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\_stat

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()

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | longitude  -122.23 | latitude  37.88 | housing\_median\_age  41.0 | total\_rooms  880.0 | total\_bedrooms \  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  <class 'pandas.core.frame.DataFrame'> | | | | 342200.0 | | NEAR BAY |

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

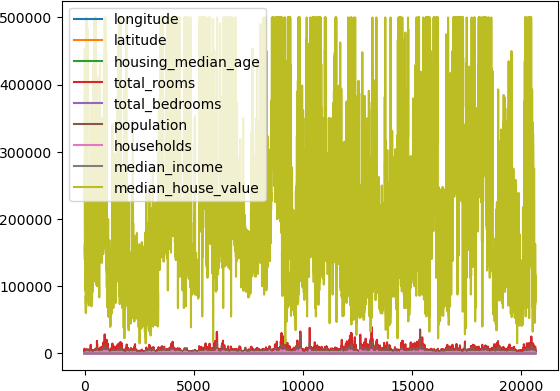
households 0

median\_income 0

median\_house\_value 0

ocean\_proximity 0

dtype: int64



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| longitude | longitude  4.014139 | latitude  -3.957054 | housing\_median\_age  -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 | |  |
| longitude | total\_rooms 1.948037e+02 | total\_bedrooms 5.876851e+01 | population 2.263778e+02 | households \ 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 | longitude  1.000000 | latitude  -0.924664 | | housing\_median\_age  -0.108197 | | total\_rooms  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 |  |

|  |  |
| --- | --- |
| longitude | median\_house\_value  -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 |

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*

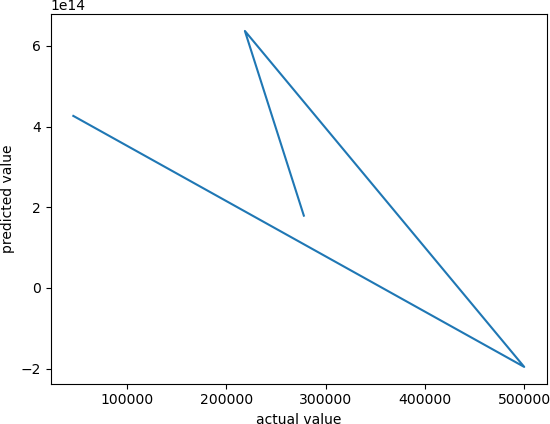
plt**.**show()

cov\_mat**=**iris**.**cov() print(cov\_mat)

[ 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



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

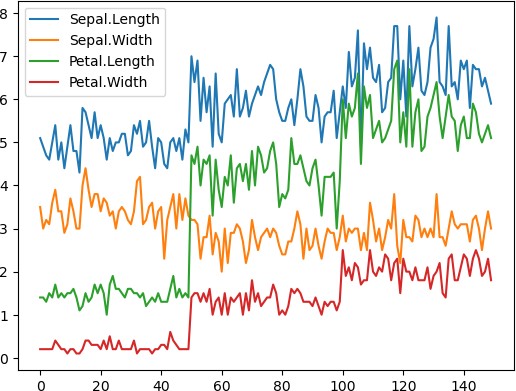
Sepal.Width 0

Petal.Length 0

Petal.Width 0

Species 0

dtype: int64



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sepal.Length | Sepal.Length  0.685694 | Sepal.Width  -0.042434 | Petal.Length  1.274315 | Petal.Width  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  Sepal.Length | -0.121639  Sepal.Width | 1.295609  Petal.Length | 0.581006  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]:

*#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()*

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)

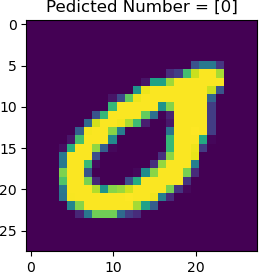
**return** hypothesis

training\_data **=** []

**with** open('enjoysport.csv.csv', 'r') **as** file: csv\_reader **=** csv**.**reader(file)

7

the accuracy is : 0.9661904761904762



**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, rand decision\_tree **=** DecisionTreeClassifier(criterion**=**'entropy')

decision\_tree**.**fit(X\_train, y\_train)

new\_sample **=** X\_test**.**iloc[[0]]

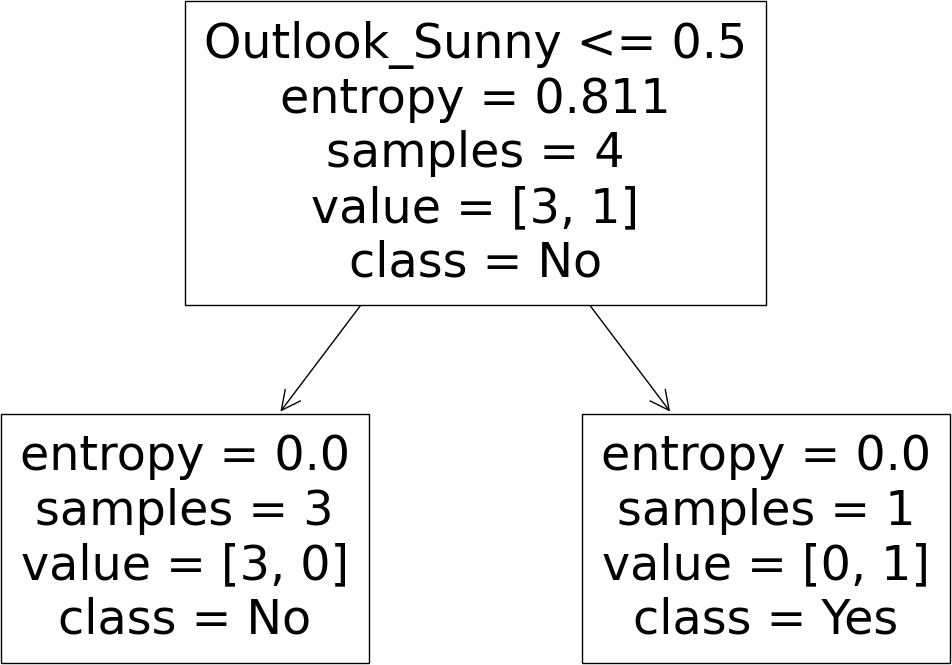
predicted\_class **=** decision\_tree**.**predict(new\_sample)

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\_t 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()

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 [==============================]  3.8335  Epoch 25/100  1/1 [==============================] | * 0s * 0s | 48ms/step  64ms/step | * loss: * loss: | 7601.6201  7597.2471 | * val\_loss: * val\_loss: | 808  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 [==============================]  9.8950  Epoch 29/100  1/1 [==============================] | * 0s * 0s | 41ms/step  51ms/step | * loss: * loss: | 7584.5078  7580.3765 | * val\_loss: * val\_loss: | 802  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 [==============================]  6.7471  Epoch 33/100  1/1 [==============================] | * 0s * 0s | 37ms/step  36ms/step | * loss: * loss: | 7568.2969  7564.9126 | * val\_loss: * val\_loss: | 797  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 [==============================]  2.3892  Epoch 37/100  1/1 [==============================] | * 0s * 0s | 40ms/step  56ms/step | * loss: * loss: | 7554.3379  7550.6963 | * val\_loss: * val\_loss: | 792  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 [==============================]  6.6704 | - 0s | 38ms/step | - loss: | 7539.4731 | - val\_loss: | 786 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 [==============================]  9.3125  Epoch 45/100  1/1 [==============================] | * 0s * 0s | 56ms/step  38ms/step | * loss: * loss: | 7523.8193  7519.7783 | * val\_loss: * val\_loss: | 780  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 [==============================]  0.0273  Epoch 49/100  1/1 [==============================] | * 0s * 0s | 41ms/step  37ms/step | * loss: * loss: | 7507.3350  7503.0762 | * val\_loss: * val\_loss: | 775  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 [==============================]  8.4634  Epoch 53/100  1/1 [==============================] | * 0s * 0s | 43ms/step  45ms/step | * loss: * loss: | 7489.9648  7485.4819 | * val\_loss: * val\_loss: | 768  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 [==============================]  4.5132  Epoch 57/100  1/1 [==============================] | * 0s * 0s | 40ms/step  38ms/step | * loss: * loss: | 7471.7402  7467.0645 | * val\_loss: * val\_loss: | 762  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 [==============================]  7.8013 | - 0s | 38ms/step | - loss: | 7452.6113 | - val\_loss: | 755 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 [==============================]  8.0298  Epoch 65/100  1/1 [==============================] | * 0s * 0s | 37ms/step  39ms/step | * loss: * loss: | 7432.3281  7427.0674 | * val\_loss: * val\_loss: | 748  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 [==============================]  5.1226  Epoch 69/100  1/1 [==============================] | * 0s * 0s | 48ms/step  41ms/step | * loss: * loss: | 7410.7852  7405.1855 | * val\_loss: * val\_loss: | 741  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 [==============================]  8.9380  Epoch 73/100  1/1 [==============================] | * 0s * 0s | 34ms/step  33ms/step | * loss: * loss: | 7387.8545  7381.8955 | * val\_loss: * val\_loss: | 733  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 [==============================]  9.3672  Epoch 77/100  1/1 [==============================] | * 0s * 0s | 34ms/step  34ms/step | * loss: * loss: | 7363.5820  7357.3340 | * val\_loss: * val\_loss: | 725  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 [==============================]  7.0132 | - 0s | 42ms/step | - loss: | 7338.1802 | - val\_loss: | 717 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 [==============================]  9.8604  Epoch 85/100  1/1 [==============================] | * 0s * 0s | 38ms/step  35ms/step | * loss: * loss: | 7311.2383  7304.4062 | * val\_loss: * val\_loss: | 708  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 [==============================]  0.8652  Epoch 89/100  1/1 [==============================] | * 0s * 0s | 38ms/step  35ms/step | * loss: * loss: | 7283.2246  7275.9375 | * val\_loss: * val\_loss: | 700  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 [==============================]  9.4131  Epoch 93/100  1/1 [==============================] | * 0s * 0s | 33ms/step  42ms/step | * loss: * loss: | 7253.4316  7245.7061 | * val\_loss: * val\_loss: | 690  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 [==============================]  4.3042  Epoch 97/100  1/1 [==============================] | * 0s * 0s | 36ms/step  38ms/step | * loss: * loss: | 7221.8379  7213.6489 | * val\_loss: * val\_loss: | 681  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 [==============================]  5.5532 | - 0s | 55ms/step | - loss: | 7188.3740 | - val\_loss: | 671 |

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\_

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",

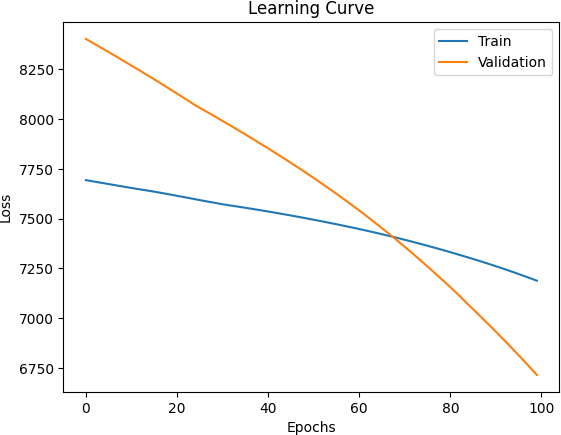
"What a great holiday, pos",

"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'

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



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, r

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,  'CLASS': [0, 1, 1, 0, 1, 0, 1, 1, 1] | 1.106, 0.419, | 1.799, | 0.186], |
|  | }  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\_stat 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

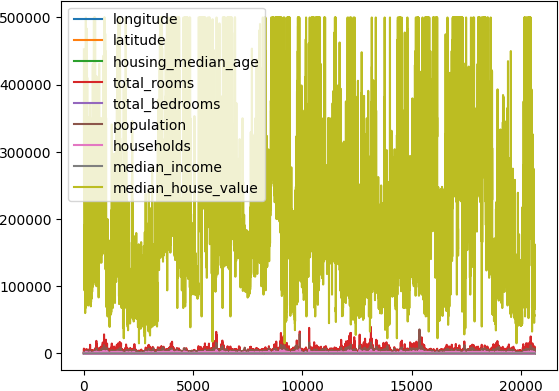
households 0

median\_income 0

median\_house\_value 0

ocean\_proximity 0

dtype: int64



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| longitude | longitude  4.014139 | latitude  -3.957054 | housing\_median\_age  -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 | |  |
| longitude | total\_rooms 1.948037e+02 | total\_bedrooms 5.876851e+01 | population 2.263778e+02 | households \ 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 | longitude  1.000000 | latitude  -0.924664 | | housing\_median\_age  -0.108197 | | total\_rooms  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 |  |

|  |  |
| --- | --- |
| longitude | median\_house\_value  -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 |

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)

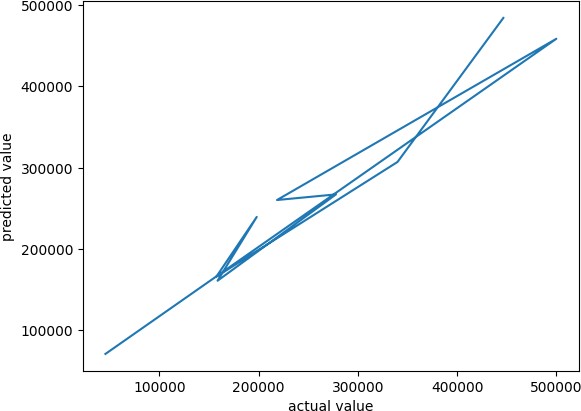
y\_pred **=** naive\_bayes\_model**.**predict(X\_test)

accuracy **=** accuracy\_score(y\_test, y\_pred)

classification\_rep **=** classification\_report(y\_test, y\_pred)

[ 53116. 70797. 458646.31 ... 207549. 122466. 201320. ]

the accuracy of the model is : 0.8129270752484333



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}")

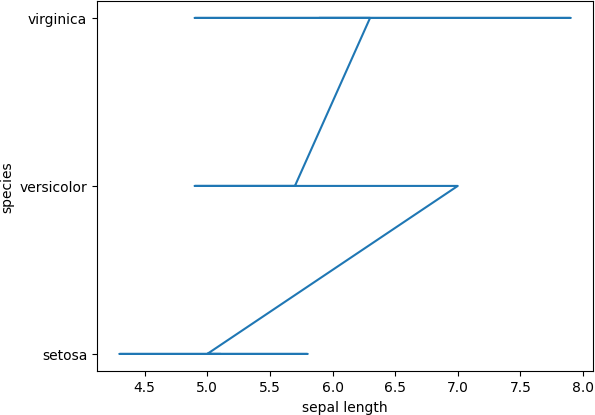
ID Sepal.Length Sepal.Width Petal.Length Petal.Width Species 0 1 5.1 3.5 1.4 0.2 setosa

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 Regres*

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

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report

iris = load\_iris()

data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

data['target'] = iris.target

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=42)

svm\_classifier = SVC(kernel='linear', C=1.0, random\_state=42) # You can adjust the kernel and C value

svm\_classifier.fit(X\_train, y\_train)

predictions = svm\_classifier.predict(X\_test)

accuracy = accuracy\_score(y\_test, predictions)

report = classification\_report(y\_test, predictions)

print(f"Accuracy: {accuracy}")

print("Classification Report:")

print(report)