Successful ===
```

Exp-15

Implement Iris Flower Classification using Naive Bayes classifier

Program

```
import numpy as np

# Small Iris-like dataset: [sepal_len, sepal_wid, petal_len,
petal_wid]
X = np.array([
    [5.1,3.5,1.4,0.2],[4.9,3.0,1.4,0.2],[5.0,3.6,1.4,0.2],  # Class 0
    [6.5,3.0,5.2,2.0],[6.2,3.4,5.4,2.3],[5.9,3.0,5.1,1.8]   # Class 1
])
y = np.array([0,0,0,1,1,1])

# Gaussian Naive Bayes training
classes = np.unique(y)
mean = {c: X[y==c].mean(axis=0) for c in classes}
var  = {c: X[y==c].var(axis=0) for c in classes}
priors = {c: np.mean(y==c) for c in classes}

# Prediction function
def predict(x):
    posteriors = []
    for c in classes:
        likelihood = np.prod(1/np.sqrt(2*np.pi*var[c]) * np.exp(-(x-mean[c])**2/(2*var[c])))
        posterior = likelihood * priors[c]
        posteriors.append(posterior)
    return int(classes[np.argmax(posteriors)])  # Convert to plain int


# Test samples
test_samples = np.array([[5.0,3.4,1.5,0.2],[6.0,3.0,5.0,1.8]])
predictions = [predict(x) for x in test_samples]


print("Predicted classes:", predictions)
```



```
Predicted classes: [1, 1]

=== Code Execution Successful ===
```

Exp-16

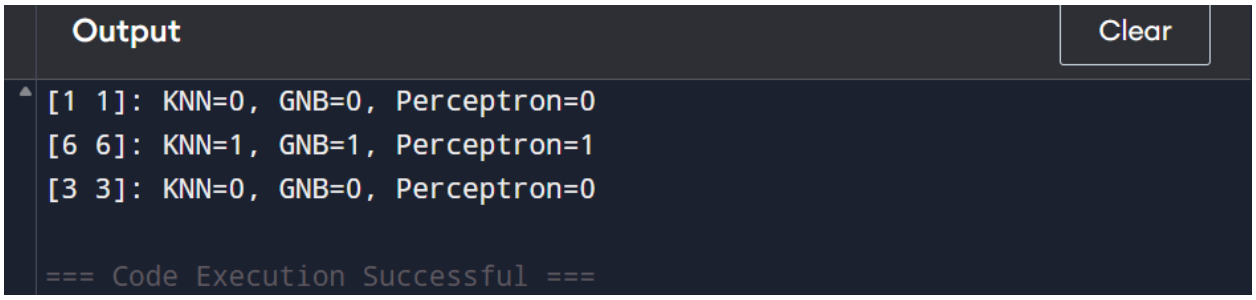
Compare different types Classification Algorithms and evaluate their performance.

Program

```
import numpy as np
```

```
# Dataset: [x1, x2], labels 0 or 1
X = np.array([[1,2],[2,1],[1.5,1.8],[5,6],[6,5],[5.5,5.5]])
y = np.array([0,0,0,1,1,1])

# ----- KNN -----
def knn(x,k=3):
    d = np.sqrt(((X - x)**2).sum(axis=1))
    return int(np.bincount(y[d.argsort()[:k]]).argmax())

# ----- Gaussian Naive Bayes -----
classes = np.unique(y)
mean = {c: X[y==c].mean(axis=0) for c in classes}
var  = {c: X[y==c].var(axis=0) for c in classes}
priors = {c: np.mean(y==c) for c in classes}
def gnb(x):
    post=[]
    for c in classes:
        like = np.prod(1/np.sqrt(2*np.pi*var[c])*np.exp(-(x-
mean[c])**2/(2*var[c])))
```

```
        post.append(like*priors[c])
    return int(classes[np.argmax(post)])


# ----- Perceptron -----
w = np.zeros(X.shape[1]+1); lr, epochs = 0.1, 10
X_b = np.c_[np.ones(X.shape[0]), X]
for _ in range(epochs):
    for xi, yi in zip(X_b, y):
        w += lr*(yi - (1 if xi@w>=0 else 0))*xi
def perceptron(x):
    return 1 if np.r_[1,x]@w>=0 else 0


# ----- Test -----
for x in np.array([[1,1],[6,6],[3,3]]):
    print(f"{x}: KNN={knn(x)}, GNB={gnb(x)},
Perceptron={perceptron(x)}")
```



```
[1 1]: KNN=0, GNB=0, Perceptron=0
[6 6]: KNN=1, GNB=1, Perceptron=1
[3 3]: KNN=0, GNB=0, Perceptron=0

=== Code Execution Successful ===
```

Exp-17

Implement Mobile Price Prediction using appropriate machine learning algorithm

Program

```
import numpy as np


# Sample mobile dataset: [RAM in GB, Storage in GB, Battery in mAh]
X = np.array([
    [2, 16, 3000],
    [3, 32, 3500],
    [4, 64, 4000],
    [6, 128, 4500],
    [8, 256, 5000]
], dtype=float)


# Prices in $ (in hundreds)
y = np.array([150, 200, 250, 350, 500], dtype=float)


# Add bias column
X_b = np.c_[np.ones((X.shape[0],1)), X]  # shape: (n_samples, n_features+1)
```

# Compute weights using pseudo-inverse (Linear Regression)

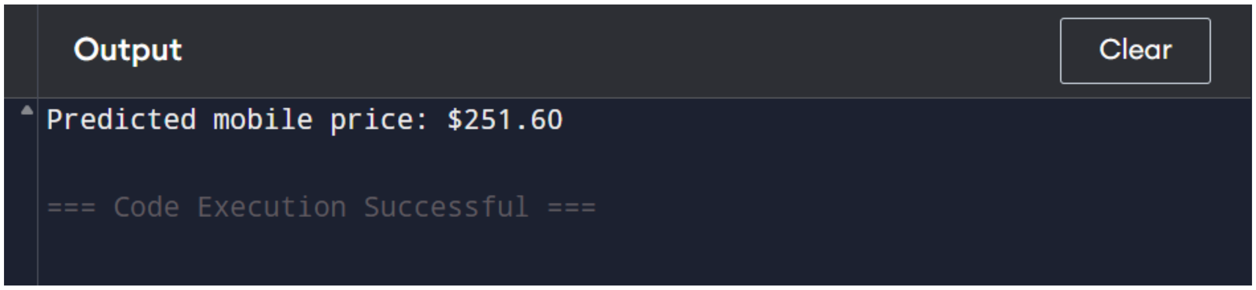
w = np.linalg.pinv(X_b) @ y

# Predict price for a new mobile: 4GB RAM, 64GB Storage, 4000mAh battery

new_mobile = np.array([1, 4, 64, 4000], dtype=float) # add bias

pred_price = new_mobile @ w

print(f"Predicted mobile price: ${pred_price:.2f}")



```
Predicted mobile price: $251.60

=== Code Execution Successful ===
```

Exp-18

Implement Perceptron based IRIS classification

Program

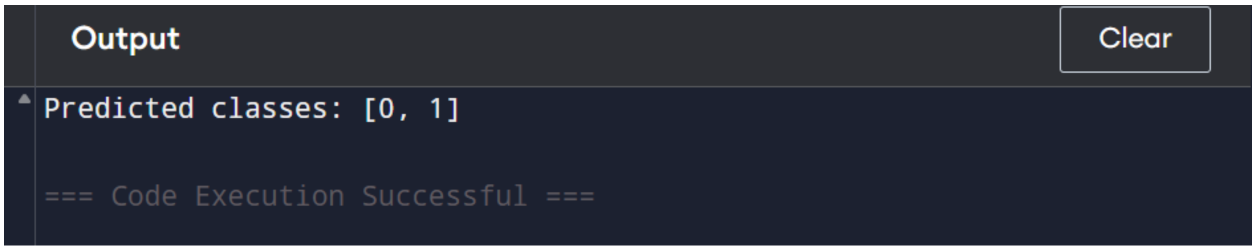
import numpy as np

# Small Iris dataset: Setosa=0, Versicolor=1

X = np.array([

```
    [5.1,3.5,1.4,0.2],[4.9,3.0,1.4,0.2],[5.0,3.6,1.4,0.2],  # Class 0
    [6.5,3.0,4.7,1.4],[6.4,3.2,4.5,1.5],[6.9,3.1,4.9,1.5]   # Class 1
])
y = np.array([0,0,0,1,1,1])

# Add bias term
X_b = np.c_[np.ones(X.shape[0]), X]

# Initialize weights
w = np.zeros(X_b.shape[1])
lr = 0.1
epochs = 20

# Train Perceptron
for _ in range(epochs):
    for xi, yi in zip(X_b, y):
        pred = 1 if xi @ w >= 0 else 0
        w += lr * (yi - pred) * xi

# Prediction function
```

```
def predict(x):
    return 1 if np.r_[1, x] @ w >= 0 else 0

# Test samples
test_samples = np.array([[5.1,3.4,1.5,0.2],[6.5,3.0,5.2,2.0]])
predictions = [predict(x) for x in test_samples]

print("Predicted classes:", predictions)
```



```
Predicted classes: [0, 1]

=== Code Execution Successful ===
```

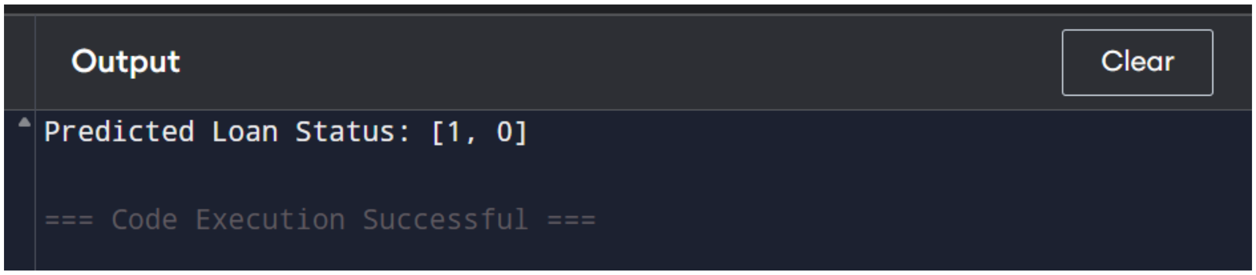
Exp-19

Implementation of Naive Bayes classification for Bank Loan prediction

Program

```
import numpy as np

# Sample dataset: [Income in $1000s, CreditScore, HasJob(1=yes,0=no)]
X = np.array([
    [50, 700, 1],
    [20, 650, 0],
    [35, 600, 1],
    [80, 720, 1],
    [25, 580, 0],
    [90, 750, 1]
])
y = np.array([1, 0, 0, 1, 0, 1])  # Loan Approved=1, Rejected=0

# Small value to avoid division by zero
epsilon = 1e-6

# Compute mean, variance, priors per class
classes = np.unique(y)
mean = {c: X[y==c].mean(axis=0) for c in classes}
var  = {c: X[y==c].var(axis=0) + epsilon for c in classes}  # add
epsilon
priors = {c: np.mean(y==c) for c in classes}

# Gaussian Naive Bayes prediction
```

```
def predict(x):
    posteriors = []
    for c in classes:
        likelihood = np.prod(1/np.sqrt(2*np.pi*var[c]) * np.exp(-(x-mean[c])**2/(2*var[c])))
        posteriors.append(likelihood * priors[c])
    return int(classes[np.argmax(posteriors)])


# Test samples: [Income, CreditScore, HasJob]
test_samples = np.array([
    [40, 680, 1],  # Likely Approved
    [30, 590, 0]   # Likely Rejected
])


predictions = [predict(x) for x in test_samples]
print("Predicted Loan Status:", predictions)
```



```
Predicted Loan Status: [1, 0]

=== Code Execution Successful ===
```

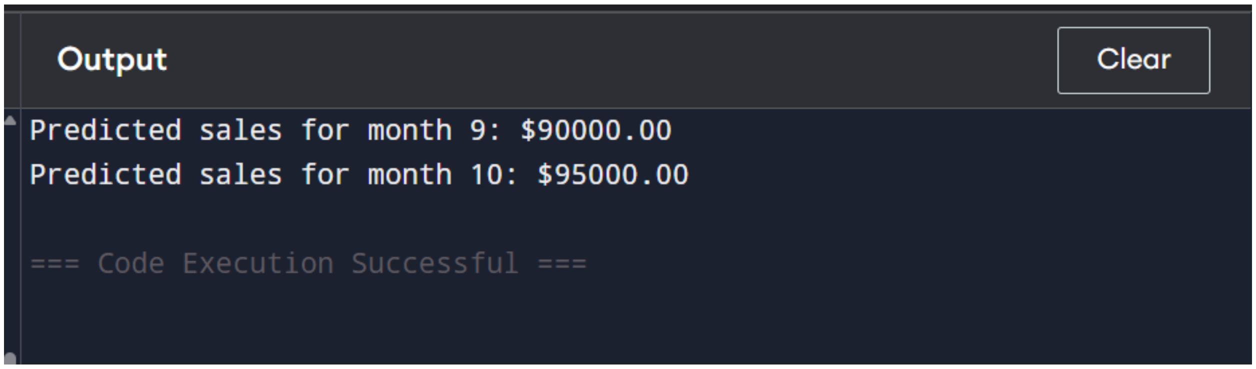
Exp-20

Implement Future Sales Prediction using a suitable machine learning algorithm

Program

```
import numpy as np

# Sample dataset: [MonthNumber], Sales in $1000
X = np.array([[1],[2],[3],[4],[5],[6],[7],[8]], dtype=float)
y = np.array([50, 55, 60, 65, 70, 75, 80, 85], dtype=float)  # Sales in $1000

# Add bias term for intercept
X_b = np.c_[np.ones((X.shape[0],1)), X]

# Compute Linear Regression weights using pseudo-inverse
w = np.linalg.pinv(X_b) @ y

# Predict future sales for months 9 and 10
future_months = np.array([[1,9],[1,10]], dtype=float)  # include bias column
pred_sales = future_months @ w
```

```
for month, sale in zip([9,10], pred_sales):
    print(f"Predicted sales for month {month}: ${sale*1000:.2f}")
```



```
Predicted sales for month 9: $90000.00
Predicted sales for month 10: $95000.00

=== Code Execution Successful ===
```