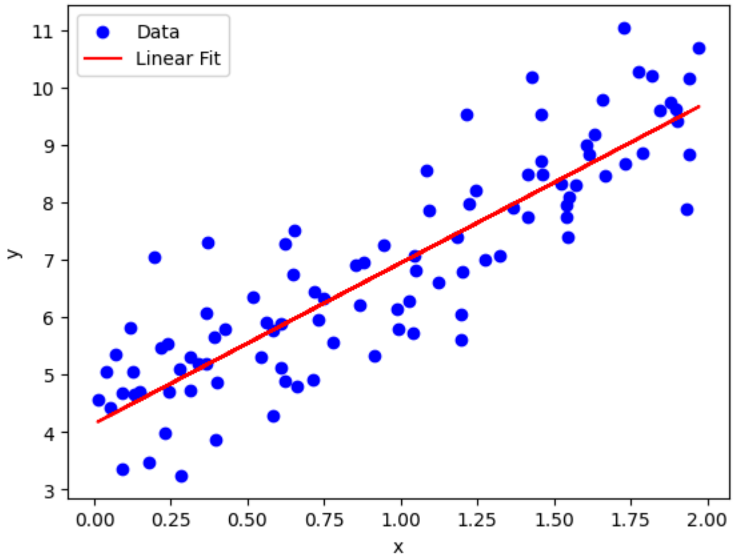
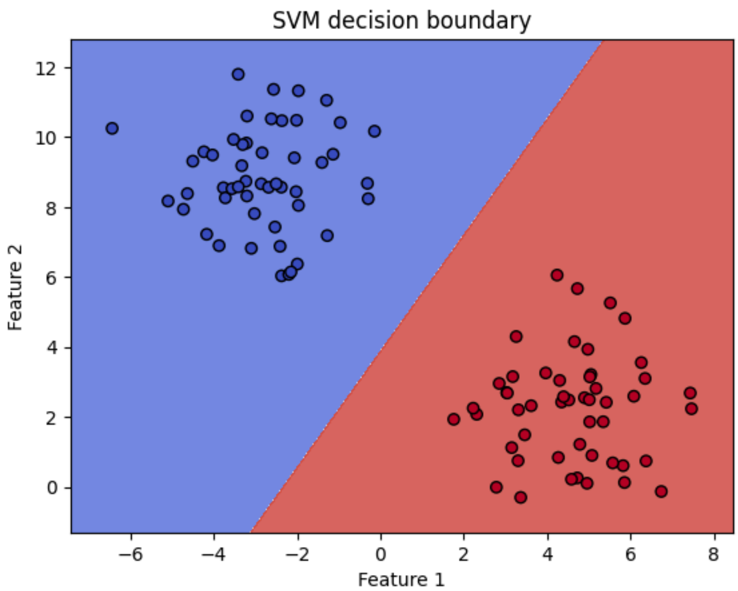
```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

np.random.seed(42)
x=2*np.random.rand(100,1)
y=4+3*x+np.random.randn(100,1)

x_train,x_test,y_train,y_test=train_test_split(
    x,y,test_size=0.2,random_state=42
)

model=LinearRegression()
model.fit(x_train,y_train)

y_pred=model.predict(x_test)

print("Slope:",model.coef_[0][0])
print("Intercept:",model.intercept_[0])

mse=mean_squared_error(y_test,y_pred)
print("MSE on test Data:",mse)

plt.scatter(x,y,color='blue',label='Data')
plt.plot(x,model.predict(x),color='red',label='Linear Fit')
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

```
Slope: 2.7993236574802762
Intercept: 4.142913319458566
MSE on test Data: 0.6536995137170021
```



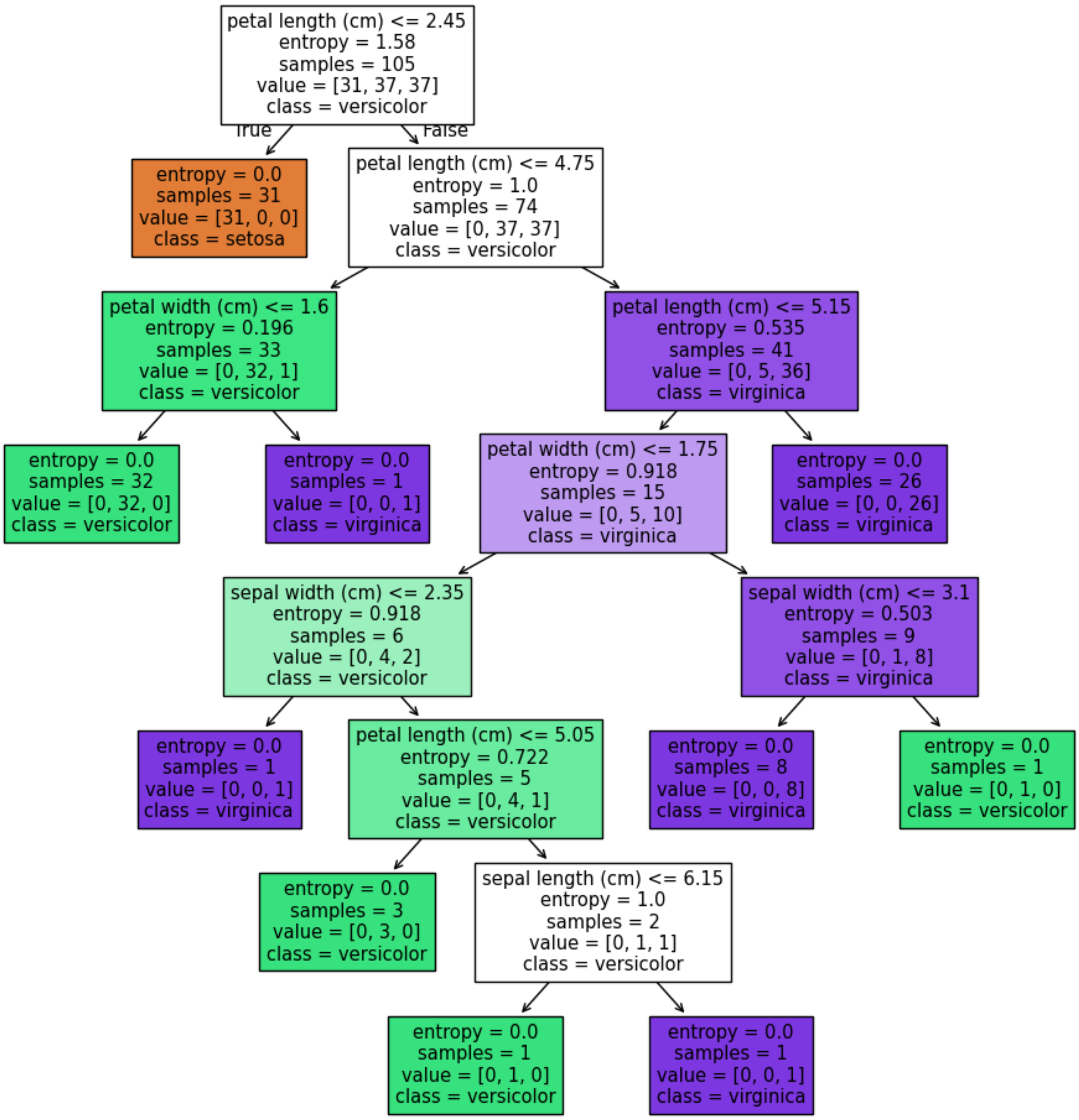
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.svm import SVC

x,y=make_blobs(n_samples=100,centers=2,random_state=42,cluster_std=1.5)
model=SVC(kernel='linear')
model.fit(x,y)

x_min,x_max=x[:,0].min()-1,x[:,0].max()+1;
y_min,y_max=x[:,1].min()-1,x[:,1].max()+1;

xx,yy=np.meshgrid(np.arange(x_min,x_max,0.01),np.arange(y_min,y_max,0.01))
z=model.predict(np.c_[xx.ravel(),yy.ravel()])
z=z.reshape(xx.shape)
plt.contourf(xx,yy,z,alpha=0.8,cmap=plt.cm.coolwarm)
plt.scatter(x[:,0],x[:,1],c=y,edgecolor='k',cmap=plt.cm.coolwarm)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("SVM decision boundary")
plt.show()
```




```
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier,export_text,plot_tree
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

data=load_iris()

x=data.data
y=data.target

x_train,x_test,y_train,y_test=train_test_split(
    x,y,test_size=0.3,random_state=42
)
id3_tree=DecisionTreeClassifier(criterion='entropy',random_state=42)
id3_tree.fit(x_train,y_train)

y_pred=id3_tree.predict(x_test)
accuracy=accuracy_score(y_test,y_pred)
print(f"accuracy : {accuracy*100:.2f}")
print('\nDecision Tree Rules:')
print(export_text(id3_tree,feature_names=data.feature_names))
plt.figure(figsize=(12,12))
plot_tree(id3_tree,feature_names=data.feature_names,class_names=list(data.target_names),filled=True)
plt.title("ID3 decision Tree")
plt.show()
```

```
accuracy : 97.78

Decision Tree Rules:
|--- petal length (cm) <= 2.45
|   |--- class: 0
|--- petal length (cm) >  2.45
|   |--- petal length (cm) <= 4.75
|   |   |--- petal width (cm) <= 1.60
|   |   |   |--- class: 1
|   |   |--- petal width (cm) >  1.60
|   |   |   |--- class: 2
|   |--- petal length (cm) >  4.75
|   |   |--- petal length (cm) <= 5.15
|   |   |   |--- petal width (cm) <= 1.75
|   |   |   |   |--- sepal width (cm) <= 2.35
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- sepal width (cm) >  2.35
|   |   |   |   |   |--- petal length (cm) <= 5.05
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- petal length (cm) >  5.05
|   |   |   |   |   |   |--- sepal length (cm) <= 6.15
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |--- sepal length (cm) >  6.15
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |--- petal width (cm) >  1.75
|   |   |   |   |--- sepal width (cm) <= 3.10
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- sepal width (cm) >  3.10
|   |   |   |   |   |--- class: 1
|   |   |--- petal length (cm) >  5.15
|   |   |   |--- class: 2
```

ID3 decision Tree



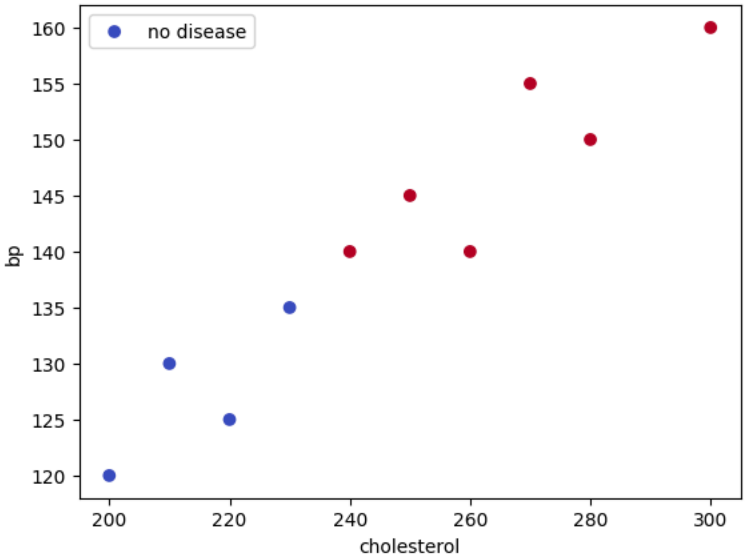
```python
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

data=np.array([
    [1200,200],
    [1500,250],
    [1700,200],
    [2100,400],
    [2300,450],
    [2500,500]
])
x=data[:,0]
y=data[:,1]

x_train,x_test,y_train,y_test=train_test_split(
    x,y,test_size=0.3,random_state=42
)

print("test_data:")
print(x_test)
print("\n train data:")
print(x_train)
print("\n")

def knn_regression(x_train,y_train,x_test,k=3):
    predictions=[]
    for test_point in x_test:
        distance=np.sqrt((x_train- test_point)**2)
        nearest_indices=np.argsort(distance)[:k]
        nearest_values=y_train[nearest_indices]
        prediction=np.mean(nearest_values)
        predictions.append(prediction)

    return np.array(predictions)


y_pred=knn_regression(x_train,y_train,x_test)
mse=mean_squared_error(y_test,y_pred)
print("mse:",mse)

for i, (size,actual,pred) in enumerate(zip(x_test, y_test,y_pred)):
    print(f"house size:{size} actual:{actual} pred={pred}")
```

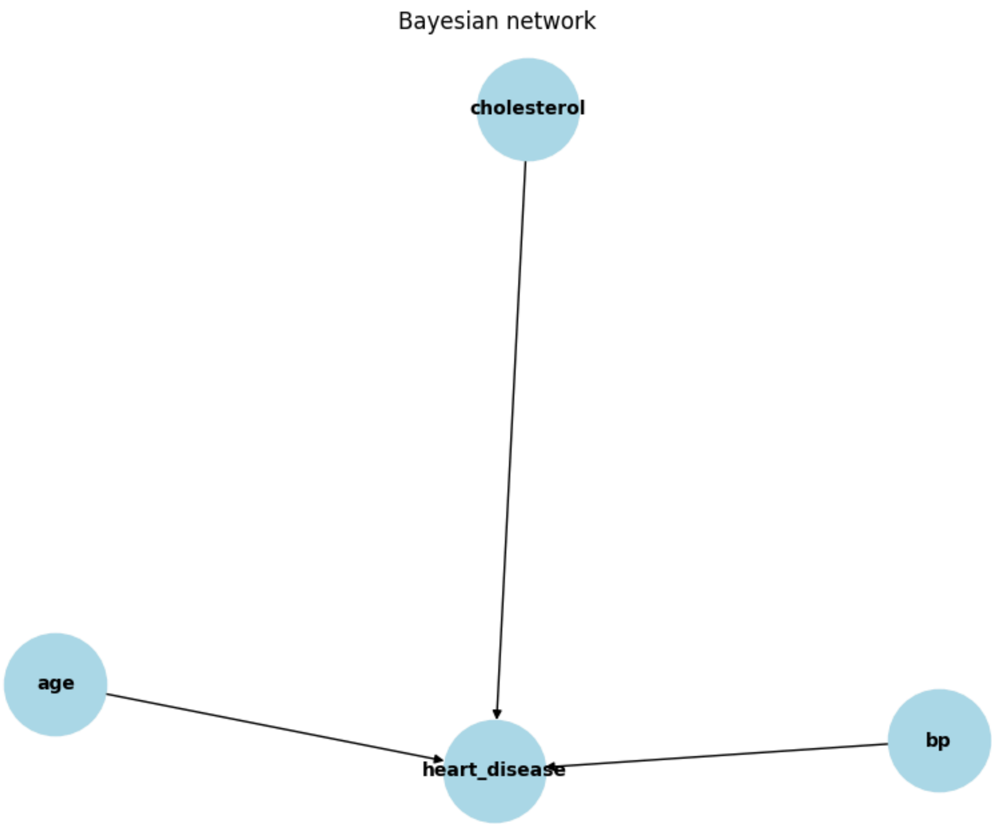
```
test_data:
[1200 1500]

 train data:
[2500 1700 2300 2100]


mse: 16250.0
house size:1200 actual:200 pred=350.0
house size:1500 actual:250 pred=350.0
```

```python
from sklearn.naive_bayes import GaussianNB
import networkx as nx
from sklearn.metrics import accuracy_score,classification_report,confusion_matrix
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
data={
 "age": [29, 45, 34, 60, 50, 41, 52, 39, 48, 59],
 "cholesterol": [200, 240, 210, 280, 260, 230, 300, 220, 250, 270],
 "bp": [120, 140, 130, 150, 140, 135, 160, 125, 145, 155],
 "heart_disease": [0, 1, 0, 1, 1, 0, 1, 0, 1, 1]
}

df=pd.DataFrame(data)

x=df[["age","cholesterol","bp"]]
y=df["heart_disease"]

x_train,x_test,y_train,y_test=train_test_split(
    x,y,test_size=0.3,random_state=42
)

model=GaussianNB()
model.fit(x_train,y_train)

y_pred=model.predict(x_test)
```

```
print("Accuracy:",accuracy_score(y_test,y_pred)*100)
print("classification report:\n",classification_report(y_test,y_pred))
print("Confusion matrix:\n",confusion_matrix(y_test,y_pred))

#predict heart disease or no heart disease for data
plt.scatter(df["cholesterol"],df["bp"],c=df["heart_disease"],cmap='coolwarm')
plt.xlabel("cholesterol")
plt.ylabel("bp")
plt.legend(["no disease","disease"])
plt.show()
#bayesian network
G=nx.DiGraph()
G.add_edges_from([
    ("age","heart_disease"),
    ("cholesterol","heart_disease"),
    ("bp","heart_disease")
])
plt.figure(figsize=(8,6))
nx.draw(G,with_labels=True,font_weight='bold',node_size=3000,node_color='lightblue',font_size=10)
plt.title("Bayesian network")
plt.show()
```

```
Accuracy: 100.0
classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         2

    accuracy                           1.00         3
   macro avg       1.00      1.00      1.00         3
weighted avg       1.00      1.00      1.00         3

Confusion matrix:
 [[1 0]
 [0 2]]
```





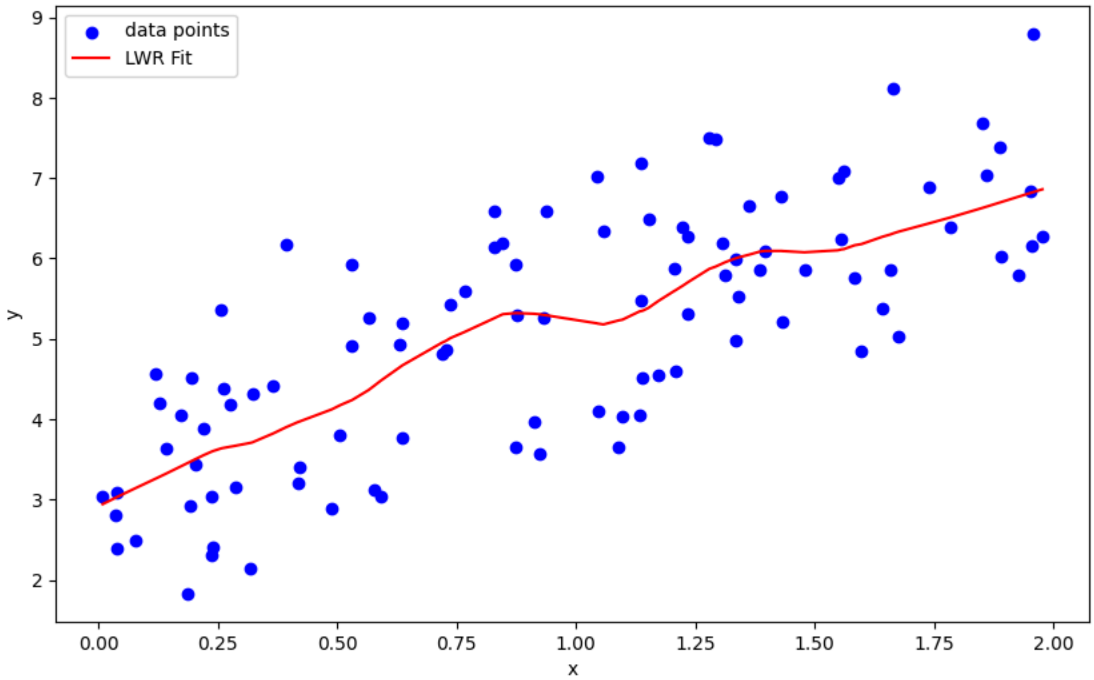

```
#6) knn to classify correct or wrong prediciton of Iris dataset

from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
import numpy as np

data=load_iris()
x=data.data
y=data.target
```

```python
x_train,x_test,y_train,y_test=train_test_split(
    x,y,test_size=0.3,random_state=42
)

def euclidean_distance(point1,point2):
  return np.sqrt(np.sum((point1-point2)**2))


def knn_classify(x_train,x_test,y_train,k=3):
  predictions=[]
  for test_point in x_test:
    # print(test_point)
    distances=[euclidean_distance(train_point,test_point) for train_point in x_train]
    # print(distances)
    nearest_indices=np.argsort(distances)[:k]
    # print(nearest_indices)
    nearest_labels=[y_train[i] for i in nearest_indices]
    # print(nearest_labels)
    most_common=Counter(nearest_labels).most_common(1)[0][0]
    # print("most_common: ",most_common)
    predictions.append(most_common)
  return predictions

correct=0
incorrect=0
predictions=knn_classify(x_train,x_test,y_train)
print(predictions)
for i,(actual, pred) in enumerate(zip(y_test,predictions)):
  if actual==pred:
    correct+=1
    print(f"Test sample {i+1}: correct (Actual:{actual} predicted: {pred})")
  else:
    incorrect+=1
    print(f"Test Sample {i+1}: Incorrect (Actual:{actual} predicted: {pred})")

print("Total correct Predictions:",correct)
print("Total Incorrect Predictions",incorrect)
print("Accuracy:",correct/len(y_test)*100)
```

```
[np.int64(1), np.int64(0), np.int64(2), np.int64(1), np.int64(1), np.int64(0), np.int64(1), np.int64(2), np.int64(1), np.int
Test sample 1: correct (Actual:1 predicted: 1)
Test sample 2: correct (Actual:0 predicted: 0)
Test sample 3: correct (Actual:2 predicted: 2)
Test sample 4: correct (Actual:1 predicted: 1)
Test sample 5: correct (Actual:1 predicted: 1)
Test sample 6: correct (Actual:0 predicted: 0)
Test sample 7: correct (Actual:1 predicted: 1)
Test sample 8: correct (Actual:2 predicted: 2)
Test sample 9: correct (Actual:1 predicted: 1)
Test sample 10: correct (Actual:1 predicted: 1)
Test sample 11: correct (Actual:2 predicted: 2)
Test sample 12: correct (Actual:0 predicted: 0)
Test sample 13: correct (Actual:0 predicted: 0)
Test sample 14: correct (Actual:0 predicted: 0)
Test sample 15: correct (Actual:0 predicted: 0)
Test sample 16: correct (Actual:1 predicted: 1)
Test sample 17: correct (Actual:2 predicted: 2)
Test sample 18: correct (Actual:1 predicted: 1)
Test sample 19: correct (Actual:1 predicted: 1)
Test sample 20: correct (Actual:2 predicted: 2)
Test sample 21: correct (Actual:0 predicted: 0)
Test sample 22: correct (Actual:2 predicted: 2)
Test sample 23: correct (Actual:0 predicted: 0)
Test sample 24: correct (Actual:2 predicted: 2)
Test sample 25: correct (Actual:2 predicted: 2)
Test sample 26: correct (Actual:2 predicted: 2)
Test sample 27: correct (Actual:2 predicted: 2)
Test sample 28: correct (Actual:2 predicted: 2)
Test sample 29: correct (Actual:0 predicted: 0)
Test sample 30: correct (Actual:0 predicted: 0)
Test sample 31: correct (Actual:0 predicted: 0)
Test sample 32: correct (Actual:0 predicted: 0)
Test sample 33: correct (Actual:1 predicted: 1)
Test sample 34: correct (Actual:0 predicted: 0)
Test sample 35: correct (Actual:0 predicted: 0)
Test sample 36: correct (Actual:2 predicted: 2)
Test sample 37: correct (Actual:1 predicted: 1)
Test sample 38: correct (Actual:0 predicted: 0)
Test sample 39: correct (Actual:0 predicted: 0)
Test sample 40: correct (Actual:0 predicted: 0)
Test sample 41: correct (Actual:2 predicted: 2)
Test sample 42: correct (Actual:1 predicted: 1)
Test sample 43: correct (Actual:1 predicted: 1)
Test sample 44: correct (Actual:0 predicted: 0)
Test sample 45: correct (Actual:0 predicted: 0)
Total correct Predictions: 45
```

```
Total Incorrect Predictions 0
Accuracy: 100.0
```

```
import numpy as np
from statsmodels.nonparametric.smoothers_lowess import lowess
import matplotlib.pyplot as plt

np.random.seed(0)
x=2*np.random.rand(100)
y=3+2*x+np.random.randn(100)

sort_idx=np.argsort(x)
x=x[sort_idx]
y=y[sort_idx]

lowess_result=lowess(y,x,frac=0.29)

x_smooth=lowess_result[:,0]
y_smooth=lowess_result[:,1]

plt.figure(figsize=(10,6))
plt.scatter(x,y,color='blue',label="data points")
plt.plot(x_smooth,y_smooth,color='red',label='LWR Fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```




```
import nltk
# nltk.download('punkt_tab')
# nltk.download('averaged_perceptron_tagger')
# nltk.download('stopwords')
from nltk.tokenize import word_tokenize,sent_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk import pos_tag


text='''Adverstising has us chasing car and clothes, working jobs we hate so we can buy shit we dont need.
we are the middle children of history. No great war or purpose.'''

#a
sentences=sent_tokenize(text)
print(sentences)
print("\n")
words=word_tokenize(text)
print(words)
print("\n")
```

```
#b
stop_words=set(stopwords.words('english'))
filtered_words=[word for word in words if word.lower() not in stop_words]
print(filtered_words)
print("\n")

#c
stemmer=PorterStemmer()
stemmed_words=[stemmer.stem(word) for word in filtered_words]
print(stemmed_words)
print("\n")

#d
pos_tagging=pos_tag(words)
print(pos_tagging)
```

```
['Adverstising has us chasing car and clothes, working jobs we hate so we can buy shit we dont need.', 'we are the middle ch


['Adverstising', 'has', 'us', 'chasing', 'car', 'and', 'clothes', ',', 'working', 'jobs', 'we', 'hate', 'so', 'we', 'can', '


['Adverstising', 'us', 'chasing', 'car', 'clothes', ',', 'working', 'jobs', 'hate', 'buy', 'shit', 'dont', 'need', '.', 'mic


['adverstis', 'us', 'chase', 'car', 'cloth', ',', 'work', 'job', 'hate', 'buy', 'shit', 'dont', 'need', '.', 'middl', 'chilc


[('Adverstising', 'NN'), ('has', 'VBZ'), ('us', 'PRP'), ('chasing', 'VBG'), ('car', 'NN'), ('and', 'CC'), ('clothes', 'NNS')
```

```
import numpy as np
gridsize=(5,5)
num_states=gridsize[0]*gridsize[1]
actions=["up","down","left","right"]
num_actions=len(actions)
# print(gridsize)

def coordinates_to_state(x,y):
  return x*gridsize[1]+y

def state_to_coordinates(state):
  return divmod(state,gridsize[1])

def take_actions(state,action):
  x,y=state_to_coordinates(state)
  if(action=="up"):
    x=max(0,x-1)
  elif(action=="down"):
    x=min(gridsize[0]-1,x+1)
  elif(action=="left"):
    y=max(0,y-1)
  elif(action=="right"):
    y=min(gridsize[1]-1,y+1)
  return coordinates_to_state(x,y)


terminal_state=coordinates_to_state(4,4)
rewards=np.full(num_states,-1)
rewards[terminal_state]=100
print(rewards)

alpha=0.1
gamma=0.9
epsilon=0.2
q_table=np.zeros((num_states,num_actions))
# print(q_table)
num_episodes=500

for episode in range(num_episodes):
  state=np.random.randint(0,num_states)
  while state!=terminal_state:
    if np.random.rand()<epsilon:
      action=np.random.randint(0,num_actions)
    else:
      action=np.argmax(q_table[state])
    next_state=take_actions(state,actions[action])
    reward=rewards[next_state]
    best_next_action=np.max(q_table[next_state])
    q_table[state,action] += alpha * (reward + gamma * best_next_action - q_table[state,action])
    state=next_state

# print(q_table)
policy=np.argmax(q_table,axis=1)
```

```
policy_grid=np.array([actions[a] for a in policy]).reshape(gridsize)
print(policy_grid)
print(q_table)
```

```
[ -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
  -1  -1  -1  -1  -1  -1 100]
[['right' 'down' 'right' 'down' 'down']
 ['right' 'right' 'right' 'right' 'down']
 ['right' 'right' 'right' 'right' 'down']
 ['right' 'right' 'down' 'down' 'down']
 ['right' 'right' 'right' 'right' 'up']]
[[-1.03839580e+00  5.23974700e-02  3.63305641e-01  1.33018730e+01]
 [ 9.04113366e+00  4.37848592e+01  1.04710003e-02  3.56274090e+00]
 [ 2.07402930e+00 -7.11821910e-01  5.76108778e+00  4.84168346e+01]
 [ 7.52803947e+00  6.18905649e+01  1.09131511e+01  1.32429587e+01]
 [ 6.30247311e+00  6.75875998e+01 -5.03149264e-01  5.37774916e+00]
 [-9.91778893e-01 -9.89438167e-01  5.86313628e-01  4.01896247e+01]
 [ 7.35553054e+00  6.67780235e-01  9.65574248e+00  5.47955486e+01]
 [ 8.62517760e+00  1.10095008e+01  6.37382589e+00  6.21593278e+01]
 [ 3.77129019e+01  5.51826959e+01  2.46188743e+01  7.01899596e+01]
 [ 2.84620880e+01  7.90999998e+01  4.79540356e+01  5.76550762e+01]
 [ 3.35348359e+00 -8.04630164e-01  1.26428103e+00  3.29316197e+01]
 [ 7.31374360e+00  4.65910248e+00  2.26419023e+00  5.63185430e+01]
 [ 1.60638663e+01  8.22473259e+00  2.45203921e+00  6.97694489e+01]
 [ 1.19413541e+01  3.84434826e+01  1.77093369e+01  7.90973473e+01]
 [ 6.05080693e+01  8.90000000e+01  5.98295315e+01  6.59769745e+01]
 [-7.62335767e-01  2.12418402e+00 -5.84412564e-01  3.61976988e+01]
 [-3.86677000e-01 -3.85978837e-01  5.22807166e+00  6.47390108e+01]
 [-3.05758000e-01  7.88974102e+01  3.28005238e+00  1.45565475e+01]
 [ 2.96913180e+01  8.85664827e+01  9.63147263e+00  1.68289998e+01]
 [ 5.65493882e+01  1.00000000e+02  6.74260127e+01  7.30428312e+01]
 [-6.80835867e-01 -4.90099501e-01 -4.90099501e-01  4.72316354e+01]
 [ 9.23002057e+00  1.07948953e+01  1.51923536e+00  7.78814802e+01]
 [ 3.36289525e+01  4.41606613e+01  3.37413280e+01  8.89998835e+01]
 [ 5.48442739e+01  6.85817254e+01  6.11711650e+01  9.99999997e+01]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00]]
```

Start coding or generate with AI.

```
Trained Output:
[[0.01617295]
 [0.98334289]
 [0.98758845]
 [0.01468905]]

Testing the ANN:
Input: [0 0], Predicted Output: [[0.016172]]
Input: [0 1], Predicted Output: [[0.98334387]]
Input: [1 0], Predicted Output: [[0.98758925]]
Input: [1 1], Predicted Output: [[0.01468812]]
```