|  |  |
| --- | --- |
| Ex no: 1 | **Data Preprocessing** |
| **Aim:**  To preprocess the given dataset to proceed with machine learning.  **Dataset Description:**  For this experiment we will go with the ‘health’ dataset which has the details of patients about their temperature, chronic disease etc.  **Sample Dataset:**  Health.csv  age temperature chronic\_disease breathing\_issue O2\_level needed\_hospital  10 Normal no no 97 No  12 Normal no no 97 No  15 Normal no no 94 No  10 Normal no no 97 No  13 Moderate no no 94 No  **Code:**  import numpy as np import pandas as pd  import matplotlib.pyplot as plt  #reading dataset  read\_file = pd.read\_excel ("health.xlsx") read\_file.to\_csv ("health.csv",  index = None, header=True)  df = pd.DataFrame(pd.read\_csv("health.csv")) data\_set=pd.read\_csv("health.csv") x=data\_set[['age','temperature','chronic\_disease','breathing\_issue','O2\_level ']].values  y=data\_set[['needed\_hospitalization']].values  #filling missing data  from sklearn.impute import SimpleImputer imputer=SimpleImputer(missing\_values=np.nan,strategy='mean') imputer=imputer.fit(x[:,0:5:4]) x[:,0:5:4]=imputer.transform(x[:,0:5:4])  #Encoding | |

from sklearn.compose import ColumnTransformer from sklearn.preprocessing import OneHotEncoder

ct=ColumnTransformer(transformers=[('encoder',OneHotEncoder(), [1])],remainder='passthrough')

x=np.array(ct.fit\_transform(x)) print(x)

from sklearn.preprocessing import LabelEncoder le=LabelEncoder()

y=le.fit\_transform(y) print(y)

#test train data split

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

x\_train , x\_test , y\_train , y\_test=train\_test\_split(x,y,test\_size=0.3) from sklearn.preprocessing import StandardScaler sc=StandardScaler()

x\_train[:,6:]=sc.fit\_transform(x\_train[:,6:]) x\_test[:,6:]=sc.fit\_transform(x\_test[:,6:])

**Output:**

X:

[[0.0 0.0 1.0 10.0 'no' 'no' 97.0]

[0.0 0.0 1.0 12.0 'no' 'no' 97.0]

[0.0 0.0 1.0 15.0 'no' 'no' 94.0]

[0.0 0.0 1.0 10.0 'no' 'no' 97.0]

[0.0 1.0 0.0 13.0 'no' 'no' 94.0]]

Y:

[0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0

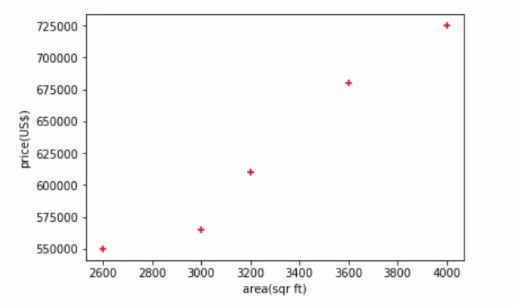
1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1]

**Result:**

Thus the given health dataset was successfully preprocessed and split into test and training datasets.

|  |  |
| --- | --- |
| Ex no: 2 | **Linear Regression** |
| **Aim:**  To implement Linear Regression to predict values based on the given dataset.  **Dataset Description:**  The given dataset has two columns namely ‘area’ and ‘price’ respectively which represents the area in square feet and its price in dollars.  **Sample Dataset:**  area.csv  **Area Price**  2600 550000  3000 565000  3200 610000  3600 680000  4000 725000  **Code:**  import pandas as pd import numpy as np  import matplotlib.pyplot as plt  from sklearn.model\_selection import train\_test\_split from pandas.core.common import random\_state from sklearn.linear\_model import LinearRegression  df = pd.read\_csv('area.csv')  plt.xlabel('area(sq.ft)') plt.ylabel('prices')  plt.scatter(df.area, df.price, color='red', marker='+')  X = df[['area']] y = df['price']  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42) reg = LinearRegression()  reg.fit(X\_train, y\_train)\ y\_pred = reg.predict(X\_test) new\_area = [[3300]]  predicted\_price = reg.predict(new\_area)  print("Predicted price for 3300 sq.ft. area:", predicted\_price) | |

**Output:**



Predicted price for 3300 sq.ft.area: 628715.75342

**Result:**

Thus linear regression was successfully implemented on the ‘area’ dataset to predict prices for given areas.

|  |  |
| --- | --- |
| Ex no: 3 | **Multivariate Regression** |
| **Aim:**  To implement Multivariate Regression to predict values based on the given dataset.  **Dataset Description:**  The given dataset has columns namely ‘area’ and ‘bedroom’, ‘age’ and ‘price’ which represents the area in square feet and the amount of bedrooms along with the tenants age and the rooms' respective prices..  **Sample Dataset:**  homeprices.csv  **area bedrooms age price**  2600 3.0 20 550000  3000 4.0 15 565000  3200 NaN 18 610000  3600 3.0 30 595000  4000 5.0 8 760000  4100 6.0 8 810000  **Code:**  import pandas as pd import numpy as np  from sklearn import linear\_model  # Read the CSV file  df = pd.read\_csv('homeprices.csv')  # Fill the NaN values in the 'bedrooms' column with the median df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())  # Training the linear regression model reg = linear\_model.LinearRegression()  reg.fit(df.drop('price', axis='columns'), df.price)  # Display coefficients and intercept print("Coefficients:", reg.coef\_) print("Intercept:", reg.intercept\_)  # Make predictions  prediction1 = reg.predict([[3000, 3, 40]])  prediction2 = reg.predict([[2500, 4, 5]])  # Display predictions | |

print("Prediction for [3000, 3, 40]:", prediction1)

print("Prediction for [2500, 4, 5]:", prediction2)

**Output:**

Coefficients: [ 112.06244194, 23388.88007794, -3231.71790863]

Intercept: 221323.001865

Prediction for [3000, 3, 40]: 498408.25158

Prediction for [2500, 4, 5]: 578876.03748933

**Result:**

Thus Multivariate regression was successfully implemented on the ‘homeprices.csv’ dataset for prediction.

|  |  |
| --- | --- |
| Ex no: 4 | **Polynomial Regression** |
| **Aim:**  To implement Polynomial Regression to predict values based on the given dataset.  **Dataset Description:**  The given dataset has columns ‘age’ and ‘height’ to represent the relation between people's ages with their heights.  **Sample Dataset:**  age.csv  **Age Height**  45 155.43  31 147.56  51 156.27  27 155.11  38 163.54  **Code:**  import pandas as pd import numpy as np  import matplotlib.pyplot as plt from sklearn import linear\_model  from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression  # Read the CSV file  df = pd.read\_csv('your\_dataset.csv') # Replace 'your\_dataset.csv' with the actual filename  x = df.iloc[:, 0:1].values y = df.iloc[:, 1].values  # Splitting the data into training and testing sets  x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=0)  # Linear Regression Model linreg = LinearRegression() linreg.fit(x\_train, y\_train)  # Plotting Linear Regression plt.scatter(x\_train, y\_train, color='green')  plt.plot(x\_train, linreg.predict(x\_train), color='blue') plt.title('Linear Regression') | |

plt.xlabel('Age') plt.ylabel('Height') plt.show()

# Polynomial Regression Model

poly = PolynomialFeatures(degree=2) x\_poly = poly.fit\_transform(x\_train) polyreg = LinearRegression() polyreg.fit(x\_poly, y\_train)

# Plotting Polynomial Regression

x\_val = np.linspace(min(x\_train), max(x\_train), 100).reshape(-1, 1) x\_val\_poly = poly.transform(x\_val)

y\_pred = polyreg.predict(x\_val\_poly) plt.scatter(x\_train, y\_train, color='green') plt.plot(x\_val, y\_pred, color='blue') plt.title('Polynomial Regression (degree=2)') plt.xlabel('Age')

plt.ylabel('Height') plt.show()

# Calculating R^2 for Simple Linear Regression y\_predict\_slr = linreg.predict(x\_test)

r\_square\_slr = metrics.r2\_score(y\_test, y\_predict\_slr) print("R^2 for Simple Linear Regression:", r\_square\_slr)

# Calculating R^2 for Polynomial Regression y\_predict\_pr = polyreg.predict(poly.fit\_transform(x\_test)) r\_square\_pr = metrics.r2\_score(y\_test, y\_predict\_pr) print("R^2 for Polynomial Regression:", r\_square\_pr)

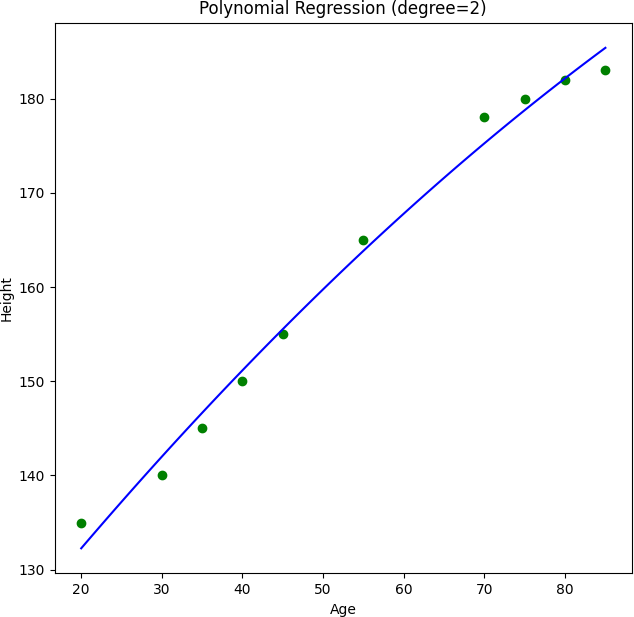
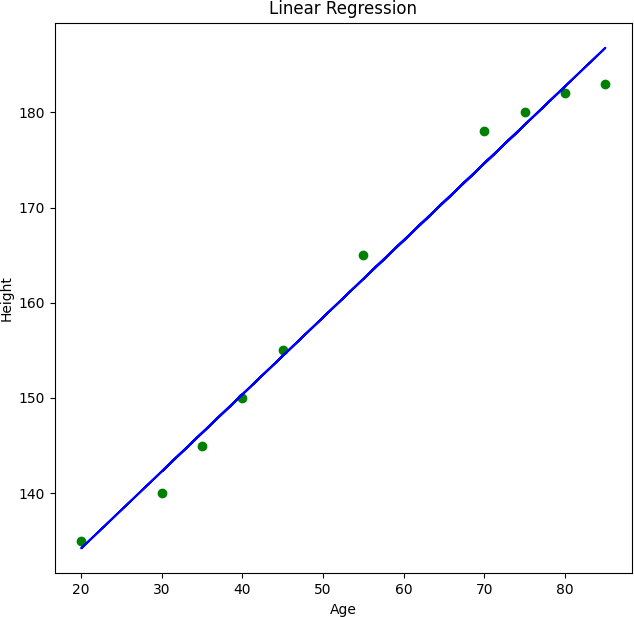
# Making predictions for a given value using both models slr\_prediction = linreg.predict([[53]])

print("Simple Linear Regression prediction for age 53:", slr\_prediction) pr\_prediction = polyreg.predict(poly.transform([[53]])) print("Polynomial Regression prediction for age 53:", pr\_prediction)

**Output:**

R^2 for Simple Linear Regression: 0.9352650699314159 R^2 for Polynomial Regression: 0.9698989897886501

Simple Linear Regression prediction for age 53: [160.89548113] Polynomial Regression prediction for age 53: [162.22590826]

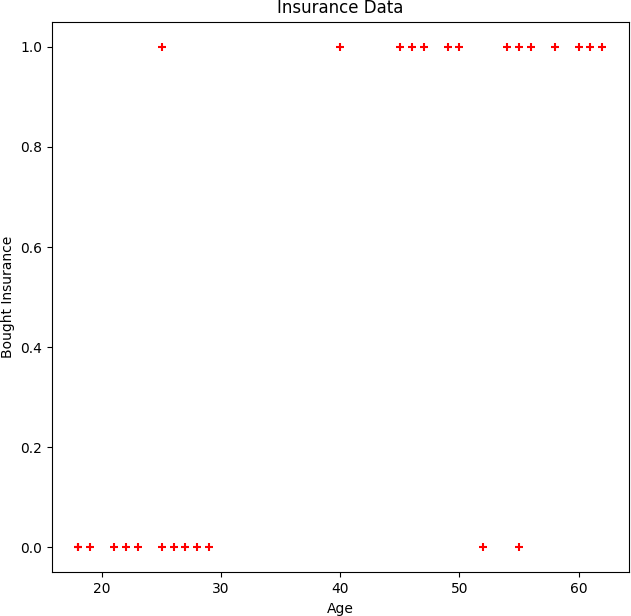


**Result:**

Thus Polynomial regression was successfully implemented on the ‘homeprices.csv’ dataset for prediction.

|  |  |
| --- | --- |
| Ex no: 5 | **Logistic Regression** |
| **Aim:**  To implement Logistic Regression to predict values based on the given dataset.  **Dataset Description:**  The given dataset has columns namely ‘age’ and ‘bought\_insurance’, representing the age of the person and whether they have bought insurance or not.  **Sample Dataset:**  insurance\_data.csv  **age bought\_insurance**  22 0  25 0  47 1  52 0  46 1  **Code:**  import pandas as pd  from matplotlib import pyplot as plt  from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LogisticRegression  # Read the CSV file  df = pd.read\_csv("insurance\_data.csv")  # Visualize the data  plt.scatter(df.age, df.bought\_insurance, marker='+', color='red') plt.xlabel('Age')  plt.ylabel('Bought Insurance') plt.title('Insurance Data') plt.show()  # Splitting the data into training and testing sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(df[['age']], df.bought\_insurance, train\_size=0.8)  # Logistic Regression model model = LogisticRegression() model.fit(X\_train, y\_train)  # Predictions and model evaluation | |

y\_predicted = model.predict(X\_test) predicted\_probabilities = model.predict\_proba(X\_test) accuracy = model.score(X\_test, y\_test)



print("Predicted probabilities:\n", predicted\_probabilities) print("Model accuracy:", accuracy)

**Output:**

Predicted probabilities:

[[0.14887204 0.85112796]

[0.27593066 0.72406934]

[0.07431556 0.92568444]

[0.95625715 0.04374285]

[0.45363364 0.54636636]

[0.03056973 0.96943027]]

Model accuracy: 0.8333333333333334

**Result:**

Thus Logistic regression was successfully implemented on the ‘homeprices.csv’ dataset for prediction.

|  |  |
| --- | --- |
| Ex no: 6 | **K MEANS CLUSTERING** |
| **Aim:**  To prepare the machine learning algorithm to perform k means clustering using the appropriate dataset.  **Dataset Description:**  The dataset contains name, age, salary of the employees which is used to do k means clustering.  **Sample Dataset:**  Income.csv  **Name Age Income($)**  Rob 27 70000  Michael 29 90000  Mohan 29 61000  Ismail 28 60000  Kory 42 150000  **Code:**  from sklearn.cluster import KMeans import pandas as pd  from sklearn.preprocessing import MinMaxScaler from matplotlib import pyplot as plt  df = pd.read\_csv("income.csv")  plt.scatter(df.Age, df['Income($)']) plt.xlabel('Age') plt.ylabel('Income($)')  km = KMeans(n\_clusters=3)  y\_predicted = km.fit\_predict(df[['Age','Income($)']]) df['cluster'] = y\_predicted  km.cluster\_centers\_  df1 = df[df.cluster==0] df2 = df[df.cluster==1] | |

df3 = df[df.cluster==2]

plt.scatter(df1.Age, df1['Income($)'], color='green') plt.scatter(df2.Age, df2['Income($)'], color='red') plt.scatter(df3.Age, df3['Income($)'], color='black')

plt.scatter(km.cluster\_centers\_[:,0], km.cluster\_centers\_[:,1], color='purple', marker='\*', label='centroid')

plt.xlabel('Age') plt.ylabel('Income ($)') plt.legend()

scaler = MinMaxScaler() scaler.fit(df[['Income($)']])

df['Income($)'] = scaler.transform(df[['Income($)']])

scaler.fit(df[['Age']])

df['Age'] = scaler.transform(df[['Age']])

plt.scatter(df.Age, df['Income($)']) km = KMeans(n\_clusters=3)

y\_predicted = km.fit\_predict(df[['Age','Income($)']]) df['cluster'] = y\_predicted

km.cluster\_centers\_

df1 = df[df.cluster==0] df2 = df[df.cluster==1] df3 = df[df.cluster==2]

plt.scatter(df1.Age, df1['Income($)'], color='green') plt.scatter(df2.Age, df2['Income($)'], color='red') plt.scatter(df3.Age, df3['Income($)'], color='black')

plt.scatter(km.cluster\_centers\_[:,0], km.cluster\_centers\_[:,1], color='purple', marker='\*', label='centroid')

plt.legend()

sse = []

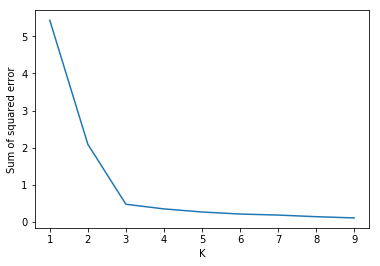
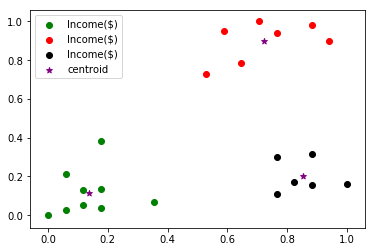
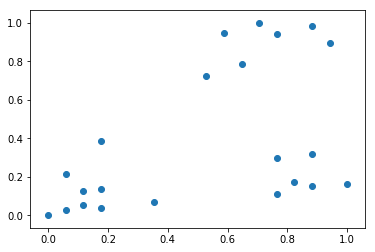
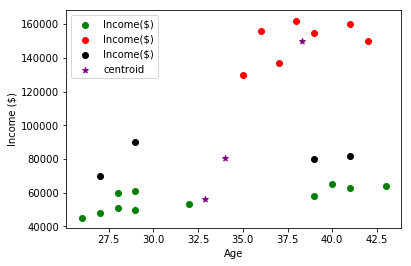
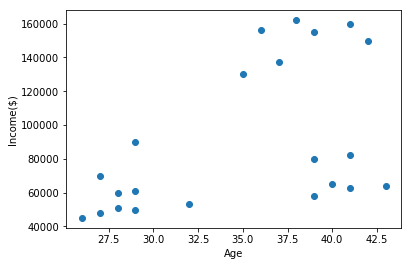
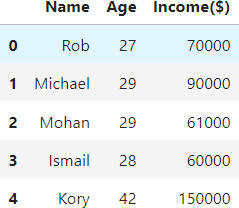
k\_rng = range(1,10) for k in k\_rng:

km = KMeans(n\_clusters=k) km.fit(df[['Age','Income($)']]) sse.append(km.inertia\_)

plt.xlabel('K')

plt.ylabel('Sum of squared error') plt.plot(k\_rng,sse)

**Output:**



**Result:**

Hence the K means clustering algorithm was implemented with the given dataset successfully.

|  |  |
| --- | --- |
| Ex no: 7 | **Decision Trees** |
| **Aim:**  To implement a machine learning algorithm for decision tree using appropriate dataset  **Dataset Description:**  The data set contains the information about employees which describes which company they are in and their position and the main field is salary of the employees more than a certain amount.  **Sample Dataset:**  Salaries.csv  **company job degree salary\_more\_then\_100k**  google sales executive bachelors 0 google sales executive masters 0  google business manager bachelors 1  google business manager masters 1  google computer programmer bachelors 0  **Code:**  import pandas as pd  from sklearn.preprocessing import LabelEncoder from sklearn import tree  # Read the CSV file  df = pd.read\_csv("salaries.csv")  # Separate inputs and target variable  inputs = df.drop('salary\_more\_then\_100k', axis='columns') target = df['salary\_more\_then\_100k']  # Label Encoding for categorical variables le\_company = LabelEncoder()  le\_job = LabelEncoder() le\_degree = LabelEncoder()  inputs['company\_n'] = le\_company.fit\_transform(inputs['company']) inputs['job\_n'] = le\_job.fit\_transform(inputs['job']) inputs['degree\_n'] = le\_degree.fit\_transform(inputs['degree']) | |

inputs\_n = inputs.drop(['company', 'job', 'degree'], axis='columns') # Decision Tree Model

model = tree.DecisionTreeClassifier() model.fit(inputs\_n, target)

# Model score

model\_score = model.score(inputs\_n, target) print("Model Score:", model\_score)

# Predictions

prediction1 = model.predict([[2, 1, 0]])

prediction2 = model.predict([[2, 1, 1]])

print("Prediction for [2, 1, 0]:", prediction1)

print("Prediction for [2, 1, 1]:", prediction2)

**Output:**

Model Score: 1.0

Prediction for [2, 1, 0]: array([0] , dtype=int64 )

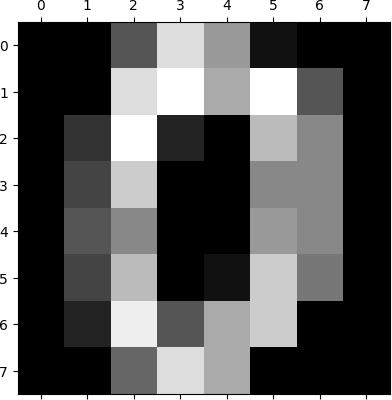
Prediction for [2, 1, 1]: array([1] , dtype=int64 )

**Result:**

Hence the decision tree classification algorithm was implemented successfully using the given dataset.

|  |  |
| --- | --- |
| Ex no: 8 | **Random Forest** |
| **Aim:**  To implement Random forest machine learning algorithm with appropriate dataset.  **Dataset Description:**  The dataset contains the handwritten digits which will be used for classification tasks in machine learning.  **Sample Dataset:**    **Code:**  import pandas as pd  from sklearn.datasets import load\_digits import matplotlib.pyplot as plt  import seaborn as sn  from sklearn.ensemble import RandomForestClassifier from sklearn.model\_selection import train\_test\_split from sklearn.metrics import confusion\_matrix  digits = load\_digits()  df = pd.DataFrame(digits.data) df['target'] = digits.target | |

X = df.drop('target', axis='columns') y = df.target



X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

model = RandomForestClassifier(n\_estimators=20) model.fit(X\_train, y\_train)

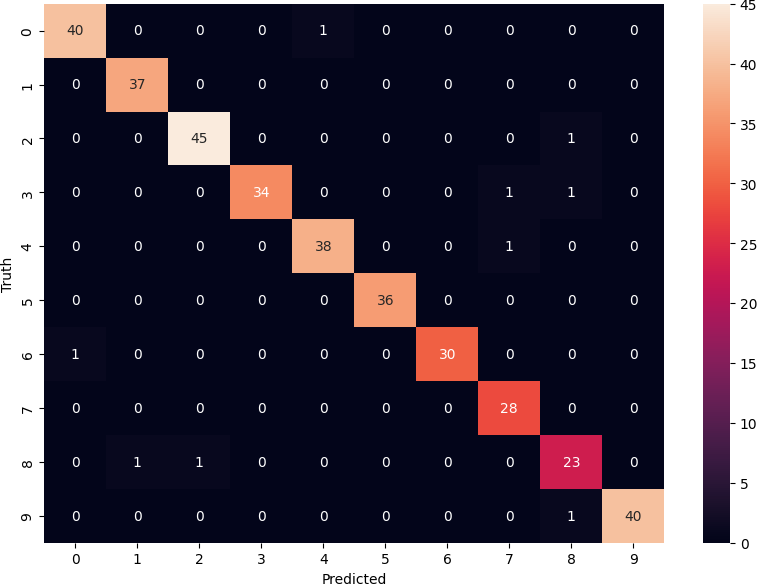
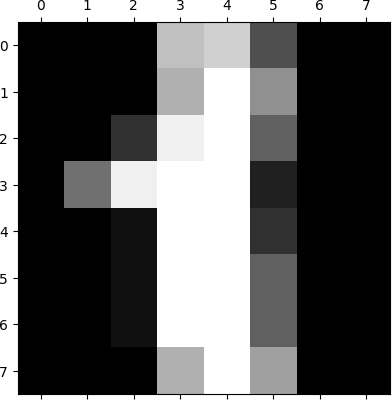
model.score(X\_test, y\_test)

y\_predicted = model.predict(X\_test)

cm = confusion\_matrix(y\_test, y\_predicted)

plt.figure(figsize=(10,7)) sn.heatmap(cm, annot=True) plt.xlabel('Predicted') plt.ylabel('Truth')

**Output:**

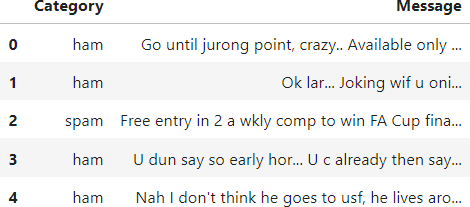


**Result:**

Hence the random forest machine learning algorithm was implemented successfully with the given dataset.

|  |  |
| --- | --- |
| Ex no: 9 | **Naive Bayes** |
| **Aim:**  To implement Naive Bayes machine learning algorithm using appropriate dataset.  **Dataset Description:**  The dataset contains the category in which the email should fall and the type of messages the email contains.  **Sample Dataset:**  Spam.csv  **Category Message**  **ham** Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...  **ham** Ok lar... Joking wif u oni...  **spam** Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's  **ham** U dun say so early hor... U c already then say...  **ham** Nah I don't think he goes to usf, he lives around here though  **Code:**  import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.feature\_extraction.text import CountVectorizer from sklearn.naive\_bayes import MultinomialNB  from sklearn.pipeline import Pipeline  # Read the CSV file  df = pd.read\_csv("spam.csv")  # Convert 'Category' column to binary labels  df['spam'] = df['Category'].apply(lambda x: 1 if x == 'spam' else 0)  # Splitting data into training and testing sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.Message, df.spam)  # Creating a pipeline clf = Pipeline([  ('vectorizer', CountVectorizer()), # Convert text to vectors ('nb', MultinomialNB()) # Multinomial Naive Bayes classifier  ])  # Training the model | |

clf.fit(X\_train, y\_train)



# Model evaluation

accuracy = clf.score(X\_test, y\_test) print("Model accuracy:", accuracy)

# Example emails emails = [

'Hey mohan, can we get together to watch footbal game tomorrow?',

'Upto 20% discount on parking, exclusive offer just for you. Dont miss this reward!'

]

# Predictions

predictions = clf.predict(emails) print("Predictions:", predictions)

**Output:**

Model accuracy: 0.9849246231155779

Predictions: [0 1]

**Result:**

Hence the Naive Bayes algorithm was implemented using the given dataset successfully.

|  |  |
| --- | --- |
| Ex no: 10 | **Implementing Neural Networks through Python** |
| **Aim:**  To implement a simple Artificial Neural Network through python.  **Code:**  from joblib.numpy\_pickle\_utils import xrange from numpy import \*  class NeuralNet(object): def init (self):  # Generate random numbers random.seed(1)  # Assign random weights to a 3 x 1 matrix, self.synaptic\_weights = 2 \* random.random((3, 1)) - 1  # The Sigmoid function def sigmoid(self, x):  return 1 / (1 + exp(-x))  # The derivative of the Sigmoid function.  # This is the gradient of the Sigmoid curve. def sigmoid\_derivative(self, x):  return x \* (1 - x)  # Train the neural network and adjust the weights each time. def train(self, inputs, outputs, training\_iterations):  for iteration in xrange(training\_iterations):  # Pass the training set through the network. output = self.learn(inputs)  # Calculate the error error = outputs - output  # Adjust the weights by a factor  factor = dot(inputs.T, error \* self. sigmoid\_derivative(output)) self.synaptic\_weights += factor  # The neural network thinks. def learn(self, inputs):  return self. sigmoid(dot(inputs, self.synaptic\_weights))  if name == " main ": # Initialize  neural\_network = NeuralNet() | |

# The training set.

inputs = array([[0, 1, 1], [1, 0, 0], [1, 0, 1]])

outputs = array([[1, 0, 1]]).T

# Train the neural network neural\_network.train(inputs, outputs, 10000)

# Test the neural network with a test example. print(“Prediction: ”+neural\_network.learn(array([1, 0, 1])))

**Output:**

Prediction: [0.9897704]

**Result:**

Hence the Neural Network was successfully implemented in Python.