1. DFS and BFS

```
from collections import deque

g = {1:[2,3], 2:[4], 3:[4], 4:[5]}

def dfs(u, goal, v=[], p=[]):
   v.append(u); p.append(u)
   if u==goal: return p
   for x in g.get(u,[]):
      if x not in v:
         r=dfs(x,goal,v,p)
         if r: return r
   p.pop(); return None

def bfs(s, goal):
   q=deque([(s,[s])]); v={s}
   while q:
      n,p=q.popleft()
      if n==goal: return p
      for x in g.get(n,[]):
         if x not in v: v.add(x); q.append((x,p+[x]))


print("DFS:", dfs(1,5))
print("BFS:", bfs(1,5))
```

OUTPUT:

```
DFS: [1, 2, 4, 5]
BFS: [1, 2, 4, 5]
```

2. Hill Climbing:

```
import numpy as np

def f(x): return -x**2 + 5

def neigh(x, s=0.1): return [x+s, x-s]

x = 2.0

for i in range(100):

    n = neigh(x)

    best = max(n, key=f)

    if f(best) > f(x):

        x = best

        print(f"Step {i+1}: x={x:.4f}, f(x)={f(x):.4f}")

    else:

        print("Converged"); break

print(f"\nBest x={x:.4f}, f(x)={f(x):.4f}")
```

OUTPUT:

```
Step 1: x=1.9000, f(x)=1.3900

Step 2: x=1.8000, f(x)=1.7600

Step 3: x=1.7000, f(x)=2.1100

Step 4: x=1.6000, f(x)=2.4400

Step 5: x=1.5000, f(x)=2.7500

Step 6: x=1.4000, f(x)=3.0400

Step 7: x=1.3000, f(x)=3.3100

Step 8: x=1.2000, f(x)=3.5600

Step 9: x=1.1000, f(x)=3.7900

Step 10: x=1.0000, f(x)=4.0000

Step 11: x=0.9000, f(x)=4.1900

Step 12: x=0.8000, f(x)=4.3600

Step 13: x=0.7000, f(x)=4.5100

Step 14: x=0.6000, f(x)=4.6400

Step 15: x=0.5000, f(x)=4.7500
Step 16: x=0.4000, f(x)=4.8400
Step 17: x=0.3000, f(x)=4.9100
Step 18: x=0.2000, f(x)=4.9600
Step 19: x=0.1000, f(x)=4.9900
Step 20: x=-0.0000, f(x)=5.0000
Converged
Best x=-0.0000, f(x)=5.0000
```

3. Knowledge Based Agent for Wumpus world

```
class KB:
    def __init__(s): s.f=set()
    def tell(s,f): s.f.add(f)

class WumpusAgent:
    def __init__(s):
        s.kb=KB(); s.gold=False
    def run(s):
        world={(0,0):["Breeze"],(0,1):["Stench"],(1,0):[],(1,1):["Breeze"]}
        for loc, percepts in world.items():
            for p in percepts: s.kb.tell((p,loc))
            if "Glitter" in percepts: s.gold=True
        print("Gold found" if s.gold else "Gold not found")
WumpusAgent().run()
```

OUTPUT:

```
Gold not found
```

4. Knowledge Representation schemes

```
facts = {"raining","cloudy"}

rules = [ (["raining"],"wet_ground"), (["cloudy"],"maybe_rain"), (["raining","cloudy"],"umbrella_needed") ]

changed = True

while changed:

    changed = False

    for pre, con in rules:

        if con not in facts and all(p in facts for p in pre):

            facts.add(con)

            changed = True


print("All facts:", facts)
```

OUTPUT:

```
All facts: {'cloudy', 'wet_ground', 'raining', 'umbrella_needed', 'maybe_rain'}
```

5. Forward state space planning alg

```
from collections import deque

class A:
   def __init__(s,n,p,a,d=[]): s.n,p,s.a,s.d=n,set(p),set(a),set(d)
   def apply(s,state): return (state - s.d) | s.a

def plan(init,acts,goal):
   q,vis=deque([(init,[])]),{frozenset(init)}
   while q:
      s,p=q.popleft()
      if goal<=s: return p
      for a in acts:
         if a.n<=s:
            ns=frozenset(a.apply(s))
            if ns not in vis: vis.add(ns); q.append((ns,p+[list(a.a)]))
   return None

init=frozenset({"onA_table","clearA","onB_table","clearB"})
acts=[A(frozenset(["onA_table","clearA"]),["holdingA"],["onA_table","clearA"]),
     A(frozenset(["holdingA","clearB"]),["onA_B"],["holdingA","clearB"])]
goal=frozenset(["onA_B"])
print("Plan:", plan(init,acts,goal))
```

OUTPUT:

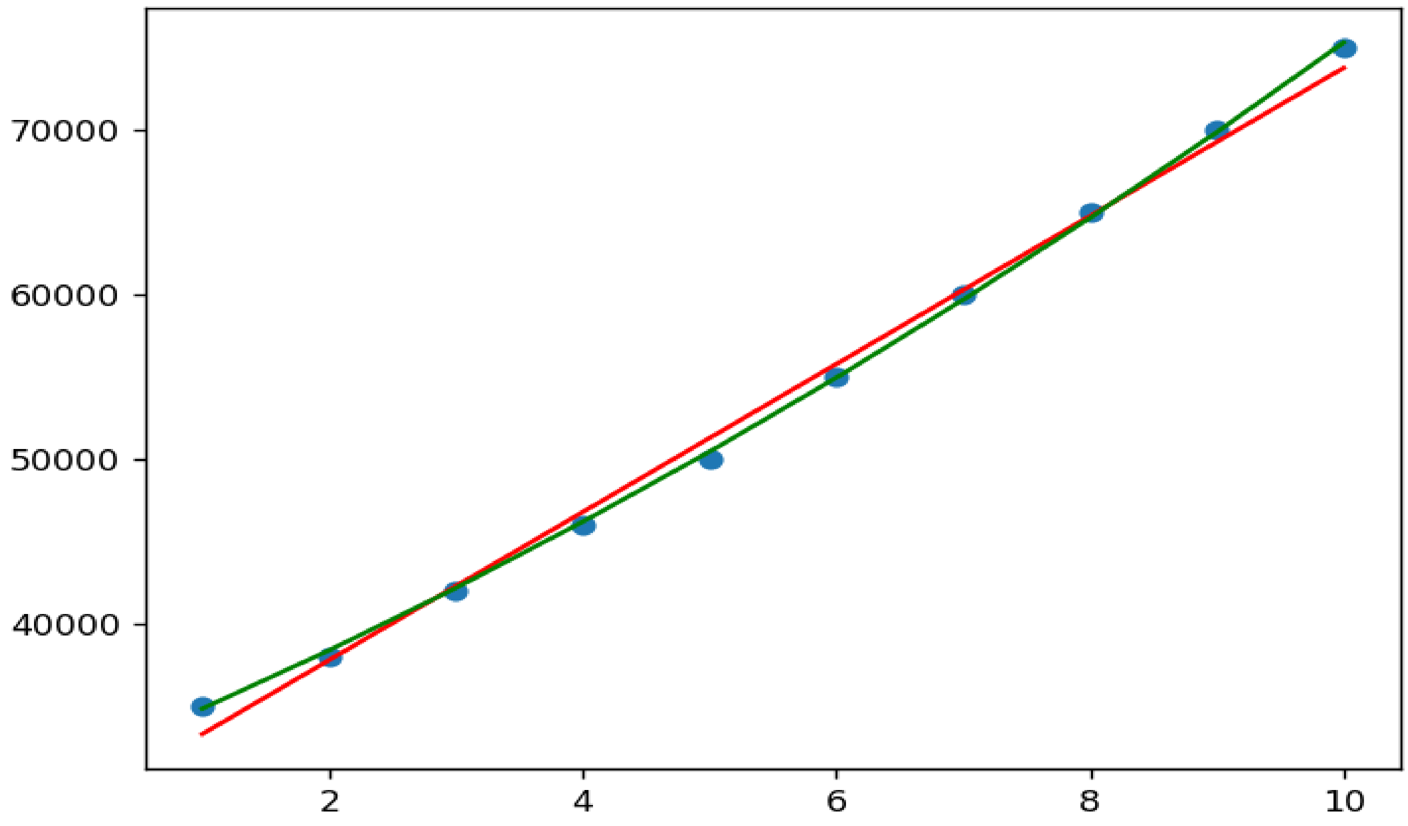
```
Plan: None
```

6. Simple and polynomial Regression

```
import numpy as np, matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score

X,y=np.arange(1,11).reshape(-
1,1),np.array([35000,38000,42000,46000,50000,55000,60000,65000,70000,75000])
Xtr,Xte,ytr,yte=train_test_split(X,y,test_size=0.2,random_state=42)

lr=LinearRegression().fit(Xtr,ytr)
poly=PolynomialFeatures(2);
pr=LinearRegression().fit(poly.fit_transform(Xtr),ytr)

print("R² Linear:", r2_score(yte,lr.predict(Xte)))
print("R² Poly:", r2_score(yte,pr.predict(poly.transform(Xte))))

plt.scatter(X,y)
plt.plot(X,lr.predict(X),'r',X,pr.predict(poly.transform(X)),'g');
plt.show()
```

OUTPUT:

```
R² Linear: 0.998779296875

R² Poly: 0.9997211508583043
```

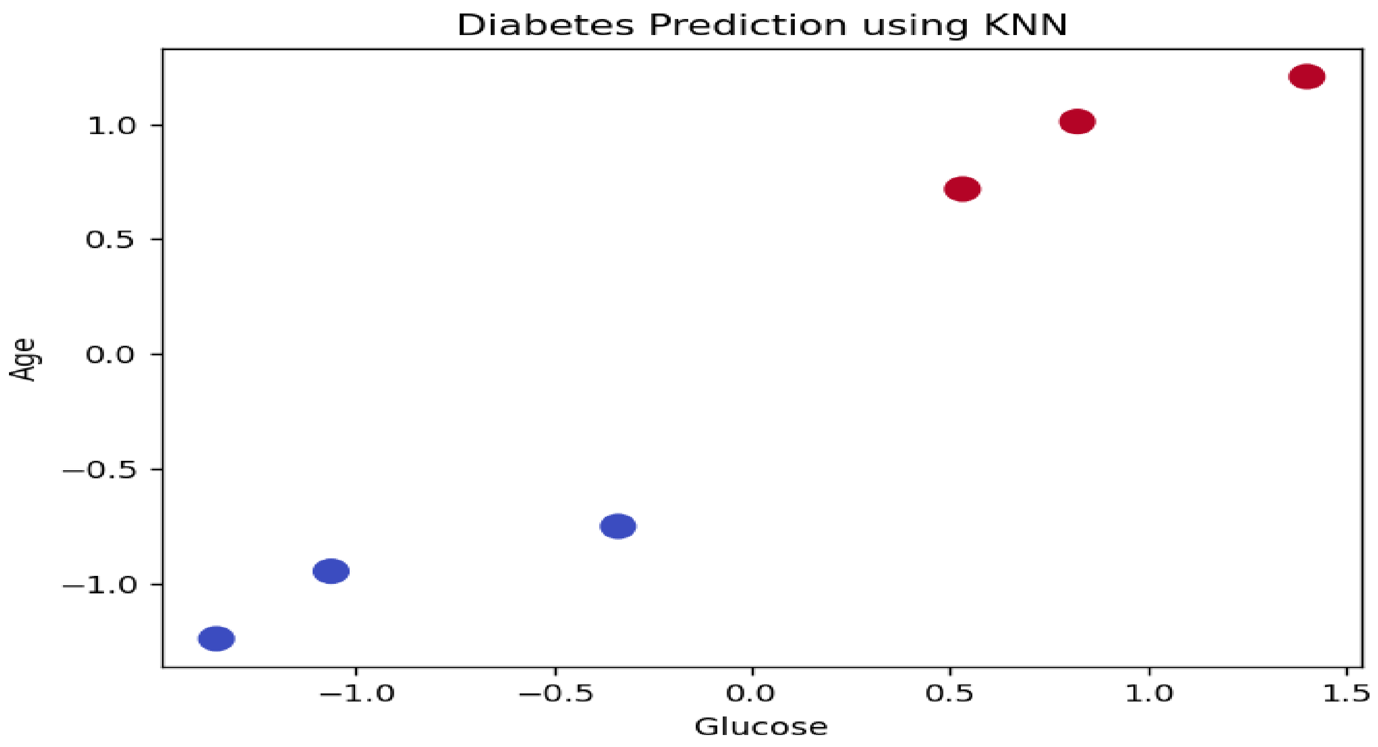


7. KNN On Indian diabetes patient

```
import numpy as np,

matplotlib.pyplot as plt


X=np.array([[2,120,70,20,80,25,0.5,30],[4,150,80,25,90,30,0.7,45],

           [1,85,60,18,60,22,0.3,25],[6,180,90,30,120,35,0.9,50],

           [3,95,65,20,70,23,0.4,28],[5,160,85,28,100,33,0.8,48]])

y=np.array([0,1,0,1,0,1])


X=(X-X.mean(0))/X.std(0)

def knn(X,y,q,k=3):

d=np.sqrt(((X-q)**2).sum(1)); r

eturn int(round(y[np.argsort(d)[:k]].mean()))


q=np.array([2,130,72,22,85,26,0.6,32]); q=(q-X.mean(0))/X.std(0)

print("Prediction (1=Diabetic,0=Not):", knn(X,y,q))


plt.scatter(X[:,1],X[:,7],c=y,cmap="coolwarm",s=100)

plt.xlabel("Glucose"); plt.ylabel("Age");

 plt.title("Diabetes Prediction using KNN");

plt.show()
```

OUTPUT:

```
Prediction (1=Diabetic,0=Not): 1
```



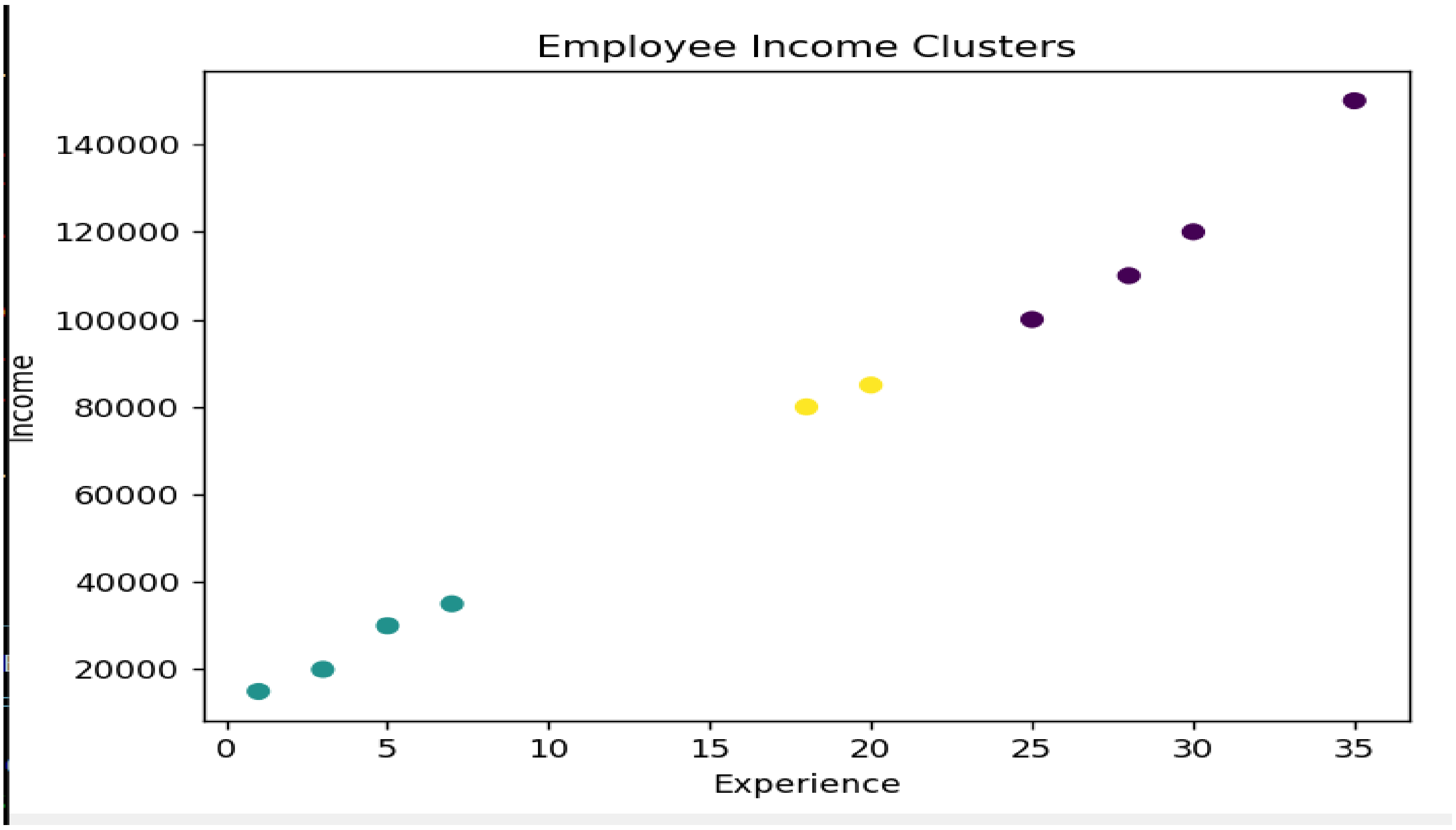
8. K-means Clustering mode

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

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler


df=pd.DataFrame({'Age':[22,25,47,52,46,56,55,60,28,30],

            'Experience':[1,3,20,25,18,30,28,35,5,7],

            'Income':[15000,20000,85000,100000,80000,120000,110000,150000,30000,35000]})


print("Employee Dataset:\n", df)

data_scaled = StandardScaler().fit_transform(df)

df['Cluster'] = KMeans(n_clusters=3, random_state=42).fit_predict(data_scaled)


plt.scatter(df['Experience'],df['Income'],c=df['Cluster'],cmap='viridis')

plt.xlabel('Experience'); plt.ylabel('Income'); plt.title('Employee Income Clusters');

plt.show()
```

OUTPUT:

Employee Dataset:

```
   Age  Experience  Income
0   22           1   15000
1   25           3   20000
2   47          20   85000
3   52          25  100000
4   46          18   80000
5   56          30  120000
6   55          28  110000
7   60          35  150000
8   28           5   30000
9   30           7   35000
```



9. IRIS Flower

```
import numpy as np

from sklearn.datasets import load_iris

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score


X,y=load_iris(return_X_y=True)

sc=StandardScaler(); X=sc.fit_transform(X)

pca=PCA(1); X=pca.fit_transform(X)

Xtr,Xte,ytr,yte=train_test_split(X,y,test_size=0.2,random_state=42)

model=LogisticRegression().fit(Xtr,ytr)

print("Accuracy:",accuracy_score(yte,model.predict(Xte)))

new=np.array([[5.1,3.5,1.4,0.2]]); new=pca.transform(sc.transform(new))

print("Predicted Species:", load_iris().target_names[model.predict(new)][0])
```

OUTPUT:

```
Accuracy: 0.9

Predicted Species: setosa
```

10. Q- learning

```
import numpy as np, random


n_states, n_actions, goal = 5, 2, 4

Q = np.zeros((n_states, n_actions))

alpha, gamma, epsilon, episodes = 0.1, 0.9, 0.2, 500


for _ in range(episodes):

    s = 0

    while s != goal:

        a = random.randint(0,1) if random.random() < epsilon else np.argmax(Q[s])

        ns = s+1 if a==1 else max(0, s-1)

        r = 10 if ns==goal else -1

        Q[s,a] += alpha * (r + gamma * np.max(Q[ns]) - Q[s,a])

        s = ns

print("Learned Q-table:")

print(np.round(Q, 2))
```

OUTPUT:

```
Learned Q-table:

[[ 3.09  4.58]

 [ 3.11  6.2 ]

 [ 4.57  8.  ]

 [ 6.19 10.  ]

 [ 0.    0.  ]]
```