Python Programs

1) Matrix Multiplication:

```
def matrix_multiply(A, B):
    n, m = len(A), len(B[0])
    p = len(B)
    result = [[0] * m for _ in range(n)]
    for i in range(n):
        for j in range(m):
            for k in range(p):
                result[i][j] += A[i][k] * B[k][j]
    return result
A = [ [9, 5],
      [3, 4] ]
B = [ [3, 6]
      [7, 9] ]
result = matrix_multiply(A, B)
print("\nResult of AxB : ")
for row in result:
    print(row)
```

output: Result of AxB :
[62, 99]
[37, 54]

2) Simple calculator:

```
def add(x, y):
    return x+y
def subtract(x, y):
    return x-y
def multiply(x, y):
    return x*y
```

```
def divide(x, y):
    if y==0:
        return "Cannot divide by zero"
    return x/y
print("Select operation:")
print("1. Add")
print("2. Subtract")
print("3. Multiply")
print("4. Divide")
choice = input("Enter choice (1/2/3/4): ")
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
if choice == '1':
    print(f"{num1} + {num2} = {add(num1, num2)}")
elif choice == '2':
    print(f"{num1} - {num2} = {subtract(num1, num2)}")
elif choice == '3':
    print(f"{num1} * {num2} = {multiply(num1, num2)}")
elif choice == '4':
    print(f"{num1} / {num2} = {divide(num1, num2)}")
else:
    print("Invalid input")
```

output : Select operation:

1. Add
2. Subtract
3. Multiply
4. Divide

Enter choice (1/2/3/4): 3

Enter first number: 2

Enter second number: 3

2.0 * 3.0 = 6.0

3) Counting the no. of occurences of an element in a list.

```
def count(lst, x):
    count = 0
    for ele in lst:
        if (ele == x):
            count = count + 1
    return count
lst = [8, 6, 8, 10, 8, 20, 10, 8, 8]
x = 8
print('{} has occured {} times'.format(x, count(lst, x)))
```

output : 8 has occured 5 times

4) Multiplication of two lists

```
def multiplylist(mylist):
    result = 1
    for x in mylist:
        result = result * x
    return result
list1 = [1, 2, 7]
list2 = [5, 1, 4]
print(multiplylist(list1))
print(multiplylist(list2))
```

output : 14
20

5) Second largest element in list

```
list1 = [10, 20, 202, 45, 45, 45, 99, 99]
list2 = list(set(list1))
list2.sort()
print("Second largest element is:", list2[-2])
```

output : second largest element is : 99

6) concatenating lists

```
list3 = [1, 5, 5, 7, 5]
list4 = [0, 5, 7, 2, 5]
list5 = list3 + list4
print("Concatenated list using + :" + str(list3))
```

output : concatenated list using + : [1, 5, 5, 7, 5, 0, 5, 7, 2, 5]

7) Deleting item in list

```
lst = ['abc', 'def', 'ghi', 'jkl', 'mno', 'pqr']
lst.remove('def')
print("After deleting the item :", lst)
```

output : After deleting the item : ['abc', 'ghi', 'jkl', 'mno', 'pqr']

8) Odd frequency characters

```
test_str = 'Hello World, how'
x = set(test_str)
res = []
for i in x:
```

```
    if (test_str.count(i) % 2 != 0):
        res.append(i)
print("The odd frequency characters are : " + str(res))
```

output : The odd frequency characters are : ['h', 'a', 'o', 'w', ' ', 'e', 'd', 'l', 'H', 'W', '!']

9) Average of array numbers

```
def find_avg(numbers):
    return sum(numbers) / len(numbers)
numbers = [10, 20, 30, 40, 50]
average = find_average(numbers)
print("Average of array elements : " + average)
```

output : Average of array elements : 30.0

10) Harshad numbers

```
def checkHarshad(n):
    sum = 0
    temp = n
while temp > 0:
    sum = sum + temp % 10
    temp = temp // 10
return n % sum == 0
if (checkHarshad(10)):
    print("Yes")
else:
    print("No")
if (checkHarshad(15)):
    print("Yes")
else:
    print("No")
```

output : Yes
No

MINI PROJECT:

TITLE : GOLD PRICE PREDICTION USING MACHINE LEARNING WITH PYTHON.

PROBLEM STATEMENT :

Gold prices are a key economic indicator that affects traders, investors and businesses across multiple areas. Fluctuations in the price of gold can influence decision making and financial objectives. Predicting gold prices is difficult due to the number of factors such as market trends, investor sentiments, etc.

TARGET AUDIENCE:

1. Investors : Individuals looking to invest in gold seek for predictions to make detailed decisions.

2. Jewellery buyers : consumers who purchase gold for personal / industrial purpose may use predictions to manage costs.

3. Financial Advisors : Professionals who provide investment advices to clients use predictions to guide their recommendations.

4. Traders : Retail traders who buy and sell gold in various markets can rely on prediction to time their trades.

5. Financial News media: journalists and media outlets could use this prediction system for reporting on gold markets.

OBJECTIVES:

1. User Experience: The interface provides clear and actionable insights like visualizati-ons to help users make decisions quickly.

2. Accuracy: Refining the model to maintain high accuracy and reliable under different conditions.

3. Efficiency: Use efficient techniques to handle large datasets and reduce time required for training and predictions.

ABSTRACT:

This project uses machine learning to predict gold prices. This process includes applying algorithms like random forest and KNN to make accurate predictions. The result is a user-friendly tool that provides reliable predictions and visualization to help users make informed decisions.

DATASET:

1. Yahoo Finance Gold Data
2. Geeksforgeeks
3. World Gold council's data.

ALGORITHMS USED:

1. Random Forest : This algorithm is an ensemble learning method that combines multiple decision trees to improve prediction accuracy. It works by training on random subsets of data and aggregating their predictions.

2. KNN (K-Nearest Neighbours) : simple, instance based ML method used for classification. for classification it assigns the most common class among the neighbours, while for regression it averages the value of neighbours.

6.9.24

Exercise : N-Queens Problem.

AIM : To implement N-Queens Problem using python.

Code :

```
def print_board(board):
    for row in board:
        print(" ".join("Q" if col else "." for col in row))
    print()
def is_safe(board, row, col, N):
    for i in range(row):
        if board[i][col]:
            return False
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j]:
            return False
    for i, j in zip(range(row, -1, -1), range(col, N)):
        if board[i][j]:
            return False
    return True
def solve_n_queens_util(board, row, N):
    if row >= N:
        return True
    for col in range(N):
        if is_safe(board, row, col, N):
            board[row][col] = True
            if solve_n_queens_util(board, row+1, N):
                return True
            board[row][col] = False
    return False
def solve_n_queens(N):
    board = [[False] * N for _ in range(N)]
```

```
if not solve_n_queens_util(board, 0, N):
    print("No solution exists")
else:
    print_board(board)
def main():
    try:
    N=int(input("Enter the number of queens(N): "))
    if N<=0
        print("The number of queens must be a positive
              integer")
    else
        solve_n_queens(N)
    except valueError:
        print("Invalid input. Please enter a positive
              integer")
if __name__=="__main__":
    main()
```

output:

1) Enter the number of queens (N): 8

```
Q + + + + + + +
+ + + + Q + + +
+ + + + + + + Q
+ + + + + Q + +
+ + Q + + + + +
+ + + + + + Q +
+ Q + + + + + +
+ + + Q + + + +
```

2) Enter the number of queens (N): -6
The number of queens must be a positive integer

3) Enter the number of queens (N) : 4

```
+ Q + +
+ + + Q
Q + + +
+ + Q +
```

RESULT : Thus the N-queens problem using python is executed successfully.

6.9.24

Exercise : Depth First Search.

AIM: To implement DFS in python.

code:

```
class Node:
    def __init__(self, value):
        self.value = value
        self.neighbours = []
def add_neighbour(self, neighbour):
    self.neighbours.append(neighbour)
class Graph:
    def __init__(self):
        self.nodes = {}
def add_node(self, value):
    if value not in self.nodes:
        self.nodes[value] = Node(value)
def add_edge(self, from_value, to_value):
    if from_value in self.nodes and to_value in
                        self.nodes:
    self.nodes[from_value].add_neighbor(self.nodes
                                [to_value])
    self.nodes[to_value].add_neighbor(self.nodes
                                [from_value])
def dfs(self, start_value):
    visited = set()
    stack = [self.nodes.get(start_value)]
while stack:
    node = stack.pop()
    if node and node.value not in visited:
        print(node.value)
        visited.add(node.value)
    for neighbor in reversed(node.neighbors):
```

```
        if neighbor.value not in visited:
            stack.append(neighbor)
graph = Graph()
graph.add_node('A')
graph.add_node('B')
graph.add_node('C')
graph.add_node('D')
graph.add_node('E')
graph.add_node('F')

graph.add_edge('A', 'B')
               ('A', 'C')
               ('B', 'D')
               ('B', 'E')
               ('C', 'F')
               ('E', 'F')

print("DFS output:")
graph.dfs('B')
```

output: DFS output:

B
A
C
F
E
D

RESULT: Thus the depth first search using python is executed successfully.

13.09.24

Exercise: A* Search Algorithm

AIM: To implement A* search algorithm using python

code:

```
def astar(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
g = {}
parents = {}
g[start_node] = 0
parents[start_node] = start_node
while len(open_set) > 0:
    n = None
    for v in open_set:
        if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
            n = v
        if n == stop_node or Graph_nodes[n] == None:
            pass
    else:
        for (m, weight) in get_neighbours(n):
            if m not in open_set and m not in closed_set:
                open_set.add(m)
                parents[m] = n
                g[m] = g[n] + weight
            else:
                if g[m] > g[n] + weight:
                    g[m] = g[n] + weight
                    parents[m] = n
                if m in closed_set:
                    closed_set.remove(m)
                    open_set.add(m)
```

```
if n== None :
    print ("path does not exist!")
    return None
if n==stop_node :
  path=[]
  while parents[n] != n :
    path.append(n)
    n=parents[n]
path.append(start_node)
path.reverse()
print("Path : {}".format(path))
return
open_set.remove(n)
closed_set.add(n)
print("path does not exist")
return None
def get_neighbours(v):
  if v in graph_nodes:
  return graph_nodes[v]
else:
  return None
def heuristic(n):
  H_dist = {
    'A' : 11,
    'B' : 6,
    'C' : 99,
    'D' : 1,
    'E' : 7,
    'G' : 0
}
return H_dist[n]
Graph_nodes = {
    'A' : [('B', 2), ('E', 3)],
    'B' : [('C', 1), ('G', 9)],
```

```
'C': None,
'E': [('D', 6)],
'D': [('G', 1)],
}
astar('A', 'G')
```

output:

Path: ['A', 'E', 'D', 'G']

RESULT: Thus the A* search algorithm using python is executed successfully.

13-09-24

Exercise: Water-Jug problem using DFS.

AIM: To implement Water-Jug problem using DFS in python.

code:

```
def water_jug_problem_dfs(jug1_cap, jug2_cap, target_amount):
    j1=0
    j2=0
    actions = [("fill",1), ("fill",2), ("empty",1), ("empty",2),
               ("pour",1,2), ("pour",2,1)]
    visited = set()
    stack = [(j1, j2, [])]
    while stack:
        j1, j2, seq = stack.pop()
        if (j1, j2) not in visited:
            visited.add((j1,j2))
            if j1 == target_amount or j2 == target_amount:
                return seq
            for action in actions:
                if action[0] == "fill":
                    if action[1] == 1:
                        next_state = (jug1_cap, j2)
                    else
                        next_state = (j1, jug2_cap)
                elif action[0] == "empty":
                    if action[1] == 1:
                        next_state = (0, j2)
                    else:
                        next_state = (j1, 0)
                else:
                    if action[1] == 1:
```

```
        amount = min(j1, jug2_cap - j2)
        next_state = (j1 - amount, j2 + amount)
    else:
        amount = min(j2, jug1_cap - j1)
        next_state = (j1 + amount, j2 - amount)
    if next_state not in visited:
        next_seq = seq + [action]
        stack.append((next_state[0], next_state[1],
                      next_seq))
return None
result = water_jug_problem_dfs(5, 6, 3)
print(result)
```

output:

```
[('fill', 2), ('pour', 2, 1), ('empty', 2), ('pour', 1, 2),
('fill', 1), ('pour', 1, 2), ('fill', 1), ('pour', 1, 2),
('empty', 2), ('pour', 1, 2), ('fill', 1), ('pour', 1,
2)]
```

RESULT: Thus the water jug problem using dfs in python is executed successfully.

Exercise : Decision tree classification technique.

AIM: To implement decision tree classification technique using python.

code:

```
import pandas as pd
import numpy as np
data = {
    'Height': [5.1, 5.5, 5.7, 5.3, 6.0, 5.8, 5.4, 6.2],
    'Weight': [100, 150, 130, 120, 180, 170, 140, 200],
    'Gender': ['Female', 'Male', 'Male', 'Female', 'Male',
        'Male', 'Female', 'Male']
}
df = pd.DataFrame(data)
print(df)

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df
    ['Gender'])
x = df['Height', 'Weight']
y = df['Gender']
x_train, x_test, y_train, y_test = train_test_split(x,
    y, test_size = 0.2, random_state = 42)

from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(x_train, y_train)

from sklearn.metrics import accuracy_score,
    classification_report
```

```
y_pred = classifier.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)

from sklearn.tree import export_graphviz
import graphviz
dot_data = export_graphviz(classifier, out_file=
                None, feature_names=['Height',
                'Weight'], class_names=label_encoder.
                classes_, filled=True, rounded=
                True, special_characters=True)
graph = graphviz.Source(dot_data)
graph.render("genders_classification_tree")
graph.view()
```

output:

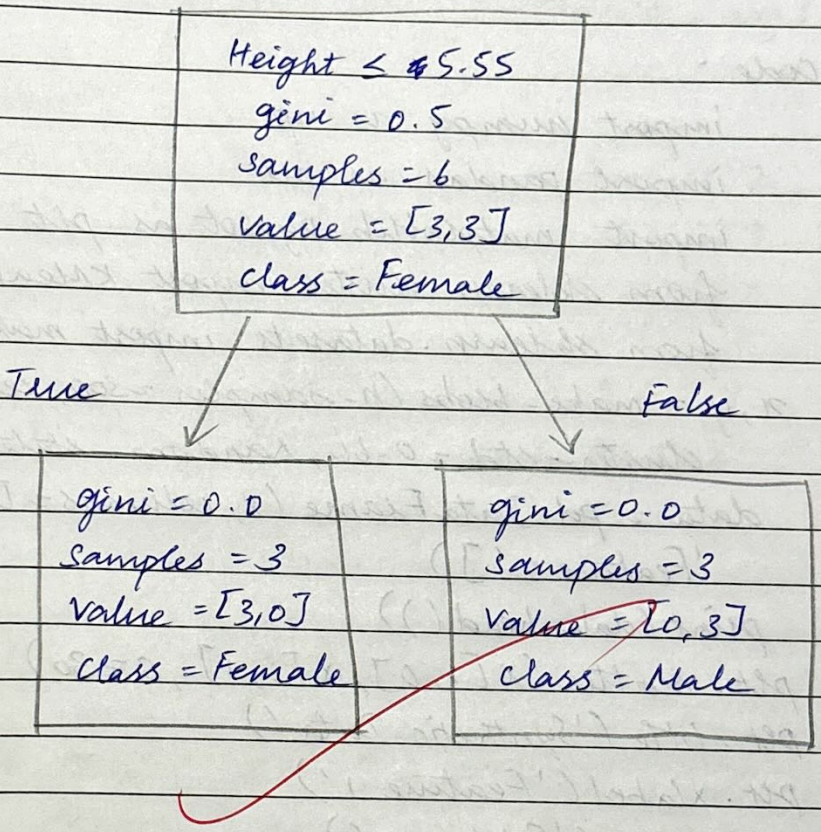
	Height	Weight	Gender
0	5.1	100	Female
1	5.5	150	Male
2	5.7	130	Male
3	5.3	120	Female
4	6.0	180	Male
5	5.8	170	Male
6	5.4	140	Female
7	6.2	200	Male

Accuracy = 0.50

Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	0
1	1.00	0.50	0.67	2

accuracy			0.50	2
macro avg	0.50	0.25	0.33	2
weighted avg	1.00	0.50	0.67	2



RESULT : Thus to implement decision tree classification technique using python is executed & verified.

Exercise : K-Means clustering technique

AIM : To implement K-Means clustering technique using python.

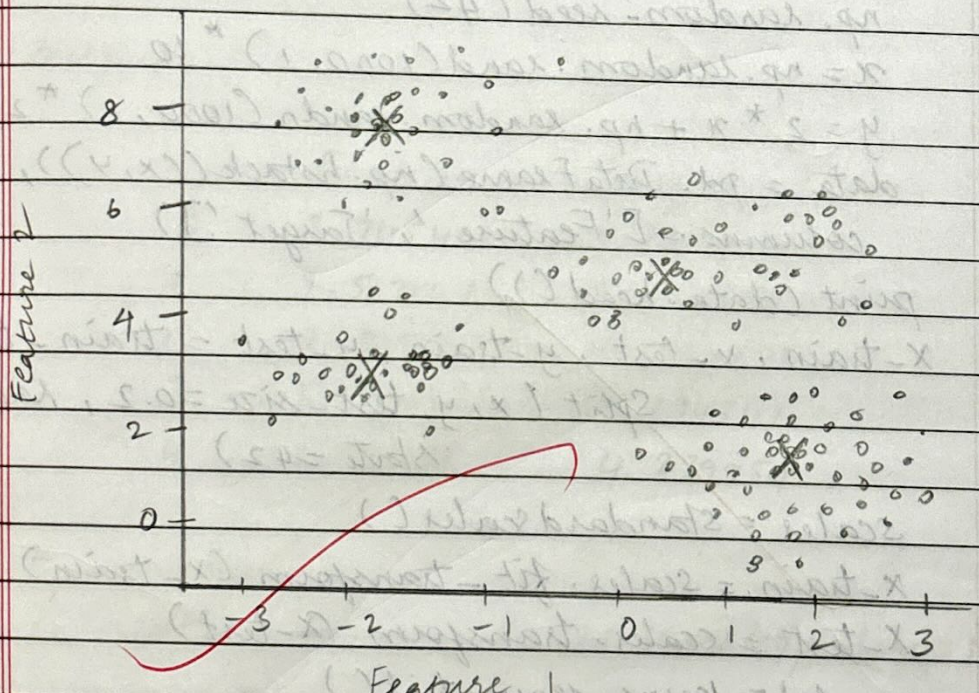
Code :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
x, y = make_blobs(n_samples=300, centers=4,
    cluster_std=0.60, random_state=0)
data = pd.DataFrame(x, columns=['Feature 1',
    'Feature 2'])
print(data.head())
plt.scatter(x[:,0], x[:,1], s=30)
plt.title('Synthetic Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
  k=4
kmeans = Kmeans(n_clusters=k)
kmeans.fit(x)
labels = kmeans.labels_
centers = kmeans.cluster_centers_
plt.scatter(x[:,0], x[:,1], c=labels, s=30, cmap:
  'viridis')
plt.scatter(centers[:,0], centers[:,1], c='red',
  s=200, alpha=0.75, marker='x')
plt.title('K-means clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

```
inertia = kmeans.inertia_
print(f'Inertia : {inertia}')
from sklearn.metrics import silhouette_score
silhouette_avg = silhouette_score(X, labels)
print(f'Silhouette Score : {silhouette_avg}')
```

output :

	Feature 1	Feature 2
0	0.836857	2.136359
1	-1.413658	7.409623
2	1.155213	5.099619
3	-1.018616	7.814915
4	1.271351	1.892542



Inertia : 212.00599621083475

Silhouette Score : 0.6819938690643478

RESULT : Thus the implementation of K-Means clustering technique using python is executed & verified successfully.

Exercise : Artificial Neural Networks

AIM: To implement ANN for an application in Regression using python.

Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras import layers
np.random.seed(42)
x = np.random.rand(1000, 1) * 10
y = 2 * x + np.random.randn(1000, 1) * 2
data = pd.DataFrame(np.hstack((x, y)),
  columns = ['Feature', 'Target'])
print(data.head())
x_train, x_test, y_train, y_test = train_test_
                    split(x, y, test_size=0.2, random_
                          state=42)
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
model = keras.sequential()
model.add(layers.dense(64, activation='relu',
input_shape = (x_train.shape[1],)))
model.add(layers.dense(32, activation='relu'))
model.add(layers.Dense(1))
model.compile(optimizer='adam', loss='mean_
    squared_error')
```

```
history = model.fit(X_train, y_train, epochs=100,
    batch_size=32, validation_split=0.2)
test_loss = model.evaluate(X_test, y_test)
print(f'Test loss (MSE): {test_loss}')
y_pred = model.predict(X_test)
plt.scatter(X_test, y_test, color='blue', label=
    'True Values')
plt.scatter(X_test, y_pred, color='red', label=
    'Predicted values')
plt.title('True vs Predicted values')
plt.xlabel('Feature')
plt.ylabel('Feature')
plt.legend()
plt.show()
```

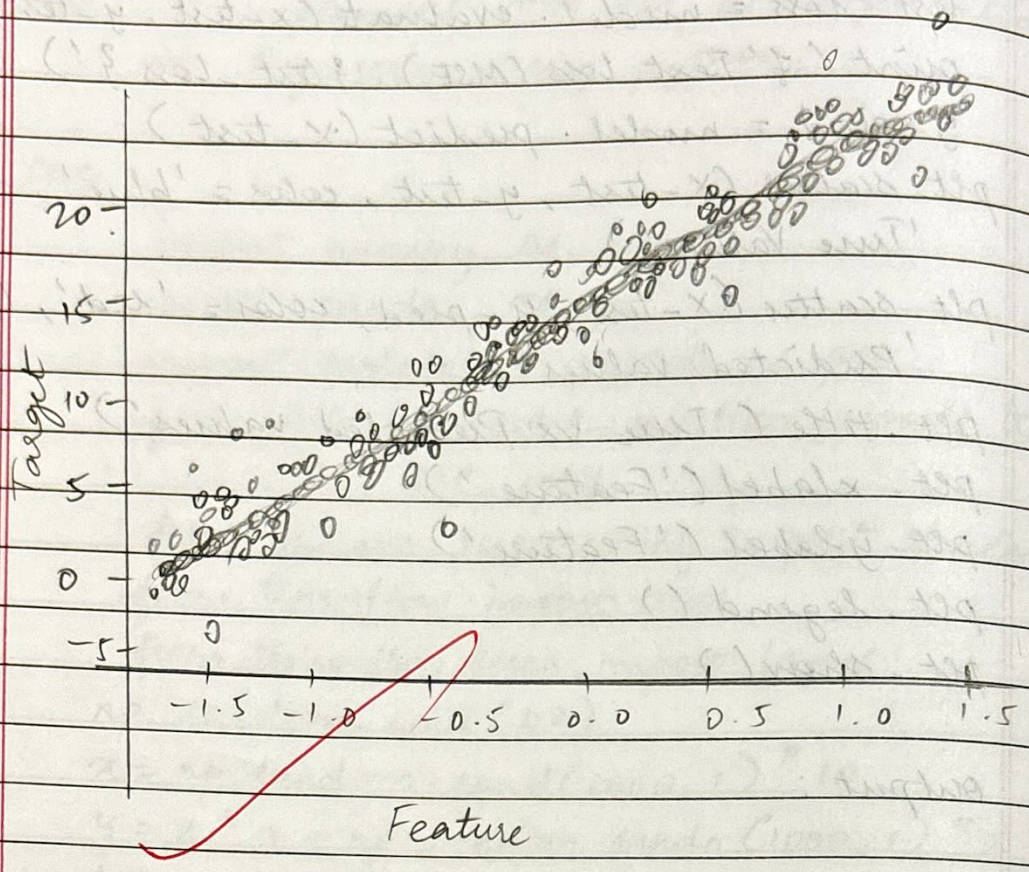
output :

	Feature	Target
0	3.745401	7.846204
1	9.507143	16.343597
2	7.319939	15.400275
3	5.986585	13.194341
4	1.560186	4.239954

```
Epoch 1/100
20/20 ———————— 2s 10ms/step - loss: 143.6730
:
:
:
Epoch 100/100
20/20 ———————— 0s 4ms/step - loss: 4.2248
```

7/7 ———— 0s 4ms/step - loss: 3.9068

Test Loss (MSE): 3.5400872230529785



RESULT : Thus the implementation of ANN for an application in regression using python is executed & verified successfully.

8.11.24

Exercise - Introduction to prolog

AIM: To learn PROLOG terminologies and write basic programs.

TERMINOLOGIES:

1) Atomic Terms: Atomic Terms are usually terms made up of lower and uppercase letters, digits and the underscore, starting with a lower case letter.

eg: dog
ab-c-321

2) Variables: Variables are strings of letters, digits and the underscore, starting with a capital letter or an underscore.

eg: Dog
Apple-420

3) Compound Terms: They are made up of a PROLOG atom and a number of arguments (PROLOG terms i.e, atoms, numbers, variables or other compound items) enclosed in parenthesis & separated by commas.

eg: is-bigger (elephant, X)
f(g(X, _), 7)

4) Facts: A fact is a predicate followed by dot.

eg: bigger-animal (whale)
life-is-beautiful.

5) Rules: A rule consists of a head (a predicate) and a body (a sequence of predicates separated by

commas).

eg: is_smaller(X, Y) :- is_bigger(Y, X).
aunt(Aunt, Child) :- sister(Aunt, Parent), parent(Parent, Child)

SOURCE CODE:

KB1:

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
```

Query 1: ?- woman(mia).
Query 2: ?- playsAirGuitar(mia).
Query 3: ?- party.
Query 4: ?- concert.

OUTPUT:

```
?- woman(mia).
true

?- playsAirGuitar(mia).
false

?- party.
true

?- concert.
ERROR: Unknown procedure: concert/0 (DWIM could not correct goal)
```

KB2:

```
happy(yolanda).
listens2music(mia).
Listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(Yolanda):- listens2music(yolanda).
```

OUTPUT:

```
?- playsAirGuitar(mia).
true

?- playsAirGuitar(yolanda).
true
```

KB3:

```
likes(dan, sally).
likes(sally, dan).
likes(john, brittney).
married(X,Y):- likes(X,Y), likes(Y,X).
friends(X,Y):- likes(X,Y); likes(Y,X).
```

OUTPUT:

```
?- likes(dan, X)
X = sally

?- married(dan, sally).
true

?- married(john, brittany)
false
```

KB4:

```
food(burger).
food(sandwhich).
food(pizza).
lunch(sandwhich).
dinner(pizza).
meal(X) :- food(X).
```

OUTPUT:

```
?- food(pizza).
true

?- meal(X), lunch(X)
X = sandwhich

?- dinner(sandwhich)
false
```

KB5:

```
owns(jack, car(bmw)).
owns(john, car(chevy)).
owns(olivia, car(civic)).
owns(jane, car(chevy)).
sedan(car(bmw)).
sedan(car(civic)).
truck(car(chevy)).
```

OUTPUT:

```
?- owns(john, X).
   X = car(chevy).
```

?- owns(john, _).
true

?- owns(who, car(cheryl)).
who = john.

?- owns(jane, X), sedan(X).
false

RESULT: Thus to learn PROLOG terminologies & write basic programs is executed successfully.