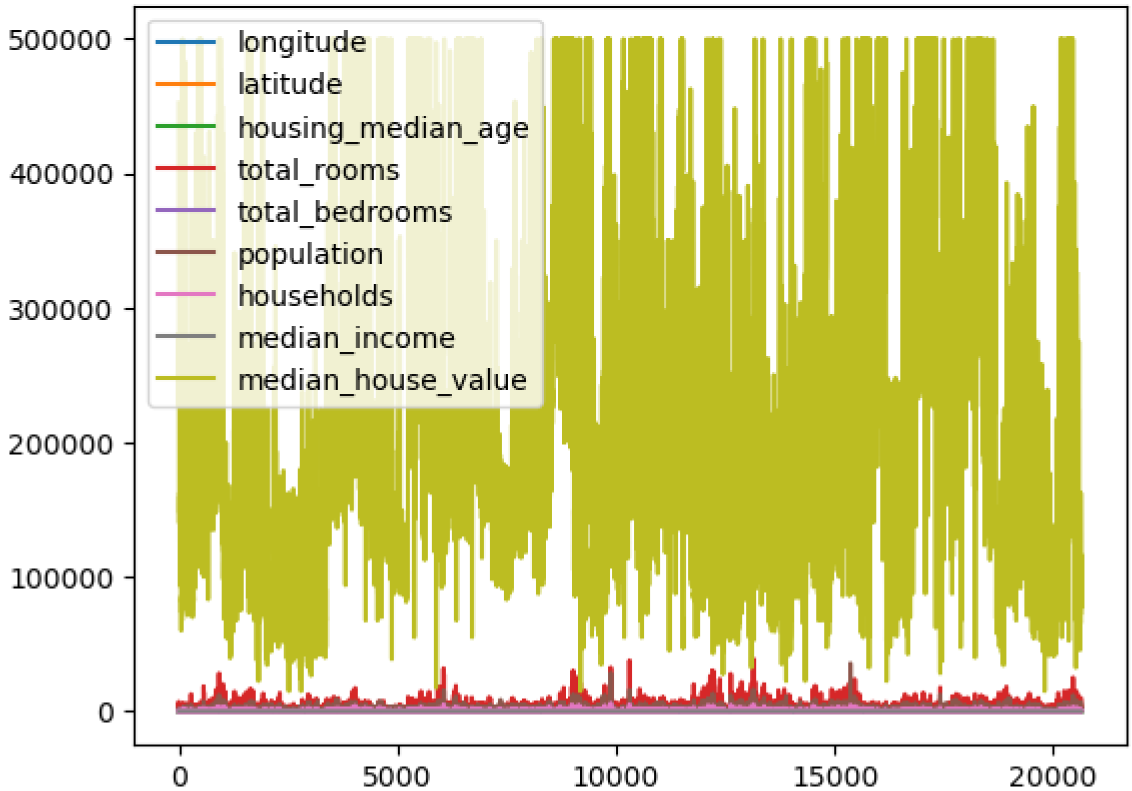
In [5]:
```
#1 linear regression
import numpy as np
from sklearn.linear_model import LinearRegression

hours_studied = np.array([2, 3, 4, 5, 6]).reshape(-1,1)
exam_scores = np.array([50, 47, 96, 89, 69])
model = LinearRegression()
model.fit(hours_studied,exam_scores)     #model_fit(independent,dependent)
new = np.array([24]).reshape(-1,1)
predicted_scores = model.predict(new)
if predicted_scores>100:
    predicted_scores=100

print("Predicted exam score for 7 hours studied:", predicted_scores)
```

```
Predicted exam score for 7 hours studied: 100
```

In [25]:
```
#2 gradient decent
import csv
import numpy as np
import pandas as pd
# import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score

path="housing.csv"

#import datset

data=pd.read_csv(path)
print(data.head()) #displaying 5 rows of data


count=data.info()
print(count)

#to print number of null values
print(data.isnull().sum())

data.plot()
plt.show()

#cov matrix and corr matrix
cov_mat=data.cov(numeric_only=True)
corr_mat=data.corr(numeric_only=True)
print(cov_mat)
print(corr_mat)

#train and test model

X=data.drop(["median_house_value"],axis=1)
y=data["median_house_value"]
X_encoded = pd.get_dummies(X, columns=['ocean_proximity'])
X_encoded.fillna(data["total_bedrooms"].mean(), inplace=True)

X_train,X_test,y_train,y_test=train_test_split(X_encoded,y,test_size=0.09,random_state
```

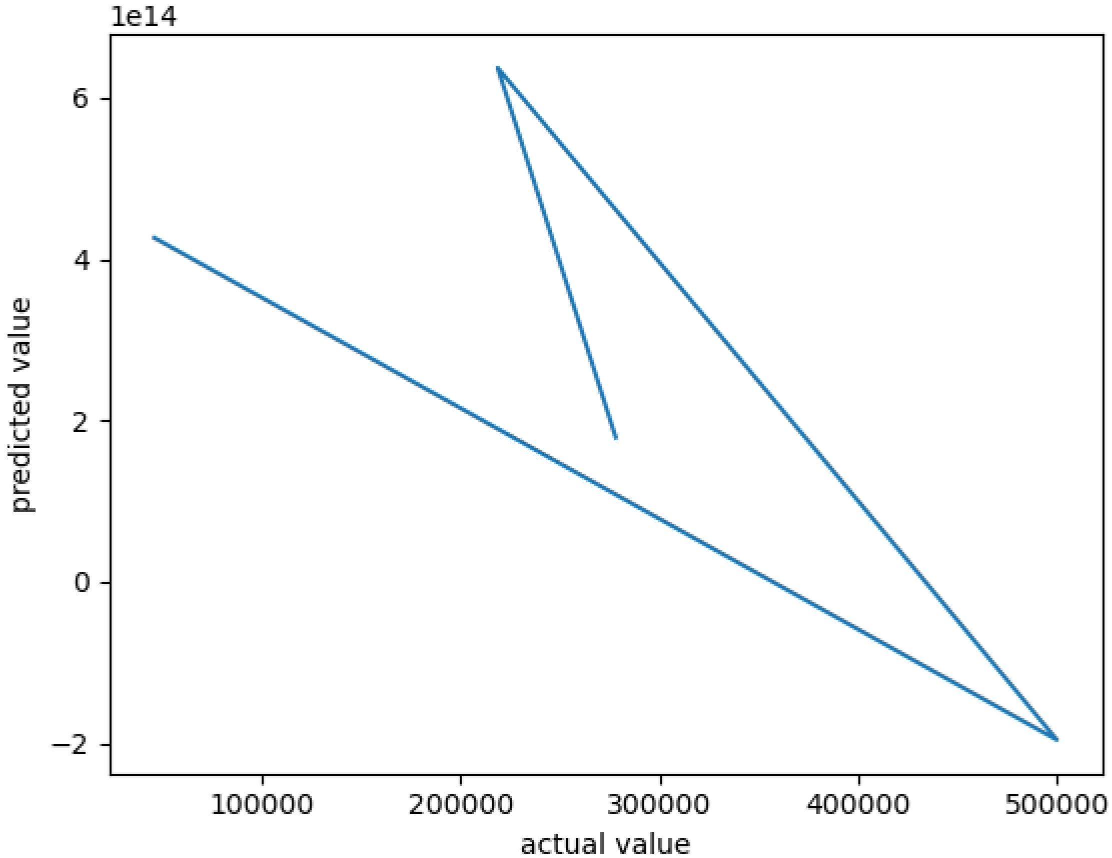
```
model=SGDRegressor()
model.fit(X_train,y_train)

#predicting values

y_pred=model.predict(X_test)
print(y_pred)


#accuracy and its graph
# mse = mean_squared_error(y_test, y_pred)
# a = 1 - (mse / np.var(y_test))
a=model.score(X_test,y_test)
# a=accuracy_score(y_test,y_pred)
print(f"the accuracy of the model is :  {a}  ")

plt.plot(y_test[1:10],y_pred[1:10])
plt.xlabel("actual value")
plt.ylabel("predicted value")
plt.show()
```

```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0   
1    -122.22     37.86                21.0       7099.0          1106.0   
2    -122.24     37.85                52.0       1467.0           190.0   
3    -122.25     37.85                52.0       1274.0           235.0   
4    -122.25     37.85                52.0       1627.0           280.0   

   population  households  median_income  median_house_value ocean_proximity  
0       322.0       126.0         8.3252            452600.0        NEAR BAY  
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY  
2       496.0       177.0         7.2574            352100.0        NEAR BAY  
3       558.0       219.0         5.6431            341300.0        NEAR BAY  
4       565.0       259.0         3.8462            342200.0        NEAR BAY  
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
None
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```


longitude
latitude
housing_median_age
total_rooms
total_bedrooms
population
households
median_income
median_house_value
500000
400000
300000
200000
100000
0
0
5000
10000
15000
20000

```
                     longitude      latitude  housing_median_age  \
longitude             4.014139     -3.957054           -2.728244
latitude             -3.957054      4.562293            0.300346
housing_median_age   -2.728244      0.300346          158.396260
total_rooms         194.803750   -168.217847        -9919.120060
total_bedrooms       58.768508    -60.299623        -1700.312817
population          226.377839   -263.137814        -4222.270582
households           42.368072    -58.010245        -1457.581290
median_income        -0.057765     -0.323860           -2.846140
median_house_value -10627.425205 -35532.559074       153398.801329

                     total_rooms  total_bedrooms    population    households  \
longitude           1.948037e+02    5.876851e+01  2.263778e+02  4.236807e+01
latitude           -1.682178e+02   -6.029962e+01 -2.631378e+02 -5.801024e+01
housing_median_age -9.919120e+03   -1.700313e+03 -4.222271e+03 -1.457581e+03
total_rooms         4.759445e+06    8.567306e+05  2.117613e+06  7.661046e+05
total_bedrooms      8.567306e+05    1.775654e+05  4.191391e+05  1.578295e+05
population          2.117613e+06    4.191391e+05  1.282470e+06  3.928036e+05
households          7.661046e+05    1.578295e+05  3.928036e+05  1.461760e+05
median_income       8.208524e+02   -6.180851e+00  1.040098e+01  9.466667e+00
median_house_value  3.377289e+07    2.416878e+06 -3.221249e+06  2.904924e+06

                    median_income  median_house_value
longitude               -0.057765       -1.062743e+04
latitude                -0.323860       -3.553256e+04
housing_median_age      -2.846140        1.533988e+05
total_rooms            820.852410        3.377289e+07
total_bedrooms          -6.180851        2.416878e+06
population              10.400979       -3.221249e+06
households               9.466667        2.904924e+06
median_income            3.609323        1.508475e+05
median_house_value  150847.482793        1.331615e+10
                    longitude  latitude  housing_median_age  total_rooms  \
longitude            1.000000 -0.924664           -0.108197     0.044568
latitude            -0.924664  1.000000            0.011173    -0.036100
housing_median_age  -0.108197  0.011173            1.000000    -0.361262
total_rooms          0.044568 -0.036100           -0.361262     1.000000
total_bedrooms       0.069608 -0.066983           -0.320451     0.930380
population           0.099773 -0.108785           -0.296244     0.857126
households           0.055310 -0.071035           -0.302916     0.918484
median_income       -0.015176 -0.079809           -0.119034     0.198050
median_house_value  -0.045967 -0.144160            0.105623     0.134153

                    total_bedrooms  population  households  median_income  \
longitude                 0.069608    0.099773    0.055310      -0.015176
latitude                 -0.066983   -0.108785   -0.071035      -0.079809
housing_median_age       -0.320451   -0.296244   -0.302916      -0.119034
total_rooms               0.930380    0.857126    0.918484       0.198050
total_bedrooms            1.000000    0.877747    0.979728      -0.007723
population                0.877747    1.000000    0.907222       0.004834
households                0.979728    0.907222    1.000000       0.013033
median_income            -0.007723    0.004834    0.013033       1.000000
median_house_value        0.049686   -0.024650    0.065843       0.688075

                    median_house_value
longitude                    -0.045967
latitude                     -0.144160
housing_median_age            0.105623
total_rooms                   0.134153
total_bedrooms                0.049686
```

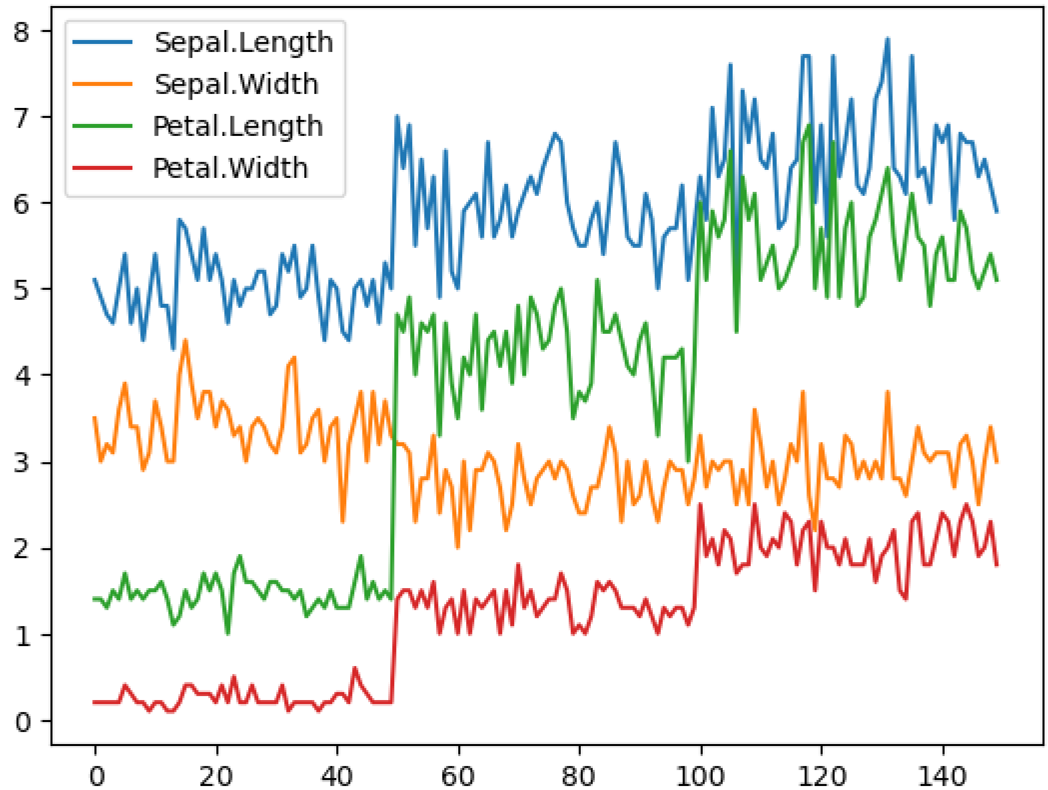
```
population                    -0.024650
households                     0.065843
median_income                  0.688075
median_house_value             1.000000
[ 5.33974811e+14  4.26342560e+14 -1.95930124e+14 ...  6.16989976e+14
  3.68270423e+14  2.12002920e+14]
the accuracy of the model is :  -2.799295789483201e+19
```



In [32]:
```
#3
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

iris_df = pd.read_csv("IRIS.csv")
print(iris_df.head())

print(iris_df.count())

print(iris_df.isnull().any()) #to check null values are present or not
print(iris_df.isnull().sum()) #to print number of null values

iris=iris_df.drop(['ID'],axis=1)
iris.plot()                          #graph representation
plt.show()

cov_mat=iris.cov()
print(cov_mat)
```

```
corr_mat=iris.corr()
print(corr_mat)

X=iris.drop(["Species"],axis=1)  #to train and test model
y=iris["Species"]

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)

model=LogisticRegression()
model.fit(X_train,y_train)

y_pred=model.predict(X_test)
print(y_pred)
a=model.score(X_test,y_test)

aa=accuracy_score(y_test,y_pred)
print(f"the accuracy is : {a}   {aa}")
```

```
   ID  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
0   1           5.1          3.5           1.4          0.2  setosa
1   2           4.9          3.0           1.4          0.2  setosa
2   3           4.7          3.2           1.3          0.2  setosa
3   4           4.6          3.1           1.5          0.2  setosa
4   5           5.0          3.6           1.4          0.2  setosa
ID              150
Sepal.Length    150
Sepal.Width     150
Petal.Length    150
Petal.Width     150
Species         150
dtype: int64
ID              False
Sepal.Length    False
Sepal.Width     False
Petal.Length    False
Petal.Width     False
Species         False
dtype: bool
ID              0
Sepal.Length    0
Sepal.Width     0
Petal.Length    0
Petal.Width     0
Species         0
dtype: int64
```



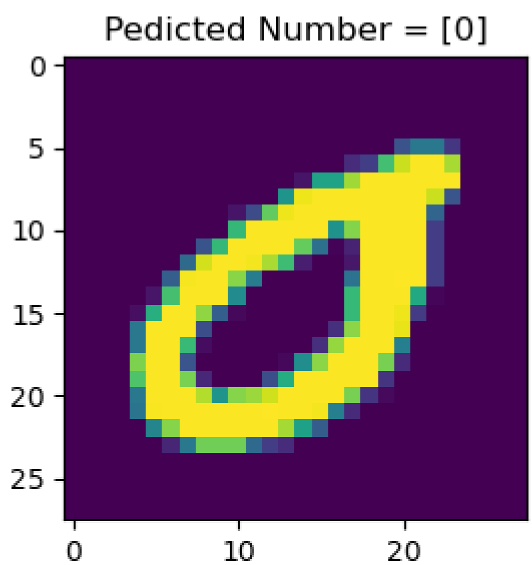

```
              Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
Sepal.Length      0.685694    -0.042434      1.274315     0.516271
Sepal.Width      -0.042434     0.189979     -0.329656    -0.121639
Petal.Length      1.274315    -0.329656      3.116278     1.295609
Petal.Width       0.516271    -0.121639      1.295609     0.581006
              Sepal.Length  Sepal.Width  Petal.Length  Petal.Width
Sepal.Length      1.000000    -0.117570      0.871754     0.817941
Sepal.Width      -0.117570     1.000000     -0.428440    -0.366126
Petal.Length      0.871754    -0.428440      1.000000     0.962865
Petal.Width       0.817941    -0.366126      0.962865     1.000000
['versicolor' 'virginica' 'setosa' 'versicolor' 'setosa' 'versicolor'
 'versicolor' 'versicolor' 'setosa' 'versicolor' 'versicolor' 'virginica'
 'versicolor' 'setosa' 'setosa' 'virginica' 'versicolor' 'setosa' 'setosa'
 'setosa' 'virginica' 'virginica' 'virginica' 'setosa' 'versicolor'
 'setosa' 'versicolor' 'versicolor' 'versicolor' 'virginica']
the accuracy is : 1.0   1.0
```

In [11]:

```python
#4  mnist
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def show_digit(no):
    pred_no = model.predict([X_test[no]])
    img_array = X_test[no].reshape((28,28))
    plt.figure(figsize=(3, 3))
    plt.title(f"Pedicted Number = {pred_no}")
    plt.imshow(img_array)

df = pd.read_csv("train.csv")
df.head(10)
```

```
X = df.values[:,1:]
y = df.values[:,0]
print(y[6])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
a=model.score(X_test,y_test)

print(f"the accuracy is : {a}")

show_digit(1)

# plt.plot(y_test,y_pred)
# plt.show()
```

```
7
the accuracy is : 0.9661904761904762
```



In [ ]:

```
#  5  FIND S

import csv

def find_s(training_data):
    hypothesis=[]
    hypothesis = training_data[0][:-1]
    for example in training_data:
        features = example[:-1]
        label = example[-1]
        if label == 'Yes':
            for i in range(len(hypothesis)):
                if hypothesis[i] != features[i]:
                    hypothesis[i] = '?'
            print(hypothesis)
    return hypothesis

training_data = []
with open('enjoysport.csv.csv', 'r') as file:
    csv_reader = csv.reader(file)
```

```
    for row in csv_reader:
        training_data.append(row)
    print(training_data)
    training_data.pop(0)
    print(training_data)

h = find_s(training_data)
print("Most specific hypothesis:", h)
```

```
[['Sky', 'Airtemp', 'Humidity', 'Wind', 'Water', 'Forecast', 'WaterSport'], ['Sunny',
'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'], ['Sunny', 'Warm', 'High', 'Stron
g', 'Warm', 'Same', 'Yes'], ['Cloudy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'N
o'], ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']]
[['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'], ['Sunny', 'Warm', 'Hig
h', 'Strong', 'Warm', 'Same', 'Yes'], ['Cloudy', 'Cold', 'High', 'Strong', 'Warm', 'C
hange', 'No'], ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']]
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
['Sunny', 'Warm', '?', 'Strong', '?', '?']
Most specific hypothesis: ['Sunny', 'Warm', '?', 'Strong', '?', '?']
```

In [ ]:

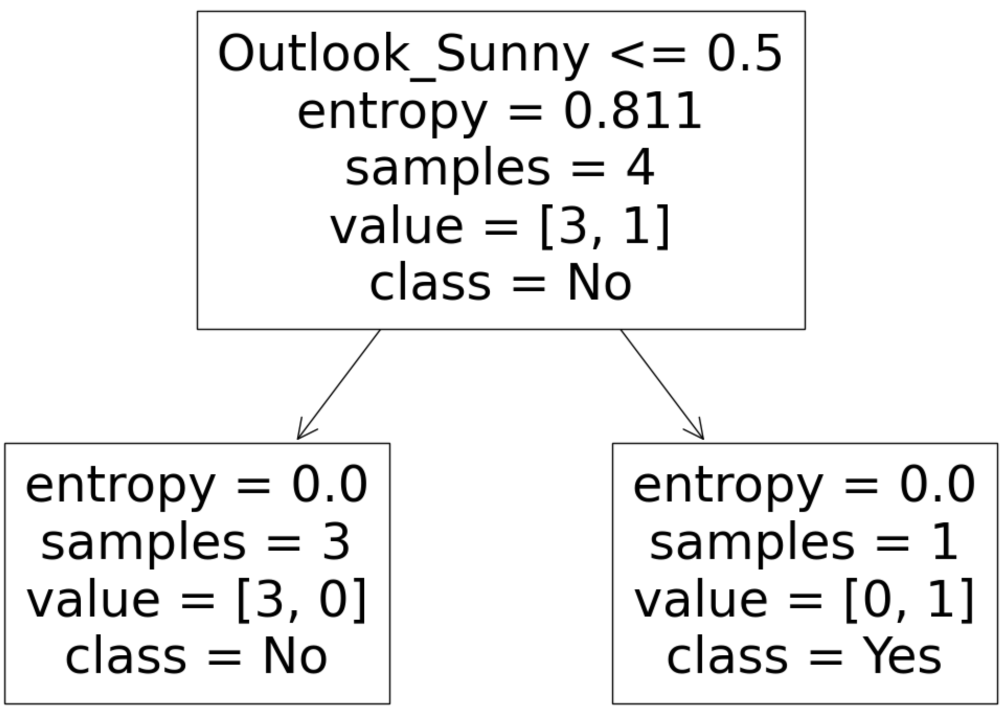
```
# 6 candidate elimination
import numpy as np
import pandas as pd

data = pd.DataFrame(data=pd.read_csv('enjoysport.csv.csv'))
concepts = np.array(data.iloc[:,:-1])
print(concepts)

target = np.array(data.iloc[:,-1])
print(target)

def learn(concepts, target):
    specific_h = concepts[0].copy()
    print(specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))
    print(general_h)

    for i, h in enumerate(concepts):
        print("\nInstance", i+1, "is", h)
        if target[i] == "Yes":
            print("Instance is Positive")
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = '?'
                    general_h[x][x] = '?'
        if target[i] == "No":
            print("Instance is Negative")
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'

        print("Specific Boundary after", i+1, "Instance is", specific_h)
        print("Generic Boundary after", i+1, "Instance is", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?
    for i in indices:
```

```
        general_h.remove(['?', '?', '?', '?', '?', '?'])

    return specific_h, general_h

s_final, g_final = learn(concepts, target)
print("Final Specific_h:", s_final, sep="\n")
print("Final General_h:", g_final, sep="\n")
```

```
[['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Cloudy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
['Yes' 'Yes' 'No' 'Yes']
['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?']]

Instance 1 is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
Instance is Positive
Specific Boundary after 1 Instance is ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Sam
e']
Generic Boundary after 1 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance 2 is ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
Instance is Positive
Specific Boundary after 2 Instance is ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
Generic Boundary after 2 Instance is [['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance 3 is ['Cloudy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
Instance is Negative
Specific Boundary after 3 Instance is ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
Generic Boundary after 3 Instance is [['Sunny', '?', '?', '?', '?', '?'], ['?', 'War
m', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]



Instance 4 is ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']
Instance is Positive
Specific Boundary after 4 Instance is ['Sunny' 'Warm' '?' 'Strong' '?' '?']
Generic Boundary after 4 Instance is [['Sunny', '?', '?', '?', '?', '?'], ['?', 'War
m', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Final Specific_h:
['Sunny' 'Warm' '?' 'Strong' '?' '?']
Final General_h:
[['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?']]
```

In [35]:
```
# 7ID3

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

path = "/content/testtennis.csv"
data = pd.read_csv(path)

X = data.drop('playtennis', axis=1)
y = data['playtennis']

X_encoded = pd.get_dummies(X)

X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, rando

decision_tree = DecisionTreeClassifier(criterion='entropy')
decision_tree.fit(X_train, y_train)

new_sample = X_test.iloc[[0]]
predicted_class = decision_tree.predict(new_sample)

print("Predicted class for the new sample:", predicted_class[0])

plt.figure(figsize=(15, 10))
plot_tree(decision_tree, feature_names=X_encoded.columns, class_names=['No', 'Yes'])
plt.show()
```

```
Predicted class for the new sample: no
```



In [ ]:
```
# 8  ANN
import numpy as np
```

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


X = np.array(([2, 9],[1, 5],[3, 6]), dtype=float)
y=np.array(([92],[86],[89]),dtype=float)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='linear')
])


model.compile(optimizer='adam', loss='mean_squared_error')

history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_te

loss = model.evaluate(X_test, y_test)
print(f"Test loss: {loss}")

plt.plot(history.history['loss'], label='Train')
plt.plot(history.history['val_loss'], label='Validation')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Learning Curve')
plt.legend()
plt.show()
```

```
Epoch 1/100
1/1 [==============================] - 1s 1s/step - loss: 7692.9087 - val_loss: 8401.
7930
Epoch 2/100
1/1 [==============================] - 0s 42ms/step - loss: 7688.9844 - val_loss: 838
8.5908
Epoch 3/100
1/1 [==============================] - 0s 41ms/step - loss: 7685.0610 - val_loss: 837
5.3535
Epoch 4/100
1/1 [==============================] - 0s 60ms/step - loss: 7681.1377 - val_loss: 836
2.0898
Epoch 5/100
1/1 [==============================] - 0s 72ms/step - loss: 7677.2168 - val_loss: 834
8.7910
Epoch 6/100
1/1 [==============================] - 0s 48ms/step - loss: 7673.2939 - val_loss: 833
5.4482
Epoch 7/100
1/1 [==============================] - 0s 65ms/step - loss: 7669.3687 - val_loss: 832
2.0605
Epoch 8/100
1/1 [==============================] - 0s 42ms/step - loss: 7665.4521 - val_loss: 830
8.6230
Epoch 9/100
1/1 [==============================] - 0s 42ms/step - loss: 7661.5322 - val_loss: 829
5.1328
Epoch 10/100
1/1 [==============================] - 0s 42ms/step - loss: 7657.6060 - val_loss: 828
1.5869
Epoch 11/100
1/1 [==============================] - 0s 42ms/step - loss: 7653.6748 - val_loss: 826
7.9814
Epoch 12/100
1/1 [==============================] - 0s 41ms/step - loss: 7649.9502 - val_loss: 825
4.3193
Epoch 13/100
1/1 [==============================] - 0s 43ms/step - loss: 7646.2866 - val_loss: 824
0.5791
Epoch 14/100
1/1 [==============================] - 0s 42ms/step - loss: 7642.6123 - val_loss: 822
6.7598
Epoch 15/100
1/1 [==============================] - 0s 42ms/step - loss: 7638.9282 - val_loss: 821
2.8213
Epoch 16/100
1/1 [==============================] - 0s 44ms/step - loss: 7635.2002 - val_loss: 819
8.7627
Epoch 17/100
1/1 [==============================] - 0s 67ms/step - loss: 7631.2007 - val_loss: 818
4.6362
Epoch 18/100
1/1 [==============================] - 0s 59ms/step - loss: 7627.1182 - val_loss: 817
0.4473
Epoch 19/100
1/1 [==============================] - 0s 77ms/step - loss: 7622.9746 - val_loss: 815
6.1963
Epoch 20/100
1/1 [==============================] - 0s 44ms/step - loss: 7618.7808 - val_loss: 814
1.8833
```

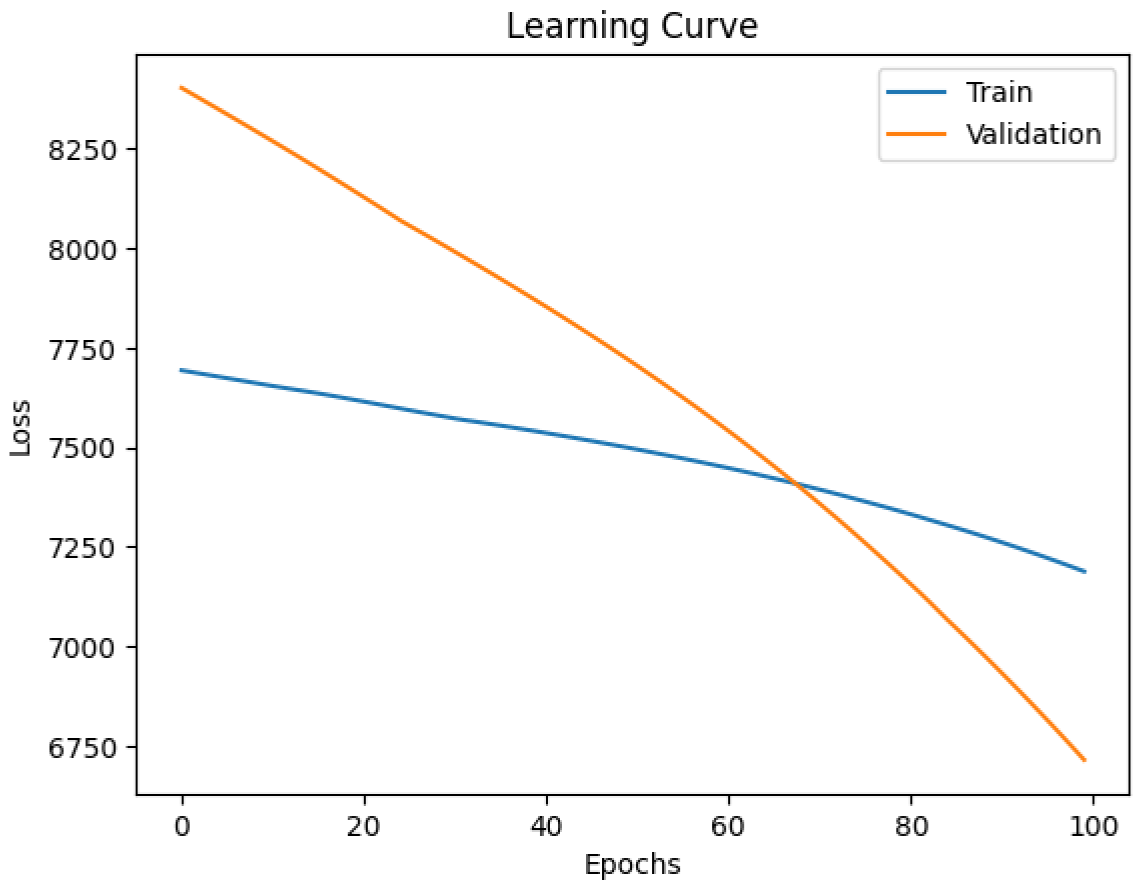
```
Epoch 21/100
1/1 [==============================] - 0s 63ms/step - loss: 7614.5449 - val_loss: 812
7.5068
Epoch 22/100
1/1 [==============================] - 0s 52ms/step - loss: 7610.2705 - val_loss: 811
3.0679
Epoch 23/100
1/1 [==============================] - 0s 44ms/step - loss: 7605.9609 - val_loss: 809
8.5254
Epoch 24/100
1/1 [==============================] - 0s 48ms/step - loss: 7601.6201 - val_loss: 808
3.8335
Epoch 25/100
1/1 [==============================] - 0s 64ms/step - loss: 7597.2471 - val_loss: 806
9.0068
Epoch 26/100
1/1 [==============================] - 0s 59ms/step - loss: 7592.8574 - val_loss: 805
5.9443
Epoch 27/100
1/1 [==============================] - 0s 37ms/step - loss: 7588.5991 - val_loss: 804
2.9624
Epoch 28/100
1/1 [==============================] - 0s 41ms/step - loss: 7584.5078 - val_loss: 802
9.8950
Epoch 29/100
1/1 [==============================] - 0s 51ms/step - loss: 7580.3765 - val_loss: 801
6.7407
Epoch 30/100
1/1 [==============================] - 0s 58ms/step - loss: 7576.2061 - val_loss: 800
3.4956
Epoch 31/100
1/1 [==============================] - 0s 58ms/step - loss: 7572.0000 - val_loss: 799
0.1548
Epoch 32/100
1/1 [==============================] - 0s 37ms/step - loss: 7568.2969 - val_loss: 797
6.7471
Epoch 33/100
1/1 [==============================] - 0s 36ms/step - loss: 7564.9126 - val_loss: 796
3.2720
Epoch 34/100
1/1 [==============================] - 0s 43ms/step - loss: 7561.4624 - val_loss: 794
9.7227
Epoch 35/100
1/1 [==============================] - 0s 63ms/step - loss: 7557.9277 - val_loss: 793
6.0962
Epoch 36/100
1/1 [==============================] - 0s 40ms/step - loss: 7554.3379 - val_loss: 792
2.3892
Epoch 37/100
1/1 [==============================] - 0s 56ms/step - loss: 7550.6963 - val_loss: 790
8.5977
Epoch 38/100
1/1 [==============================] - 0s 51ms/step - loss: 7547.0039 - val_loss: 789
4.7168
Epoch 39/100
1/1 [==============================] - 0s 38ms/step - loss: 7543.2632 - val_loss: 788
0.7422
Epoch 40/100
1/1 [==============================] - 0s 38ms/step - loss: 7539.4731 - val_loss: 786
6.6704
```

```
Epoch 41/100
1/1 [==============================] - 0s 39ms/step - loss: 7535.6343 - val_loss: 785
2.4961
Epoch 42/100
1/1 [==============================] - 0s 38ms/step - loss: 7531.7451 - val_loss: 783
8.2144
Epoch 43/100
1/1 [==============================] - 0s 42ms/step - loss: 7527.8076 - val_loss: 782
3.8218
Epoch 44/100
1/1 [==============================] - 0s 56ms/step - loss: 7523.8193 - val_loss: 780
9.3125
Epoch 45/100
1/1 [==============================] - 0s 38ms/step - loss: 7519.7783 - val_loss: 779
4.6836
Epoch 46/100
1/1 [==============================] - 0s 40ms/step - loss: 7515.6846 - val_loss: 777
9.9297
Epoch 47/100
1/1 [==============================] - 0s 45ms/step - loss: 7511.5376 - val_loss: 776
5.0449
Epoch 48/100
1/1 [==============================] - 0s 41ms/step - loss: 7507.3350 - val_loss: 775
0.0273
Epoch 49/100
1/1 [==============================] - 0s 37ms/step - loss: 7503.0762 - val_loss: 773
4.8706
Epoch 50/100
1/1 [==============================] - 0s 39ms/step - loss: 7498.7598 - val_loss: 771
9.5713
Epoch 51/100
1/1 [==============================] - 0s 37ms/step - loss: 7494.3872 - val_loss: 770
4.0908
Epoch 52/100
1/1 [==============================] - 0s 43ms/step - loss: 7489.9648 - val_loss: 768
8.4634
Epoch 53/100
1/1 [==============================] - 0s 45ms/step - loss: 7485.4819 - val_loss: 767
2.6851
Epoch 54/100
1/1 [==============================] - 0s 55ms/step - loss: 7480.9346 - val_loss: 765
6.8213
Epoch 55/100
1/1 [==============================] - 0s 38ms/step - loss: 7476.3486 - val_loss: 764
0.7515
Epoch 56/100
1/1 [==============================] - 0s 40ms/step - loss: 7471.7402 - val_loss: 762
4.5132
Epoch 57/100
1/1 [==============================] - 0s 38ms/step - loss: 7467.0645 - val_loss: 760
8.1030
Epoch 58/100
1/1 [==============================] - 0s 50ms/step - loss: 7462.3188 - val_loss: 759
1.5156
Epoch 59/100
1/1 [==============================] - 0s 48ms/step - loss: 7457.5020 - val_loss: 757
4.7490
Epoch 60/100
1/1 [==============================] - 0s 38ms/step - loss: 7452.6113 - val_loss: 755
7.8013
```

```
Epoch 61/100
1/1 [==============================] - 0s 38ms/step - loss: 7447.6479 - val_loss: 754
0.6655
Epoch 62/100
1/1 [==============================] - 0s 37ms/step - loss: 7442.6553 - val_loss: 752
3.3086
Epoch 63/100
1/1 [==============================] - 0s 38ms/step - loss: 7437.5083 - val_loss: 750
5.7651
Epoch 64/100
1/1 [==============================] - 0s 37ms/step - loss: 7432.3281 - val_loss: 748
8.0298
Epoch 65/100
1/1 [==============================] - 0s 39ms/step - loss: 7427.0674 - val_loss: 747
0.1001
Epoch 66/100
1/1 [==============================] - 0s 37ms/step - loss: 7421.7231 - val_loss: 745
1.9746
Epoch 67/100
1/1 [==============================] - 0s 40ms/step - loss: 7416.2969 - val_loss: 743
3.6494
Epoch 68/100
1/1 [==============================] - 0s 48ms/step - loss: 7410.7852 - val_loss: 741
5.1226
Epoch 69/100
1/1 [==============================] - 0s 41ms/step - loss: 7405.1855 - val_loss: 739
6.3896
Epoch 70/100
1/1 [==============================] - 0s 34ms/step - loss: 7399.4990 - val_loss: 737
7.4482
Epoch 71/100
1/1 [==============================] - 0s 33ms/step - loss: 7393.7227 - val_loss: 735
8.2998
Epoch 72/100
1/1 [==============================] - 0s 34ms/step - loss: 7387.8545 - val_loss: 733
8.9380
Epoch 73/100
1/1 [==============================] - 0s 33ms/step - loss: 7381.8955 - val_loss: 731
9.3618
Epoch 74/100
1/1 [==============================] - 0s 32ms/step - loss: 7375.8418 - val_loss: 729
9.5688
Epoch 75/100
1/1 [==============================] - 0s 35ms/step - loss: 7369.6929 - val_loss: 727
9.5571
Epoch 76/100
1/1 [==============================] - 0s 34ms/step - loss: 7363.5820 - val_loss: 725
9.3672
Epoch 77/100
1/1 [==============================] - 0s 34ms/step - loss: 7357.3340 - val_loss: 723
9.0259
Epoch 78/100
1/1 [==============================] - 0s 33ms/step - loss: 7351.0513 - val_loss: 721
8.5254
Epoch 79/100
1/1 [==============================] - 0s 36ms/step - loss: 7344.6675 - val_loss: 719
7.8560
Epoch 80/100
1/1 [==============================] - 0s 42ms/step - loss: 7338.1802 - val_loss: 717
7.0132
```

```
Epoch 81/100
1/1 [==============================] - 0s 38ms/step - loss: 7331.5894 - val_loss: 715
5.9917
Epoch 82/100
1/1 [==============================] - 0s 31ms/step - loss: 7324.8926 - val_loss: 713
4.7827
Epoch 83/100
1/1 [==============================] - 0s 36ms/step - loss: 7318.0898 - val_loss: 711
2.6753
Epoch 84/100
1/1 [==============================] - 0s 38ms/step - loss: 7311.2383 - val_loss: 708
9.8604
Epoch 85/100
1/1 [==============================] - 0s 35ms/step - loss: 7304.4062 - val_loss: 706
6.9990
Epoch 86/100
1/1 [==============================] - 0s 34ms/step - loss: 7297.4590 - val_loss: 704
5.1914
Epoch 87/100
1/1 [==============================] - 0s 52ms/step - loss: 7290.3975 - val_loss: 702
3.1465
Epoch 88/100
1/1 [==============================] - 0s 38ms/step - loss: 7283.2246 - val_loss: 700
0.8652
Epoch 89/100
1/1 [==============================] - 0s 35ms/step - loss: 7275.9375 - val_loss: 697
8.3530
Epoch 90/100
1/1 [==============================] - 0s 38ms/step - loss: 7268.5581 - val_loss: 695
5.6021
Epoch 91/100
1/1 [==============================] - 0s 36ms/step - loss: 7261.0420 - val_loss: 693
2.6216
Epoch 92/100
1/1 [==============================] - 0s 33ms/step - loss: 7253.4316 - val_loss: 690
9.4131
Epoch 93/100
1/1 [==============================] - 0s 42ms/step - loss: 7245.7061 - val_loss: 688
5.9771
Epoch 94/100
1/1 [==============================] - 0s 42ms/step - loss: 7237.8652 - val_loss: 686
2.3135
Epoch 95/100
1/1 [==============================] - 0s 40ms/step - loss: 7229.9092 - val_loss: 683
8.4224
Epoch 96/100
1/1 [==============================] - 0s 36ms/step - loss: 7221.8379 - val_loss: 681
4.3042
Epoch 97/100
1/1 [==============================] - 0s 38ms/step - loss: 7213.6489 - val_loss: 678
9.9575
Epoch 98/100
1/1 [==============================] - 0s 39ms/step - loss: 7205.3428 - val_loss: 676
5.3838
Epoch 99/100
1/1 [==============================] - 0s 40ms/step - loss: 7196.9180 - val_loss: 674
0.5825
Epoch 100/100
1/1 [==============================] - 0s 55ms/step - loss: 7188.3740 - val_loss: 671
5.5532
```

```
1/1 [==============================] - 0s 26ms/step - loss: 6715.5532
Test loss: 6715.55322265625
```



In [43]:
```
#9
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score, confusion_m

text_data = [
    "I love this sandwich, pos",
    "This is an amazing place, pos",
    "I feel very good about these cheese, pos",
    "This is my best work, pos",
    "What an awesome view, pos",
    "I do not like this restaurant, neg",
    "I am tired of this stuff, neg",
    "I can't deal with this, neg",
    "He is my sworn enemy, neg",
    "My boss is horrible, neg",
    "This is an awesome place, pos",
    "I do not like the taste of this juice, neg",
    "I love to dance, pos",
    "I am sick and tired of this place, neg",
    "What a great holiday, pos",
    "That is a bad locality to stay, neg",
    "We will have good fun tomorrow, pos",
    "I went to my enemy's house today, neg"
]

labels = ['pos', 'pos', 'pos', 'pos', 'pos', 'neg', 'neg', 'neg', 'neg', 'neg', 'pos',
```

```
df = pd.DataFrame({'text': text_data, 'label': labels})


X = df['text']
y = df['label']


vectorizer = CountVectorizer()
X_vectorized = vectorizer.fit_transform(X)



X_train, X_test, y_train, y_test = train_test_split(X_vectorized, y, test_size=0.2, ra


classifier = MultinomialNB()
classifier.fit(X_train, y_train)


y_pred = classifier.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
recall = recall_score(y_test, y_pred, pos_label='pos')
precision = precision_score(y_test, y_pred, pos_label='pos')
confusion_mat = confusion_matrix(y_test, y_pred)

print("Total Instances of Dataset:", len(df))
print("Accuracy:", accuracy)
print("Recall:", recall)
print("Precision:", precision)
print("Confusion Matrix:")
print(confusion_mat)
```

```
Total Instances of Dataset: 18
Accuracy: 1.0
Recall: 1.0
Precision: 1.0
Confusion Matrix:
[[2 0]
 [0 2]]
```

In [44]:

```
#10
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans


data = {
    'VAR1': [1.713, 0.180, 0.353, 0.940, 1.486, 1.266, 1.540, 0.459, 0.773],
    'VAR2': [1.586, 1.786, 1.240, 1.566, 0.759, 1.106, 0.419, 1.799, 0.186],
    'CLASS': [0, 1, 1, 0, 1, 0, 1, 1, 1]
}

df = pd.DataFrame(data)

X = df[['VAR1', 'VAR2']]

kmeans_model = KMeans(n_clusters=3, random_state=42)
```

```
kmeans_model.fit(X)

new_case = np.array([[0.906, 0.606]])
predicted_cluster = kmeans_model.predict(new_case)

print("Predicted cluster for the new case:", predicted_cluster[0])
```

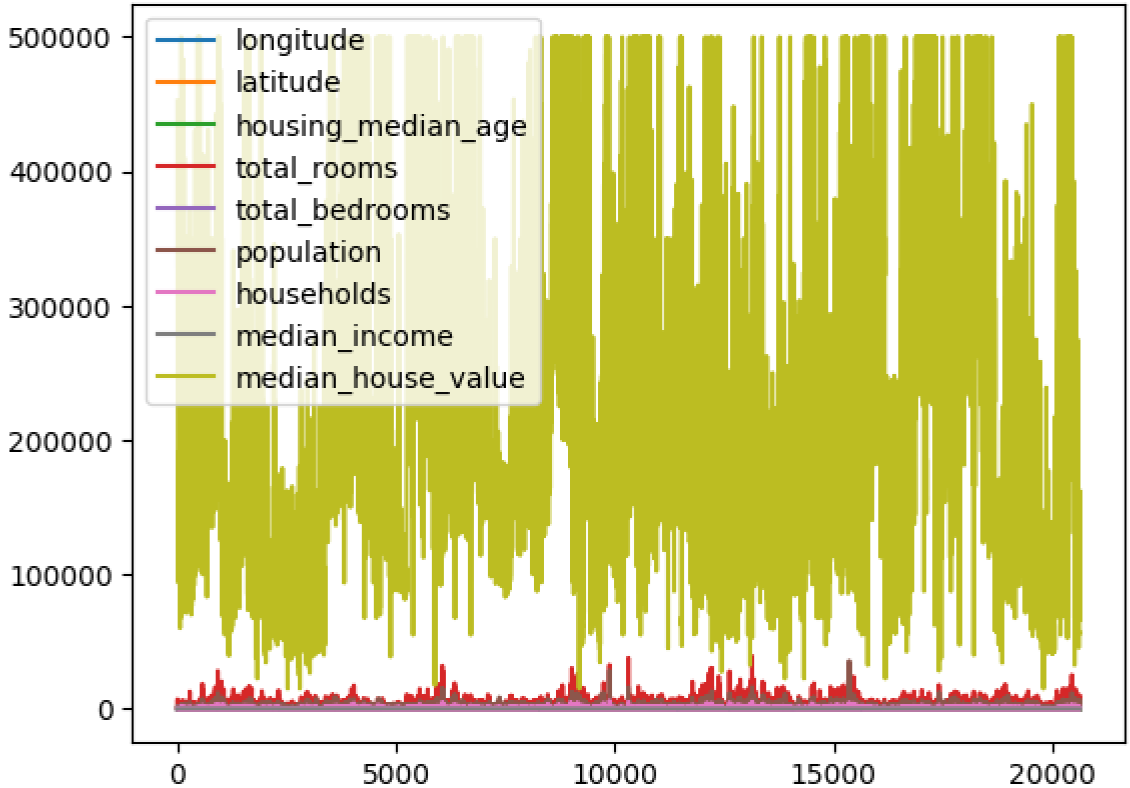
```
Predicted cluster for the new case: 1
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarnin
g: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value
of `n_init` explicitly to suppress the warning
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not
have valid feature names, but KMeans was fitted with feature names
  warnings.warn(
```

In [5]:

```
#11
from sklearn.ensemble import RandomForestRegressor
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

path="/content/housing.csv"

#import datset

data=pd.read_csv(path)
print(data.head()) #displaying 5 rows of data


count=data.count()
print(count)

#to print number of null values
print(data.isnull().sum())

#graph representation
data.plot()
plt.show()

#cov matrix and corr matrix
cov_mat=data.cov(numeric_only=True)
corr_mat=data.corr(numeric_only=True)
print(cov_mat)
print(corr_mat)

#train and test model

X=data.drop(["median_house_value"],axis=1)
y=data["median_house_value"]
X_encoded = pd.get_dummies(X, columns=['ocean_proximity'])
X_encoded.fillna(data["total_bedrooms"].mean(), inplace=True)

X_train,X_test,y_train,y_test=train_test_split(X_encoded,y,test_size=0.09,random_state

model=RandomForestRegressor()
```

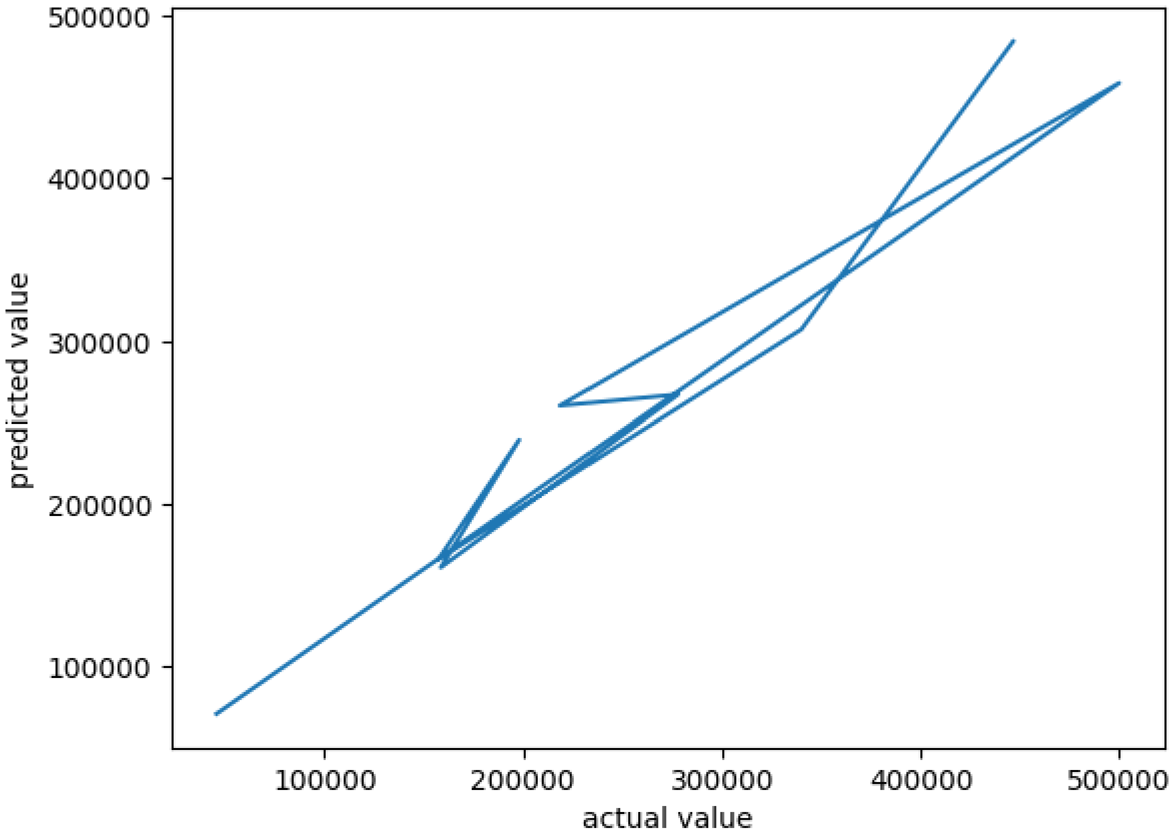
```
model.fit(X_train,y_train)

#predicting values

y_pred=model.predict(X_test)
print(y_pred)


#accuracy and its graph
mse = mean_squared_error(y_test, y_pred)
a = 1 - (mse / np.var(y_test))

print(f"the accuracy of the model is : {a}")

plt.plot(y_test[1:10],y_pred[1:10])
plt.xlabel("actual value")
plt.ylabel("predicted value")
plt.show()
```

```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0   
1    -122.22     37.86                21.0       7099.0          1106.0   
2    -122.24     37.85                52.0       1467.0           190.0   
3    -122.25     37.85                52.0       1274.0           235.0   
4    -122.25     37.85                52.0       1627.0           280.0   

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
longitude             20640
latitude              20640
housing_median_age    20640
total_rooms           20640
total_bedrooms        20433
population            20640
households            20640
median_income         20640
median_house_value    20640
ocean_proximity       20640
dtype: int64
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```


longitude
latitude
housing_median_age
total_rooms
total_bedrooms
population
households
median_income
median_house_value
500000
400000
300000
200000
100000
0
0
5000
10000
15000
20000

```
                    longitude      latitude  housing_median_age  \
longitude            4.014139     -3.957054           -2.728244
latitude            -3.957054      4.562293            0.300346
housing_median_age  -2.728244      0.300346          158.396260
total_rooms        194.803750   -168.217847        -9919.120060
total_bedrooms      58.768508    -60.299623        -1700.312817
population         226.377839   -263.137814        -4222.270582
households          42.368072    -58.010245        -1457.581290
median_income       -0.057765     -0.323860           -2.846140
median_house_value -10627.425205 -35532.559074      153398.801329

                    total_rooms  total_bedrooms    population    households  \
longitude          1.948037e+02    5.876851e+01  2.263778e+02  4.236807e+01
latitude          -1.682178e+02   -6.029962e+01 -2.631378e+02 -5.801024e+01
housing_median_age -9.919120e+03   -1.700313e+03 -4.222271e+03 -1.457581e+03
total_rooms        4.759445e+06    8.567306e+05  2.117613e+06  7.661046e+05
total_bedrooms     8.567306e+05    1.775654e+05  4.191391e+05  1.578295e+05
population         2.117613e+06    4.191391e+05  1.282470e+06  3.928036e+05
households         7.661046e+05    1.578295e+05  3.928036e+05  1.461760e+05
median_income      8.208524e+02   -6.180851e+00  1.040098e+01  9.466667e+00
median_house_value  3.377289e+07    2.416878e+06 -3.221249e+06  2.904924e+06

                    median_income  median_house_value
longitude               -0.057765       -1.062743e+04
latitude                -0.323860       -3.553256e+04
housing_median_age      -2.846140        1.533988e+05
total_rooms            820.852410        3.377289e+07
total_bedrooms          -6.180851        2.416878e+06
population              10.400979       -3.221249e+06
households               9.466667        2.904924e+06
median_income            3.609323        1.508475e+05
median_house_value  150847.482793        1.331615e+10
                    longitude  latitude  housing_median_age  total_rooms  \
longitude            1.000000 -0.924664           -0.108197     0.044568
latitude            -0.924664  1.000000            0.011173    -0.036100
housing_median_age  -0.108197  0.011173            1.000000    -0.361262
total_rooms          0.044568 -0.036100           -0.361262     1.000000
total_bedrooms       0.069608 -0.066983           -0.320451     0.930380
population           0.099773 -0.108785           -0.296244     0.857126
households           0.055310 -0.071035           -0.302916     0.918484
median_income       -0.015176 -0.079809           -0.119034     0.198050
median_house_value  -0.045967 -0.144160            0.105623     0.134153

                    total_bedrooms  population  households  median_income  \
longitude                 0.069608    0.099773    0.055310      -0.015176
latitude                 -0.066983   -0.108785   -0.071035      -0.079809
housing_median_age       -0.320451   -0.296244   -0.302916      -0.119034
total_rooms               0.930380    0.857126    0.918484       0.198050
total_bedrooms            1.000000    0.877747    0.979728      -0.007723
population                0.877747    1.000000    0.907222       0.004834
households                0.979728    0.907222    1.000000       0.013033
median_income            -0.007723    0.004834    0.013033       1.000000
median_house_value        0.049686   -0.024650    0.065843       0.688075

                    median_house_value
longitude                    -0.045967
latitude                     -0.144160
housing_median_age            0.105623
total_rooms                   0.134153
total_bedrooms                0.049686
```

```
population                    -0.024650
households                     0.065843
median_income                  0.688075
median_house_value             1.000000
[ 53116.    70797.   458646.31 ... 207549.   122466.   201320.  ]
the accuracy of the model is : 0.8129270752484333
```



In [1]:
```
#12
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


path= "/content/HEART DISEASE.csv"  # Replace with the actual path
data = pd.read_csv(path)


X = data.drop('target', axis=1)
y = data['target']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Step 4: Build the Naive Bayes classifier (Gaussian Naive Bayes)
naive_bayes_model = GaussianNB()
naive_bayes_model.fit(X_train, y_train)

y_pred = naive_bayes_model.predict(X_test)


accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)
```

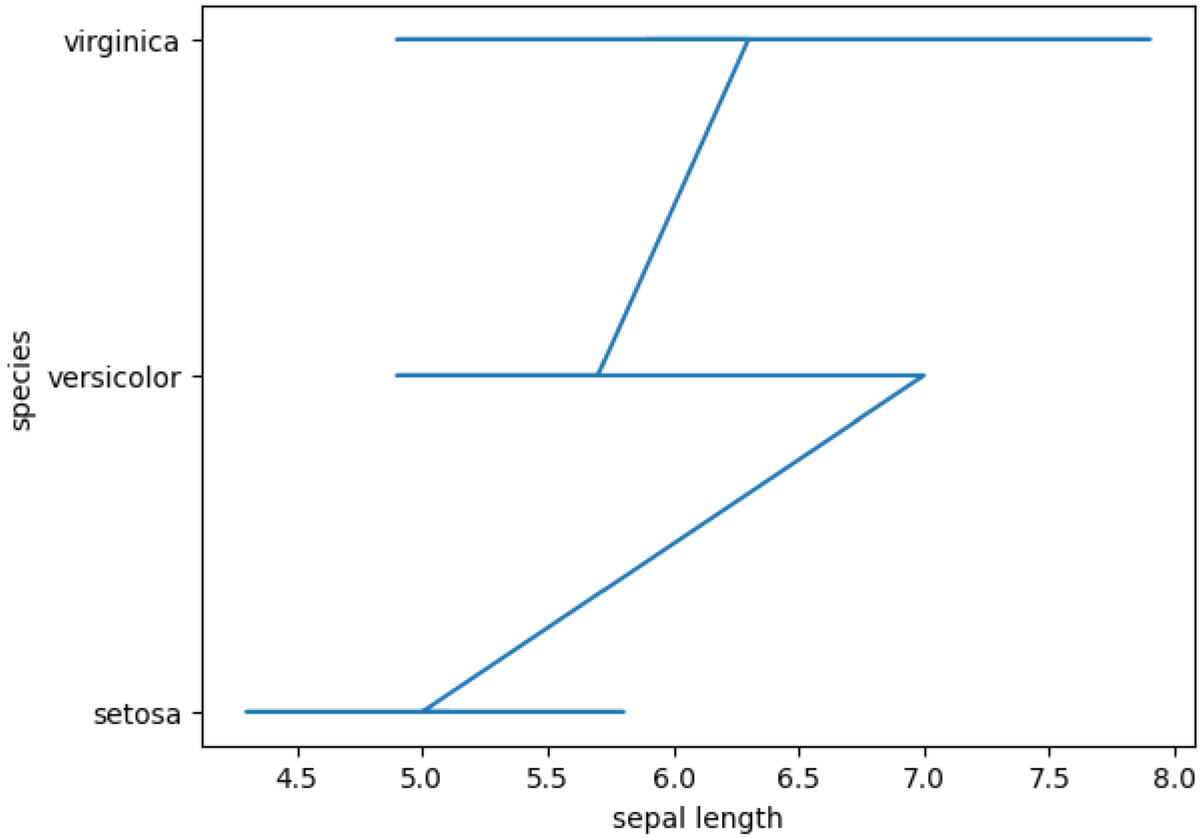
```
confusion_mat = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_rep)
print("Confusion Matrix:")
print(confusion_mat)
```

```
Accuracy: 0.8688524590163934
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.90      0.87        29
           1       0.90      0.84      0.87        32

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.87      0.87      0.87        61

Confusion Matrix:
[[26  3]
 [ 5 27]]
```

In [9]:

```
#13
from sklearn.neighbors import KNeighborsClassifier
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

path="/content/IRIS.csv"

data=pd.read_csv(path)
print(data.head())

x=data["Sepal.Length"]
y=data["Species"]
plt.xlabel("sepal length")
plt.ylabel("species")
plt.plot(x,y)
plt.show()

X=data.drop(["Species"],axis=1)
y=data["Species"]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=15)

model=KNeighborsClassifier()
model.fit(X,y)

y_pred=model.predict(X_test)
print(y_pred)

a=accuracy_score(y_test,y_pred)
print(f"the accuracy is : {a}")
```

```
   ID  Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
0   1           5.1          3.5           1.4          0.2  setosa
1   2           4.9          3.0           1.4          0.2  setosa
2   3           4.7          3.2           1.3          0.2  setosa
3   4           4.6          3.1           1.5          0.2  setosa
4   5           5.0          3.6           1.4          0.2  setosa
```




```
['setosa' 'versicolor' 'versicolor' 'setosa' 'setosa' 'versicolor'
 'virginica' 'versicolor' 'versicolor' 'virginica' 'virginica'
 'versicolor' 'versicolor' 'versicolor' 'virginica' 'setosa' 'versicolor'
 'virginica' 'setosa' 'virginica' 'versicolor' 'setosa' 'versicolor'
 'versicolor' 'setosa' 'setosa' 'virginica' 'virginica' 'virginica'
 'versicolor']
the accuracy is : 1.0
```

In [15]:

```
#14
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Generate sample data with a known curve (linear or non-linear)
# For this example, we'll use a non-linear curve y = sin(x) + noise
np.random.seed(42)
X = np.sort(5 * np.random.rand(80, 1), axis=0)
y = np.sin(X).ravel() + 0.1 * np.random.randn(80)

# Step 2: Set the value for the smoothing parameter (t)
t = 0.1

# Step 3: Set the bias/point of interest (x0)
x0 = 2.5

# Step 4: Determine the weight matrix using Gaussian Kernel
weights = np.exp(-0.5 * ((X - x0) / t) ** 2)
print(weights)
```

```
# Step 5: Determine the value of model term parameter B using Locally Weighted Regress
X_with_bias = np.hstack([np.ones((X.shape[0], 1)), X])
W = np.diag(weights.ravel())
B = np.linalg.inv(X_with_bias.T @ W @ X_with_bias) @ X_with_bias.T @ W @ y

# Step 6: Prediction
x0_with_bias = np.array([1, x0])
prediction = x0_with_bias @ B

print("Prediction at x0:", prediction)
```

```
[[1.83739085e-133]
 [1.68894274e-125]
 [2.03904057e-118]
 [5.30867391e-113]
 [2.12866551e-112]
 [9.62137954e-107]
 [1.99594700e-103]
 [3.18507492e-099]
 [5.45744491e-099]
 [1.17916018e-092]
 [1.34021275e-088]
 [7.87480360e-081]
 [2.80882811e-078]
 [2.79461952e-071]
 [1.01175986e-070]
 [5.71628144e-065]
 [5.83609198e-065]
 [1.17301710e-059]
 [1.10295258e-055]
 [3.86244205e-055]
 [1.21373486e-054]
 [6.67797511e-051]
 [5.28014710e-050]
 [1.08531813e-049]
 [1.19763063e-045]
 [2.58274137e-032]
 [4.15115292e-029]
 [8.87010118e-027]
 [2.18245929e-024]
 [3.51568124e-024]
 [1.57288679e-021]
 [1.88619761e-021]
 [5.67348334e-020]
 [2.73759526e-017]
 [7.25368307e-012]
 [1.33442238e-011]
 [2.01759210e-010]
 [2.85205165e-009]
 [1.87229203e-007]
 [3.05996865e-003]
 [1.13657025e-002]
 [8.96082526e-002]
 [9.71340945e-001]
 [7.76255123e-001]
 [6.04467821e-001]
 [4.64821818e-001]
 [1.02417837e-001]
 [6.53947862e-002]
 [2.31025935e-005]
 [6.26515447e-006]
 [5.19990277e-006]
 [2.81567933e-006]
 [5.26313411e-007]
 [1.61487585e-007]
 [4.58096766e-015]
 [3.75037822e-019]
 [5.89787049e-024]
 [3.14022732e-024]
 [3.38588029e-029]
 [6.05610281e-030]
```

```
 [1.12626008e-040]
 [5.80936227e-041]
 [8.05252569e-042]
 [7.09697040e-045]
 [2.65245514e-050]
 [2.33560143e-052]
 [9.46205000e-055]
 [2.15347557e-059]
 [1.00749676e-060]
 [1.62011903e-073]
 [2.36065022e-085]
 [1.11210433e-091]
 [2.40708459e-097]
 [1.38027840e-105]
 [4.10242509e-110]
 [5.24729560e-111]
 [1.99083310e-118]
 [1.96022227e-120]
 [1.33793143e-120]
 [2.03371812e-129]]
Prediction at x0: 0.5051913502645387
```