RATHINAM TECHNICAL CAMPUS

RATHINAM TECHZONE

POLLACHI MAIN ROAD, EACHANARI, COIMBATORE-641021.

MASTER OF COMPUTER APPLICATIONS

RECORD NOTE BOOK

23MC301 MACHINE LEARNING LABORATORY

NAME :

REGISTER NUMBER :

YEAR/SEMESTER :

ACADEMIC YEAR :

RATHINAM TECHNICAL CAMPUS

RATHINAM TECHZONE

POLLACHI MAIN ROAD, EACHANARI, COIMBATORE-641021.

BONAFIDE CERTIFICATE

NAME :

ACADEMIC YEAR :

YEAR/SEMESTER :

BRANCH :

UNIVERSITY REGISTER NUMBER: ……………………………….

Certified that this is the bonafide record of work done by the above student in the

____________________________________Laboratory during the year 2024-2025.

Head of the Department **Staff-in-Charge**

Submitted for the Practical Examination held on _________________________

Internal Examiner **External Examiner**

S.No	Date	Experiment Name	Marks	Staff Sign

EX NO: 01 DATE:	**Explore the significant steps involved in data preprocessing in Machine Learning.**

Data preprocessing is a crucial step in the machine learning pipeline. It involves transforming raw data into a clean and usable format, ensuring that it is suitable for building machine learning models. Below is a Python program that illustrates the significant steps involved in data preprocessing using a sample dataset.

We'll use the popular scikit-learn library along with pandas and numpy for this demonstration.

- Import necessary libraries.
- Load the dataset.
- Handle missing values.
- Encode categorical data.
- Feature scaling.
- Split the dataset into training and testing sets.

Program:

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Sample dataset
data = {
```

```
    'age': [25, 30, np.nan, 35, 40, 22],
    'income': [50000, 70000, 60000, np.nan, 80000, 90000],
    'gender': ['Male', 'Female', 'Female', 'Male', 'Male', 'Female'],
    'purchased': ['No', 'Yes', 'No', 'Yes', 'Yes', 'No']
}

df = pd.DataFrame(data)

# Step 1: Handling missing data
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
df[['age', 'income']] = imputer.fit_transform(df[['age', 'income']])

# Step 2: Encoding categorical data
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [2])], remainder='passthrough')
df_encoded = pd.DataFrame(ct.fit_transform(df))

# Step 3: Feature scaling
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df_encoded.iloc[:, :-1]))

# Step 4: Splitting the dataset into training and testing sets
X = df_scaled
y = df_encoded.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
# Display the processed data
print("Original DataFrame:")
print(df)

print("\nDataFrame after handling missing data:")
print(df_encoded)

print("\nDataFrame after encoding categorical data and scaling numerical data:")
print(df_scaled)

print("\nTraining and Testing sets:")
print("X_train:\n", X_train)
print("\nX_test:\n", X_test)
print("\ny_train:\n", y_train)
print("\ny_test:\n", y_test)
```

OUTPUT

IDLE Shell 3.12.3

File Edit Shell Debug Options Window Help

```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=============== RESTART: C:\Users\WELCOME\Documents\ML lab\ex1.py ==============
Original DataFrame:
    age   income  gender purchased
0  25.0  50000.0    Male        No
1  30.0  70000.0  Female       Yes
2  30.4  60000.0  Female        No
3  35.0  70000.0    Male       Yes
4  40.0  80000.0    Male       Yes
5  22.0  90000.0  Female        No

DataFrame after handling missing data:
     0    1     2        3    4
0  0.0  1.0  25.0  50000.0   No
1  1.0  0.0  30.0  70000.0  Yes
2  1.0  0.0  30.4  60000.0   No
3  0.0  1.0  35.0  70000.0  Yes
4  0.0  1.0  40.0  80000.0  Yes
5  1.0  0.0  22.0  90000.0   No

DataFrame after encoding categorical data and scaling numerical data:
     0    1             2         3
0 -1.0  1.0 -9.058907e-01 -1.549193
1  1.0 -1.0 -6.710301e-02  0.000000
2  1.0 -1.0 -5.959945e-16 -0.774597
3 -1.0  1.0  7.716846e-01  0.000000
4 -1.0  1.0  1.610472e+00  0.774597
5  1.0 -1.0 -1.409163e+00  1.549193

Training and Testing sets:
X_train:
      0    1         2         3
1  1.0 -1.0 -0.067103  0.000000
3 -1.0  1.0  0.771685  0.000000
0 -1.0  1.0 -0.905891 -1.549193
4 -1.0  1.0  1.610472  0.774597

X_test:
      0    1             2         3
5  1.0 -1.0 -1.409163e+00  1.549193
2  1.0 -1.0 -5.959945e-16 -0.774597

y_train:
 1    Yes
3    Yes
0     No
4    Yes
Name: 4, dtype: object

y_test:
 5    No
2    No
Name: 4, dtype: object
>>>
```

RESULT:

EX NO: 02 DATE:	**Choose a model and train a model in machine learning.**

Step-by-Step Python Program for Training a Machine Learning Model.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Sample dataset (using a toy dataset for demonstration)
data = {
   'age': [25, 30, 35, 20, 40, 45, 22, 38, 50, 60],
   'income': [50000, 70000, 60000, 30000, 80000, 90000, 40000, 120000, 100000, 150000],
   'purchased': [0, 1, 1, 0, 1, 1, 0, 1, 1, 1]
}

df = pd.DataFrame(data)

# Separate features and target variable
X = df[['age', 'income']]
y = df['purchased']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Initialize Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)

# Train the model
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred, zero_division=1))
```

OUTPUT

```
= RESTART: C:\Users\WELCOME\Documents\ML lab\ex2.py
Accuracy: 0.50

Classification Report:
              precision    recall  f1-score   support

           0       0.00      1.00      0.00         0
           1       1.00      0.50      0.67         2

    accuracy                           0.50         2
   macro avg       0.50      0.75      0.33         2
weighted avg       1.00      0.50      0.67         2
```

RESULT

EX NO: 03 DATE:	**Explain the application of Bayes Theorem and how it's useful to predict the future**

```
import numpy as np
from collections import defaultdict

# Sample dataset of emails and their labels (1 for not spam, 0 for spam)
emails = [
    ('Buy now, limited offer!', 0),
    ('Hello, how are you?', 1),
    ('Exclusive deal just for you', 0),
    ('Meeting agenda for next week', 1),
    ('Claim your prize now', 0)
]

# Function to tokenize words from text
def tokenize(text):
    return text.lower().split()

# Initialize dictionaries to store word counts in spam and non-spam emails
word_count_spam = defaultdict(int)
word_count_ham = defaultdict(int)

# Process each email to populate word counts
total_spam = 0
total_ham = 0
```

```
for email, label in emails:
    words = tokenize(email)
    if label == 1:
        for word in words:
            word_count_spam[word] += 1
            total_spam += 1
    else:
        for word in words:
            word_count_ham[word] += 1
            total_ham += 1

# Function to calculate probability of each word being spam
def calculate_word_probabilities(word_counts, total_count):
    probabilities = {}
    for word, count in word_counts.items():
        probabilities[word] = count / total_count
    return probabilities

# Calculate probabilities of each word being spam or ham
prob_word_spam = calculate_word_probabilities(word_count_spam, total_spam)
prob_word_ham = calculate_word_probabilities(word_count_ham, total_ham)

# Prior probabilities (probability of an email being spam or ham)
p_spam = sum(1 for _, label in emails if label == 1) / len(emails)
p_ham = 1 - p_spam

# Function to predict if an email is spam using Bayes' Theorem
```

```
def predict_spam(email):
    words = tokenize(email)
    log_p_spam_given_email = np.log(p_spam)
    log_p_ham_given_email = np.log(p_ham)

    for word in words:
        if word in prob_word_spam:
            log_p_spam_given_email += np.log(prob_word_spam[word])
        if word in prob_word_ham:
            log_p_ham_given_email += np.log(prob_word_ham[word])

    # Normalize probabilities using log-sum-exp trick
    max_log_prob = max(log_p_spam_given_email, log_p_ham_given_email)
    p_spam_given_email = np.exp(log_p_spam_given_email - max_log_prob)
    p_ham_given_email = np.exp(log_p_ham_given_email - max_log_prob)

    return p_spam_given_email / (p_spam_given_email + p_ham_given_email)

# Test the model with new emails
test_emails = [
    'Limited offer, claim your prize now!',
    'Hello, how about meeting for lunch tomorrow?',
    'Exclusive deal just for you'
]

for email in test_emails:
    spam_probability = predict_spam(email)
```

```
print(f"Email: '{email}'")
print(f"Spam Probability: {spam_probability:.4f}")
if spam_probability > 0.5:
    print("Prediction: Spam\n")
else:
    print("Prediction: Not Spam\n")
```

OUTPUT

```
=========== RESTART: C:\Users\WELCOME\Documents\ML lab\ex3.py ===========
Email: 'Limited offer, claim your prize now!'
Spam Probability: 0.9999
Prediction: Spam

Email: 'Hello, how about meeting for lunch tomorrow?'
Spam Probability: 0.0013
Prediction: Not Spam

Email: 'Exclusive deal just for you'
Spam Probability: 1.0000
Prediction: Spam
```

RESULT:

EX NO: 04 **DATE:**	**Make the difference between supervised Learning and unsupervised Learning Techniques**

Supervised Learning vs. Unsupervised Learning

Supervised Learning:

Definition: Supervised learning is a type of machine learning where the model is trained on labeled data, meaning the input data has corresponding output labels.

Objective: The goal is to learn a mapping function from input variables (features) to output variables (labels) based on labeled training data.

Examples: Classification (predicting discrete labels) and regression (predicting continuous values) are common tasks in supervised learning.

Unsupervised Learning:

Definition: Unsupervised learning is a type of machine learning where the model is trained on unlabeled data, meaning the input data does not have corresponding output labels.

Objective: The goal is to discover patterns or hidden structures in the input data without explicit feedback.

Examples: Clustering (grouping similar data points) and dimensionality reduction (reducing the number of input variables) are common tasks in unsupervised learning.

Supervised Learning (Classification)

Let's use a simple example of supervised learning with the Iris dataset, where we predict the species of iris flowers based on their features.

```
# Importing necessary libraries

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report
```

```
# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features
y = iris.target  # Labels

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the model (Logistic Regression for classification)
model = LogisticRegression(max_iter=200)

# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

OUTPUT

```
======= RESTART: C:/Users/WELCOME/Documents/ML lab/ex4 1.py =======
Accuracy: 1.00

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

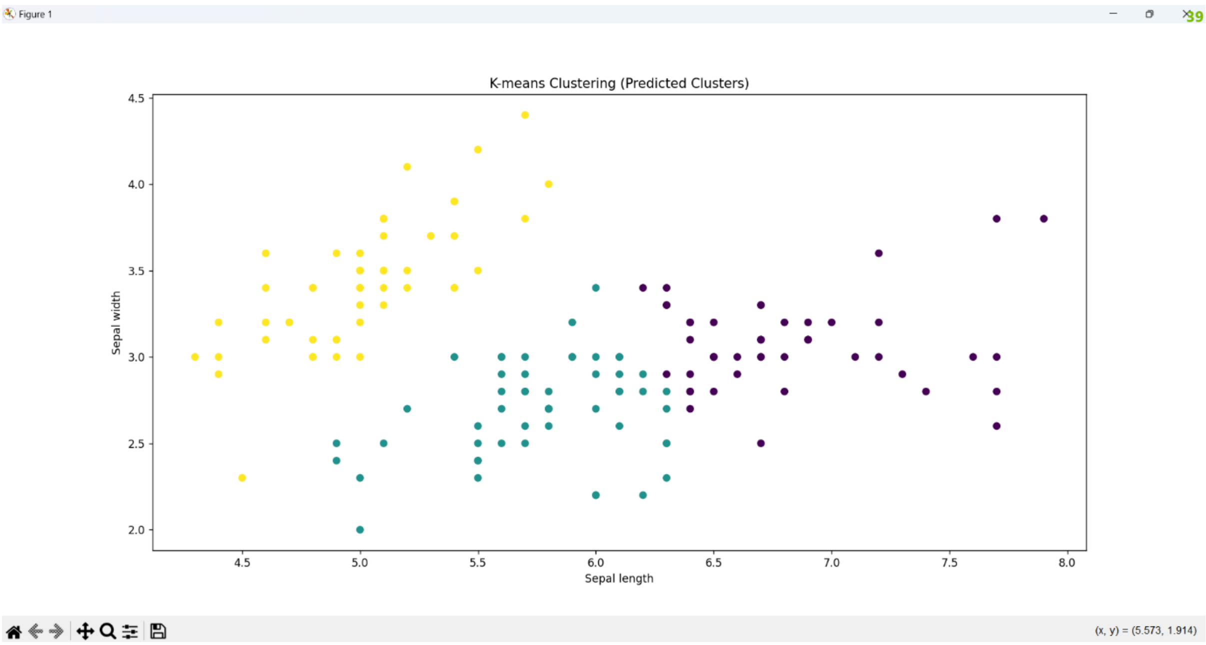
Unsupervised Learning (Clustering)

Now, let's demonstrate unsupervised learning with a simple example using the Iris dataset again, but this time we'll perform K-means clustering to group similar data points.

```
# Importing necessary libraries

from sklearn.cluster import KMeans

from sklearn.datasets import load_iris

import matplotlib.pyplot as plt


iris = load_iris()

# Using only two features (sepal length and sepal width) for simplicity

X = iris.data[:, :2]  # Selecting first two features (sepal length and sepal width)


# Initialize the K-means model

kmeans = KMeans(n_clusters=3, random_state=42)


# Fit the model to the data
```

```
kmeans.fit(X)

# Predict the cluster labels
y_pred = kmeans.labels_

# Visualize the clusters
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('K-means Clustering (Predicted Clusters)')
plt.show()
```

OUTPUT



RESULT

EX NO: 05 **DATE:**	**Differentiate Perceptron, Neural Network, Convolutional Neural Network and Deep Learning**

To differentiate between Perceptron, Neural Network (NN), Convolutional Neural Network (CNN), and Deep Learning, let's discuss each concept briefly and then provide a simple Python example for Perceptron to illustrate its basic functionality.

1. Perceptron:

Definition: The Perceptron is a basic unit of a neural network, simulating a single neuron. It takes several binary inputs, calculates a weighted sum, and applies a step function to produce a binary output.

Use: It's typically used for binary classification problems and forms the basis of more complex neural networks.

2. Neural Network (NN):

Definition: A Neural Network consists of multiple layers of neurons (Perceptrons), organized in an interconnected manner. Each neuron receives input, applies weights, and passes the output through an activation function to the next layer.

Use: Neural Networks are versatile and can handle various tasks like classification, regression, and pattern recognition.

3. Convolutional Neural Network (CNN):

Definition: A CNN is a specialized type of neural network designed for processing structured grids of data such as images. It uses convolutional layers to automatically learn hierarchical patterns.

Use: CNNs excel in tasks like image and video recognition, where spatial relationships between data points (pixels) are important.

4. Deep Learning:

Definition: Deep Learning refers to neural networks with multiple hidden layers, allowing them to learn complex representations of data. It encompasses various architectures like CNNs, Recurrent Neural Networks (RNNs), and more.

Use: Deep Learning has revolutionized fields like computer vision, natural language processing, and speech recognition due to its ability to automatically learn features from data.

Perceptron in Python

Let's implement a Perceptron to classify a simple dataset.

```
import numpy as np

class Perceptron:
    def __init__(self, input_size, learning_rate=0.01, epochs=100):
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.weights = np.zeros(input_size + 1)  # +1 for bias

    def activation_function(self, x):
        # Step function
        return 1 if x >= 0 else 0

    def predict(self, inputs):
        summation = np.dot(inputs, self.weights[1:]) + self.weights[0]  # w0 is bias
        return self.activation_function(summation)

    def train(self, training_inputs, labels):
        for _ in range(self.epochs):
            for inputs, label in zip(training_inputs, labels):
                prediction = self.predict(inputs)
                self.weights[1:] += self.learning_rate * (label - prediction) * inputs
                self.weights[0] += self.learning_rate * (label - prediction)

# Example usage
if __name__ == "__main__":
    # Sample dataset (OR gate)
```

```
training_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
labels = np.array([0, 1, 1, 1])  # OR gate output

# Initialize and train the Perceptron
perceptron = Perceptron(input_size=2)
perceptron.train(training_inputs, labels)

# Test the trained Perceptron
test_inputs = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
for inputs in test_inputs:
    print(f"Input: {inputs}, Predicted Output: {perceptron.predict(inputs)}")
```

OUTPUT

```
=========== RESTART: C:/Users/WELCOME/Documents/ML lab/ex5.py ==========
Input: [0 0], Predicted Output: 0
Input: [0 1], Predicted Output: 1
Input: [1 0], Predicted Output: 1
Input: [1 1], Predicted Output: 1
```

RESULT