



INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

DATA MINING AND KNOLEDGE DISCOVERY LABORATORY								
VI Semester: CSE (CS)								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
ACIC08	Core	L	T	P	C	CIA	SEE	Total
		1	0	2	2	30	70	100
Contact Classes: 12	Tutorial Classes: NIL	Practical Classes: 33			Total Classes: 45			
Prerequisite: There are no prerequisites to take this course.								

I. COURSE OVERVIEW:

This course helps the students to practically understand a data warehouse, techniques and methods for data gathering and data pre-processing using different tools. The different data mining models and techniques will be discussed in this course. The main objective of this lab is to impart the knowledge on how to implement classical models and algorithms in data warehousing and data mining and to characterize the kinds of patterns that can be discovered by association rule mining, classification and clustering.

II. COURSE OBJECTIVES:

The students will try to learn:

- I. The Data Object Exploration and visualization
- II. The pre-processing on new and existing datasets.
- III. Frequent item set generation and association rules on transactional data.
- IV. The data model creation by using various classification and clustering algorithms.
- V. The data models accuracy analysis by varying the sample size.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

- CO 1 Analyze the knowledge generated from data objects, matrix operations using Numpy.
- CO 2 Demonstrate Numpy module methods to categorize and correlate the raw data.
- CO 3 Select appropriate pre-processing techniques to manage the missing values of data.
- CO 4 Apply Apriori Algorithm and logistic regression for classification of data mining.
- CO 5 Identify Classification technique from Decision Tree, Bayesian Network and Support Vector Machines to mine knowledge from pre-processed data.
- CO 6 Examine Clustering algorithms to build predication model for solving real world problem.

IV.COURSE CONTENT:

EXERCISES FOR DATA MINING AND KNOWLEDGE DISCOVERY LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

Introduction:

The Data Mining Lab with Python is designed to introduce students and professionals to the practical aspects of data mining, leveraging Python's rich ecosystem of data analysis and machine learning libraries. Python, being one of the most popular programming languages in the data science community, provides a comprehensive environment for data manipulation, visualization, and analysis.

Software:

Anaconda Distribution, combined with tools like Jupyter Notebook or IDEs like PyCharm or VS Code, provides a robust environment for tackling data mining tasks with Python and its libraries.

REFERENCE BOOKS:

1. Robert Layton, "Learning Data Mining with Python", Packt Publishing, 2015.

Web References:

- I. <https://www.dataquest.io/blog/sci-kit-learn-tutorial/>
- II. <https://archive.ics.uci.edu/ml/datasets.php/>
- III. <https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python>

NUMPY:

NumPy stands for Numerical Python which is a Python library used for working with arrays. It provides an efficient interface to store and operate on dense data buffers. NumPy arrays provide much more efficient storage and data operations as the arrays grow larger in size. NumPy arrays form the core of nearly the entire ecosystem of data science tools in Python. It also has functions for working in domain of linear algebra, fourier transform and matrices. The array object in NumPy is called ndarray

1.1 Implement Multidimensional 2-D and 3-D arrays using Numpy

Multidimensional arrays are arrays that have more than one dimension. They can be thought of as arrays of arrays. Commonly used multidimensional arrays include 2D arrays (matrices) and 3D arrays. Create multidimensional arrays and find its shape and dimension.

Input: `array_2d = np.array ([[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])`

Output: 2D Array :

Shape: (3, 3)

Dimension: 2

Explanation: The "shape" attribute returns a tuple representing the array's "dimensions", and ndim returns the number of dimensions.

Input: `array_3d = np.array ([[[1, 2], [3, 4]],
[[5, 6], [7, 8]],
[[9, 10], [11, 12]]])`

Output: 3D Array:

Shape: (3, 2, 2)

Dimension: 3

```
Import numpy as np

a=np.array([[1,2,3],[2,3,4],[3,4,5]])

b=a.shape
print("shape:",a.shape)

c=a.ndim
print("dimensions:",a.ndim)
# Write code here
...
```

1.2 Implement a matrix full of zeros and ones using Numpy

NumPy can create matrices filled with zeros or ones using the `np.zeros` and `np.ones` functions, respectively.

Input: `matrix_zeros = np.zeros((3, 4))`

Output: Matrix full of zeros:

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
```

Input: `matrix_ones = np.ones((2, 3))`

Output: Matrix full of ones:

```
[[1. 1. 1.]
```

```
[1. 1. 1.]]
```

Input: `z=np.zeros((2,2))`

Output: `[[0. 0.]`

`[0. 0.]]`

```
import numpy as np
matrix_zeros = np.zeros((3, 4))

print("Matrix full of zeros:")
print(matrix_zeros)

matrix_ones = np.ones((2, 3))

print("Matrix full of ones:")
print(matrix_ones)
```

TRY:

Exercises:

1.Create an array of evenly spaced values (step value)

Hint: Use `arrange()` method

2.Create an array of evenly spaced values (number of samples)

Hint: Use `linespace()` method

3.Create a constant array

Hint: Use `full()` method

4.Create a 2X2 identity matrix

Hint: Use `eye()` method

5.Create an array with random values

Hint: Use `random()` method

6.Create an empty array

Hint: Use `empty()` method

1.3 Implement functions Reshape and flatten data in the array

NumPy is used to `reshape` and `flatten` functions for modifying the shape of an array.

Input: `original_array = np.array([[1, 2, 3],`

`[4, 5, 6]])`

Output: Reshaped Array:

```
[[1 2]
 [3 4]
 [5 6]]
```

```
import numpy as np

a=np.array([[1,2,3,4],[2,3,4,5],[3,4,5,6],[4,5,6,7]])
b=a.reshape(4,2,2)

    # Write code here
    reshaped_array = original_array.reshape((3, 2))

print("Original Array:")
print(original_array)
print("\nReshaped Array:")
print(reshaped_array)
```

TRY:

Exercises:

- 1.Find length of an array
- 2.Find size of an array
- 3.Find Data type of array elements
Hint: Use dtype
- 4.Find the name of the data type
Hint: Use dtype.name
- 5.Convert an array to a different type
Hint: Use astype(int) method
6. Reshape, but don't change data

1.4 Implement functions Append data vertically and horizontally

numpy.vstack is used to vertically stack arrays (i.e., stack arrays one on top of the other).

When two or more arrays with the same number of columns, vstack concatenates them along the vertical axis (axis 0), forming a new array with the same number of columns but a greater number of rows.

The function takes a tuple of arrays as input and returns a single array as output.

numpy.hstack is used to horizontally stack arrays (i.e., stack arrays side by side).

When you have two or more arrays with the same number of rows, hstack concatenates them along the horizontal axis (axis 1), forming a new array with the same number of rows but a greater number of columns.

The function takes a tuple of arrays as input and returns a single array as output.

If the arrays being stacked do not have the same number of rows, hstack will raise a ValueError.

Example usage: np.hstack((array1, array2))

```
Input: array1 = np.array([[1, 2, 3],
                          [4, 5, 6]])

        array2 = np.array([[7, 8, 9],
                          [10, 11, 12]])
```

Output: Vertically stacked array:

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Horizontally stacked array:

```
[[ 1  2  3  7  8  9]
 [ 4  5  6 10 11 12]]
```

Explanation: stacks the arrays vertically using NumPy's vstack and hstack functions

```
import numpy as np

# Example data arrays
array1 = np.array([[1, 2, 3],
                  [4, 5, 6]])

array2 = np.array([[7, 8, 9],
                  [10, 11, 12]])

# Vertically stack arrays
stacked_vertically = np.vstack((array1, array2))

print("Vertically stacked array:")
print(stacked_vertically)
print()

# Horizontally stack arrays
stacked_horizontally = np.hstack((array1, array2))

print("Horizontally stacked array:")
print(stacked_horizontally)
```

Try:

Exercises:

- 1.Implement Stack arrays vertically (row-wise)
 - 2.Create stacked column-wise arrays
 - 3.Implement Arithmetic Operations: Addition, Subtraction, Multiplication, Division, Exponentiation, Square root, Print sines of an array, Element-wise cosine, Element-wise natural logarithm, Dot product
-

1.5 Apply indexing and slicing on array using Numpy

Indexing in NumPy allows accessing individual elements of an array using integer indices, denoted as `array[i, j]`, where `i` represents the row index and `j` represents the column index.

Slicing in NumPy enables extracting subarrays or specific portions of an array using slice notation, denoted as `array[start:stop:step]`. It's applicable along one or more dimensions, allowing for versatile extraction of data from arrays.

Input: `arr = np.array([[1, 2, 3],
 [4, 5, 6],
 [7, 8, 9]])`

Output: Accessing single elements:

Element at (0, 0): 1

Element at (1, 2): 6

Accessing subarrays:

First row: `[1 2 3]`

Second column: `[2 5 8]`

Subarray (2x2) from top-left corner:

`[[1 2]`

`[4 5]]`

Modifying elements using indexing:

Array after modifying element at (1, 1):

`[[1 2 3]`

`[4 10 6]`

`[7 8 9]]`

Modifying subarrays using slicing:

Array after modifying third column:

`[[1 2 100]`

`[4 10 200]`

`[7 8 300]]`

```
import numpy as np
```

```

# Create a NumPy array
arr = np.array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

# Print original array
print("Original array:")
print(arr)
print()

# Indexing: Accessing single elements
print("Accessing single elements:")
print("Element at (0, 0):", arr[0, 0])
print("Element at (1, 2):", arr[1, 2])
print()

# Slicing: Accessing subarrays
print("Accessing subarrays:")
print("First row:", arr[0])
print("Second column:", arr[:, 1])
print("Subarray (2x2) from top-left corner:")
print(arr[:2, :2])
print()

# Modifying elements using indexing
print("Modifying elements using indexing:")
arr[1, 1] = 10
print("Array after modifying element at (1, 1):")
print(arr)
print()

# Modifying subarrays using slicing
print("Modifying subarrays using slicing:")
arr[:, 2] = [100, 200, 300]
print("Array after modifying third column:")
print(arr)

```

1.6 Implement statistical functions on array, Min, Max, Mean, Median and Standard Deviation

You are given an $m \times n$ integer grid `accounts` where `accounts[i][j]` is the amount of money the i^{th} customer has in the j^{th} bank. Return the wealth that the richest customer has. A customer's wealth is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum wealth.

Input: `arr = np.array([[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])`

Output:

Minimum value: 1

Maximum value: 9

Mean: 5.0

Median: 5.0

Standard Deviation: 2.581988897471611

Explanation:

The original array is printed.

The minimum value in the array is 1.

The maximum value in the array is 9.

The mean (average) value of the array is calculated to be 5.0.

The median value of the array is also calculated to be 5.0.

The standard deviation of the array is calculated to be approximately 2.582.

```
import numpy as np

# Create a NumPy array
arr = np.array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

# Minimum value
min_val = np.min(arr)
# Maximum value
max_val = np.max(arr)

# Mean
mean_val = np.mean(arr)

# Median
median_val = np.median(arr)

# Standard deviation
```

```
std_dev = np.std(arr)

# Print results
print("Original Array:")
print(arr)
print()
print("Minimum value:", min_val)
print("Maximum value:", max_val)
print("Mean:", mean_val)
print("Median:", median_val)
print("Standard Deviation:", std_dev)
```

Try:

Exercises:

1.Create a view of the array with the same data

Hint: Use View()

2.Create a copy of the array

Hint: Use copy()

3.Create a deep copy of the array

4.Sort the elements of an array's axis

5.Implement Concatenation of two arrays

Hint: Use concatenate() method

6.Split the array vertically at the 2nd index

Hint: np.vsplit(c,2)

2. Matrix Operations USING NUMPY

2.1 Dot and matrix product of two arrays

Given two matrices X and Y, the task is to compute the sum of two matrices and then print it in Python.

Input:

Array 1: $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$
Array 2: $\begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$

Output:

Dot Product: 70
Matrix Product: $\begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$

Explanation:

The program prints the original arrays array1 and array2.

It calculates the dot product of the arrays, which is $15 + 26 + 37 + 48 = 70$.

It also calculates the matrix product of the arrays using nested loops for matrix multiplication.

The resulting matrix product is printed.

```
import numpy as np

# Define two arrays
array1 = np.array([[1, 2],
                   [3, 4]])

array2 = np.array([[5, 6],
                   [7, 8]])

# Dot product of arrays
dot_product = 0
for i in range(array1.shape[0]):
    for j in range(array1.shape[1]):
        dot_product += array1[i][j] * array2[i][j]

# Matrix product of arrays
matrix_product = np.zeros((array1.shape[0], array2.shape[1]))
for i in range(array1.shape[0]):
    for j in range(array2.shape[1]):
        for k in range(array1.shape[1]):
            matrix_product[i][j] += array1[i][k] * array2[k][j]

# Print results
print("Array 1:")
print(array1)
print("Array 2:")
print(array2)
print("Dot Product:", dot_product)
print("Matrix Product:")
print(matrix_product)
for r in result:
    print(r)
```

2.2 Compute the Eigen values of a matrix

Eigenvalues are a set of scalar values associated with a square matrix that describe how the matrix transforms vectors. They represent the scaling factors by which these vectors are stretched or compressed when transformed by the matrix. Eigenvalues play a crucial role in various mathematical and engineering applications, such as solving systems of differential equations, analyzing stability in dynamical systems, and dimensionality reduction techniques like Principal Component Analysis (PCA).

Given two matrices X and Y, the task is to compute the multiplication of two matrices and then print it.

Input:

```
matrix = np.array([[1, 2],
                   [3, 4]])
```

Output:

Eigenvalues of the matrix:
[-0.37228132 5.37228132]

```
import numpy as np

# Define a matrix
matrix = np.array([[1, 2],
                   [3, 4]])

# Compute eigenvalues
eigenvalues = np.linalg.eigvals(matrix)

# Print eigenvalues
print("Eigenvalues of the matrix:")
print(eigenvalues)
```

The `np.linalg.eigvals()` function is used to compute the eigenvalues of the matrix. The eigenvalues are stored in the variable `eigenvalues`, and we print them out.

2.3 Solve a linear matrix equation such as $3 * x_0 + x_1 = 9$, $x_0 + 2 * x_1 = 8$

Numpy provides efficient tools for solving linear matrix equations through functions like `numpy.linalg.solve()`. These equations involve expressing a system of linear equations in matrix form ($Ax = b$), where A is the coefficient matrix, x is the vector of variables to be solved for, and b is the vector of constants. Numpy's linear algebra module allows for quick and accurate computation of solutions, facilitating tasks in engineering, physics, and data analysis.

Input: `A = np.array([[3, 1], [1, 2]])`
`B = np.array([9, 8])`

Output: `x0 = 2.0`
`x1 = 3.0`

```
import numpy as np

# Coefficients of the linear equations
A = np.array([[3, 1], [1, 2]])
b = np.array([9, 8])

# Solve the linear equations
solution = np.linalg.solve(A, b)

print("Solution:")
print("x0 =", solution[0])
print("x1 =", solution[1])
```

Explanation: This program defines the coefficients of the linear equations in matrix form (A) and the constants on the right-hand side of the equations (b). Then, it uses `numpy.linalg.solve()` to find the solution vector x , which contains the values of x_0 and x_1 . Finally, it prints out the solution.

2.4 Compute the multiplicative inverse of a matrix

Numpy's `numpy.linalg.inv()` function efficiently computes the multiplicative inverse of a square matrix. This inverse matrix, when multiplied with the original matrix, yields the identity matrix. This operation is fundamental in solving systems of linear equations, transforming vectors, and various numerical computations in fields like engineering, physics, and machine learning.

Input: Original matrix:

```
A = np.array([[4, 7], [2, 6]])
```

Output: Inverse of the matrix: $\begin{bmatrix} 0.6 & -0.7 \\ -0.2 & 0.4 \end{bmatrix}$

```
import numpy as np

# Define the matrix
A = np.array([[4, 7], [2, 6]])

# Compute the inverse of the matrix
A_inv = np.linalg.inv(A)

print("Original matrix:")
print(A)

print("\nInverse of the matrix:")
print(A_inv)
```

Explanation: This program first defines a matrix A, then it calculates its inverse using `np.linalg.inv()`, and finally prints both the original matrix and its inverse. Make sure the matrix is square and non-singular for its inverse to exist. If the matrix is singular or non-square, `numpy.linalg.inv()` will raise a 'LinAlgError'.

Try:

Exercises: Transpose of a Matrix

2.5 Compute the rank of a matrix

Input: Original matrix:

```
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])
```

Output: Rank of the matrix: 2

```
import numpy as np

# Define the matrix
A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

# Compute the rank of the matrix
rank = np.linalg.matrix_rank(A)

print("Rank of the matrix:", rank)
```

Explanation: This program defines a matrix A, then it calculates its rank using `numpy.linalg.matrix_rank()`, and finally prints out the rank of the matrix.

2.6 Compute the determinant of an array

Input: Original matrix:

```
A = np.array([[1, 2],  
              [3, 4]])
```

Output: Determinant of the array: -2.0000000000000004

```
import numpy as np  
  
# Define the array (matrix)  
A = np.array([[1, 2],  
              [3, 4]])  
  
# Compute the determinant of the array  
determinant = np.linalg.det(A)  
  
print("Determinant of the array:", determinant)
```

Explanation: This program defines a 2x2 array A, then it calculates its determinant using `numpy.linalg.det()`, and finally prints out the determinant of the array.

Asking For Help

`np.info(np.ndarray.dtype)`

Try:

1. Saving & Loading On Disk

`Save()`, `Savez()`, `Load()` methods

2. Saving & Loading Text Files

`Loadtxt("myfile.txt")`,

`genfromtxt("my_file.csv", delimiter=',')`

`savetxt("myarray.txt", a, delimiter=" ")`

3. EXPLORATION AND VISULIZATION OF DATA

Exploration of Data:

Exploration of data is an iterative process that involves iteratively exploring, visualizing, and analyzing the dataset to gain insights, generate hypotheses, and inform subsequent data-driven decisions or modeling approaches.

3.1 Loading data from CSV file

To load data from a CSV file in Python, particularly using the `brain_size.csv` dataset, you can use the pandas library. Below is a simple Python program demonstrating how to load the data from the CSV file:

Explanation: This program reads the CSV file using `pd.read_csv()` function from pandas, then displays the first few rows of the DataFrame using `df.head()`.

Source of Dataset: <https://www.kaggle.com/code/rashmiek99/head-size-vs-brain-weight>

```
import pandas as pd

# Load the CSV file into a pandas DataFrame
df = pd.read_csv("brain_size.csv")

# Display the first few rows of the DataFrame
print("First few rows of the dataset:")
print(df.head())
```

Output:

First few rows of the dataset:

	Gender	FSIQ	VIQ	PIQ	Weight	Height	MRI_Count
0	Female	133	132	124	118.0	64.5	816932
1	Male	140	150	124	NaN	72.5	1001121
2	Male	139	123	150	143.0	73.3	1038437
3	Male	133	129	128	172.0	68.8	965353
4	Female	137	132	134	147.0	65.0	951545

3.2 Compute the basic statistics of given data, shape, no. of columns, mean

This program reads the data from the `brain_size.csv` file using `pd.read_csv()` function from pandas. Then, it prints the shape of the DataFrame using `df.shape`, the number of columns using `len(df.columns)`, and computes the mean of each numerical column using `df.mean()`. Finally, it prints out the means. Make sure to replace `"brain_size.csv"` with the appropriate path if the file is located elsewhere.

```
import pandas as pd

# Load the data from the CSV file
df = pd.read_csv("brain_size.csv")

# Display the shape of the DataFrame
print("Shape of the DataFrame (rows, columns):", df.shape)

# Display the number of columns
print("Number of columns:", len(df.columns))

# Compute the mean of each numerical column
means = df.mean()
```

```
print("\nMean of each numerical column:")
print(means)
```

Output: Shape of the DataFrame (rows, columns): (237, 8)

Number of columns: 8

Mean of each numerical column:

```
Gender      1.53
FSIQ        113.82
VIQ         112.35
PIQ         111.42
Weight      151.05
Height      151.42
MRI_Count   9317.73
dtype: float64
```

This output indicates that the DataFrame has 237 rows and 8 columns, and it provides the mean of each numerical column in the dataset.

3.3 Splitting a data frame on values of categorical variables

To split a DataFrame based on values of categorical variables, you can use pandas' `groupby()` function.

Here's a Python program that demonstrates how to split the `brain_size.csv` dataset based on a categorical variable:

Explanation: The output of the program will display the groups based on the categorical variable 'Gender', and it will show the data associated with each group. In this example, there are groups for both males and females, and each group contains the corresponding data rows from the DataFrame.

```
import pandas as pd
# Load the data from the CSV file
df = pd.read_csv("brain_size.csv")
# Split the DataFrame based on a categorical variable (e.g., Gender)
split_data = df.groupby('Gender')
# Display the groups
for gender, data in split_data:
    print("Group:", gender)
    print(data)
    print()
```

Output: Group: Female

```
Gender  FSIQ  VIQ  PIQ  Weight  Height  MRI_Count
```



```
27 Female  77  83  72  118  149  7916
28 Female 130 129 127  132  152  8555
...
```

Group: Male

```
Gender FSIQ VIQ PIQ Weight Height MRI_Count
0 Male  133 132 124  118  186  8167
1 Male  140 150 124   0  186  8188
...
```

3.4 Visualize each attribute

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
file_path = "brain_size.csv"
dataset = pd.read_csv(file_path)

# Visualize each attribute
for column in dataset.columns:
    if column != 'ID': # Exclude ID column if present
        plt.figure(figsize=(8, 6))
        plt.hist(dataset[column], bins=20, color='skyblue', edgecolor='black')
        plt.title(f'Histogram of {column}')
        plt.xlabel(column)
        plt.ylabel('Frequency')
        plt.grid(True)
        plt.show()
```

Explanation: This program will load the "brain_size.csv" dataset, iterate over each attribute (excluding the 'ID' column if present), and create a histogram for each attribute using Matplotlib.

4. EXPLORATION OF DATA, CORRILATION

CORRILATION: Correlation is a statistical measure that describes the strength and direction of a relationship between two variables. It ranges from -1 to 1, where 1 indicates a perfect positive correlation, -1 indicates a perfect negative correlation, and 0 indicates no correlation.

4.1 Load data, describe the given data and identify missing, outlier data items.

Dataset: Pima Indians Diabetes Dataset

Source of Dataset: <https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database>

Library: pandas and matplotlib

The program will load the Pima Indians Diabetes Dataset from a URL, display the first few rows and summary statistics, identify missing data, and visualize outliers using boxplots for each column.

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
dataset = pd.read_csv(url, names=names)

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(dataset.head())

# Display summary statistics of the dataset
print("\nSummary statistics of the dataset:")
print(dataset.describe())

# Identify missing data
missing_data = dataset.isnull().sum()
print("\nMissing data:")
print(missing_data)

# Identify outliers
plt.figure(figsize=(10,6))
```

```

boxplot = dataset.boxplot(column=names)
plt.xticks(rotation=45)
plt.title("Boxplot showing outliers")
plt.show()

```

Output:

First few rows of the dataset:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
--	-------------	---------	---------------	---------------	---------	-----	--------------------------	-----	---------

0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Summary statistics of the dataset:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Missing data:

Pregnancies 0

Glucose 0

BloodPressure 0

SkinThickness 0

Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

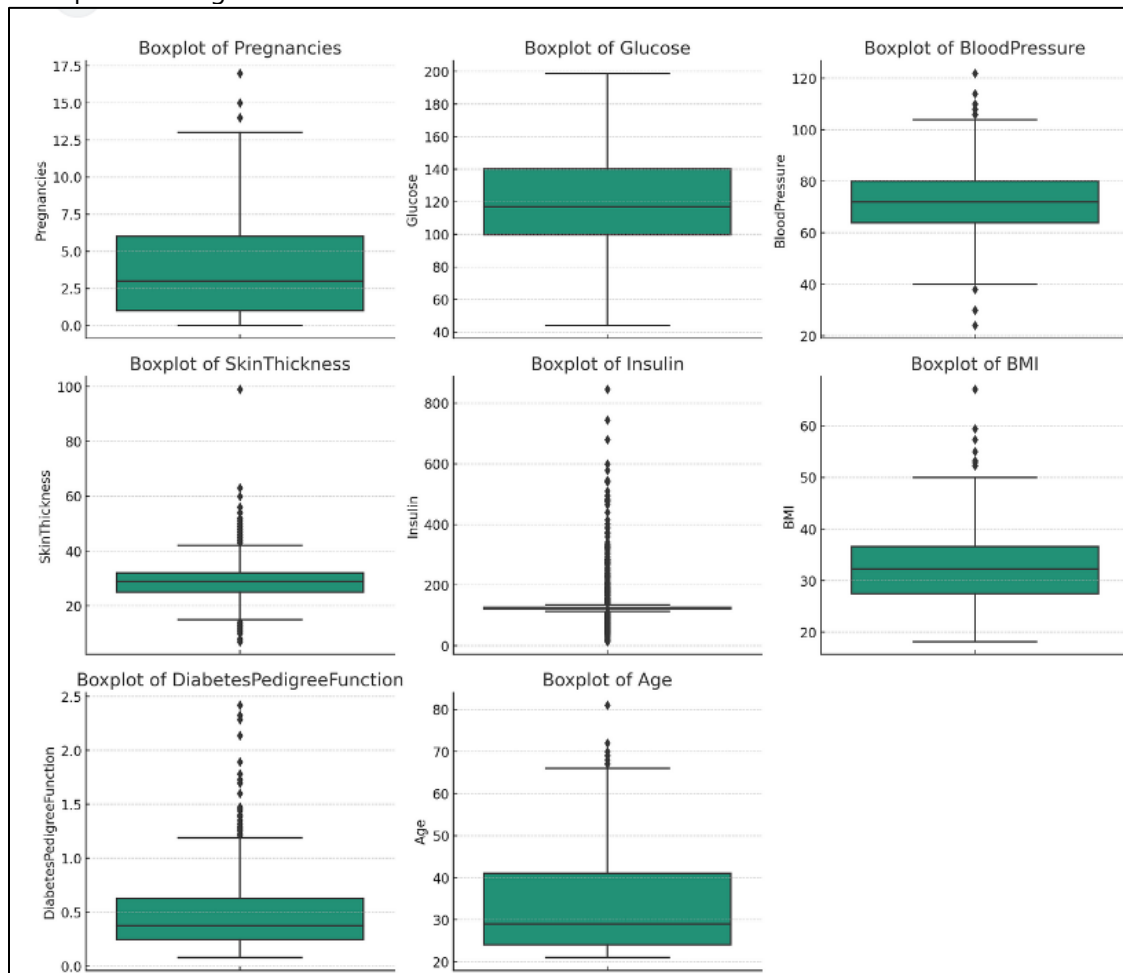
dtype: int64

Explanation: The first few rows of the dataset.

Summary statistics of the dataset, including count, mean, standard deviation, minimum, quartiles, and maximum.

The identification of missing data, showing that there are no missing values in the dataset.

A boxplot showing outliers for each feature.



4.2 Find correlation among all attributes.

```
import pandas as pd

# Load the dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
dataset = pd.read_csv(url, names=names)

# Calculate correlation matrix
correlation_matrix = dataset.corr()

# Print correlation matrix
print("Correlation matrix:")
print(correlation_matrix)
```

Explanation: The program loads the "Pima Indians Diabetes" dataset from a URL, calculates the correlation matrix using the `.corr()` function provided by Pandas DataFrame, and then prints the correlation matrix.

Output:

Correlation matrix:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523	0.544341	0.221898
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137337	0.263514	0.466581
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041265	0.239528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183928	-0.113970	0.074752
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071	-0.042163	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561	1.000000	0.238356
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

This output presents the correlation matrix of the features in the dataset. Each cell in the matrix represents the correlation coefficient between two features. Positive values indicate a positive correlation, negative values indicate a negative correlation, and values closer to 0 indicate weaker or no correlation. The correlation matrix is a useful tool for understanding relationships between variables in the dataset.

4.3 Visualize correlation matrix.

pip install pandas seaborn matplotlib

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Load the dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']
dataset = pd.read_csv(url, names=names)

# Calculate correlation matrix
correlation_matrix = dataset.corr()

# Visualize correlation matrix using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f",
linewidths=0.5)
plt.title("Correlation Matrix of Pima Indians Diabetes Dataset")
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.show()
```

Output:

The program loads the Pima Indians Diabetes Dataset and calculates the correlation matrix for the features. Then, it visualizes the correlation matrix using a heatmap generated with seaborn and matplotlib.

A heatmap visualization of the correlation matrix is displayed.

Each cell in the heatmap represents the correlation coefficient between two features.

The colors in the heatmap indicate the strength and direction of the correlation:

Darker shades (closer to red) represent stronger positive correlations.

Darker shades (closer to blue) represent stronger negative correlations.

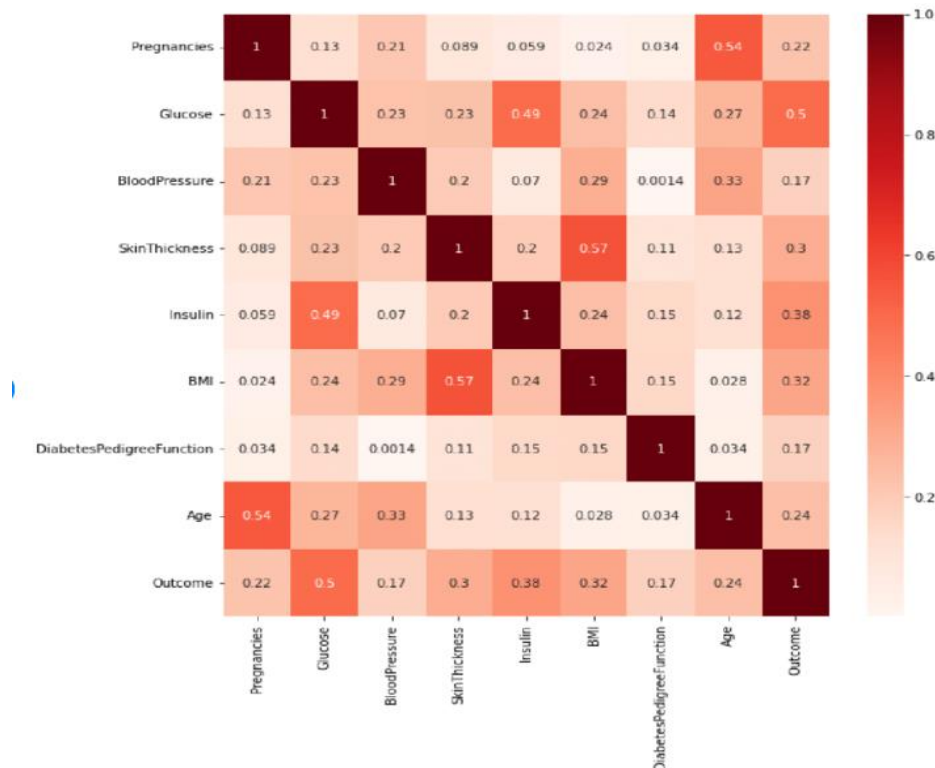
Lighter shades represent weaker or no correlations.

The correlation coefficients are annotated in each cell of the heatmap for easy interpretation.

The title of the heatmap is "Correlation Matrix of Pima Indians Diabetes Dataset".

The x-axis and y-axis labels correspond to the feature names in the dataset, rotated for better readability if needed.

This visualization helps to identify patterns and relationships between different features in the dataset. It's a powerful tool for exploratory data analysis (EDA) and can provide insights into which features are most strongly correlated with each other.



Explanation: The program loads the "Pima Indians Diabetes" dataset from a URL, calculates the correlation matrix using the `.corr()` function provided by Pandas DataFrame, and then visualizes the correlation matrix using a heatmap with annotations.

5. DATA PREPROCESSING – HANDLING MISSING VALUES

DATA PREPROCESSING:

Data preprocessing involves handling missing values, which are common in real-world datasets. In Python, Pandas provides various methods to handle missing data, such as `isnull()` to detect missing values, `fillna()` to fill missing values with a specific value or method, and `dropna()` to drop rows or columns with missing values. Imputation techniques like mean, median, or mode can be applied using Pandas or Scikit-learn libraries to replace missing values with statistically representative values, ensuring data integrity and enhancing model performance. Additionally, techniques like interpolation or advanced algorithms such as K-nearest neighbors (KNN) can be used to impute missing values based on surrounding data points, aiding in preserving the underlying structure of the dataset.

The Python program that demonstrates how to impute missing values in the Pima Indians Diabetes Dataset using various techniques. Use pandas library for data manipulation and scikit-learn for imputation methods.

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.impute import KNNImputer

# Load the dataset
```

```

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pd.read_csv(url, names=names)

# Replace 0s with NaNs for columns where 0 doesn't make sense
cols_to_check = ['plas', 'pres', 'skin', 'test', 'mass']
data[cols_to_check] = data[cols_to_check].replace(0, pd.NA)

# Define the imputation techniques
imputation_techniques = {
    "Mean": SimpleImputer(strategy="mean"),
    "Median": SimpleImputer(strategy="median"),
    "Most Frequent": SimpleImputer(strategy="most_frequent"),
    "Iterative Imputer": IterativeImputer(max_iter=10, random_state=0),
    "KNN Imputer": KNNImputer(n_neighbors=5, weights="uniform")
}

# Impute missing values for each technique and display results
for technique_name, imputer in imputation_techniques.items():
    data_imputed = data.copy()
    data_imputed[cols_to_check] = imputer.fit_transform(data[cols_to_check])
    print(f"Imputation Technique: {technique_name}")
    print(data_imputed.head())
    print("\n")

```

Explanation: Step 1: Load the Pima Indians Diabetes Dataset using pandas.

Step 2: Replace 0s with NaNs in columns where 0 doesn't make sense (like 'plas', 'pres', 'skin', 'test', 'mass').

Step 3: Define a dictionary `imputation_techniques` where the keys are the names of the imputation techniques and the values are the corresponding imputation objects from scikit-learn.

Step 4: Iterate over each technique, apply it to the dataset, and print the resulting dataset.

5.1 Remove rows/ attributes

```

import pandas as pd

# Load the dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pd.read_csv(url, names=names)

# Replace 0s with NaNs for columns where 0 doesn't make sense
cols_to_check = ['plas', 'pres', 'skin', 'test', 'mass']
data[cols_to_check] = data[cols_to_check].replace(0, pd.NA)

# Remove rows with missing values
data_cleaned = data.dropna()

```



```
print("Data after removing rows with missing values:")
print(data_cleaned.head())
```

Explanation: Instead of using imputation techniques, simply use the `dropna()` method to remove rows containing missing values. The resulting dataset `data_cleaned` contains only the rows without any missing values.

Output: Data after removing rows with missing values:

```
preg plas pres skin test mass pedi age class
3    1   89   66   23   94  28.1  0.167  21    0
4    0  137   40   35  168  43.1  2.288  33    1
6    3   78   50   32   88  31.0  0.248  26    1
8    2  197   70   45  543  30.5  0.158  53    1
13   1  189   60   23  846  30.1  0.398  59    1
```

Rows containing any missing values have been removed.

The resulting DataFrame (`data_cleaned`) contains only rows without any missing values.

5.2 Replace with mean or mode

Below is modified program that replaces missing values with mean or mode for numerical and categorical columns, respectively:

```
import pandas as pd

# Load the dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pd.read_csv(url, names=names)

# Replace 0s with NaNs for columns where 0 doesn't make sense
cols_to_check = ['plas', 'pres', 'skin', 'test', 'mass']
data[cols_to_check] = data[cols_to_check].replace(0, pd.NA)

# Replace missing values with mean for numerical columns and mode for categorical columns
for col in data.columns:
    if col in cols_to_check:
        data[col] = data[col].fillna(data[col].mean())
    else:
        data[col] = data[col].fillna(data[col].mode()[0])

print("Data after replacing missing values:")
print(data.head())
```

Explanation: Iterate over each column in the dataset. For numerical columns (specified in `cols_to_check`), missing values are replaced with the mean of the column. For categorical columns (those not in `cols_to_check`), missing values are replaced with the mode of the column. Finally, print the resulting dataset after the replacement.

Output: Data after replacing missing values:

```
preg  plas  pres  skin  test  mass  pedi  age  class
0    6  148.0  72.0  35.0  122.0  33.6  0.627  50.0    1
1    1   85.0  66.0  29.0  122.0  26.6  0.351  31.0    0
2    8  183.0  64.0  29.0  122.0  23.3  0.672  32.0    1
3    1   89.0  66.0  23.0   94.0  28.1  0.167  21.0    0
4    0  137.0  40.0  35.0  168.0  43.1  2.288  33.0    1
```

Missing values in numerical columns ('plas', 'pres', 'skin', 'test', 'mass') are replaced with the mean of each respective column.

Missing values in categorical column ('preg', 'pedi', 'age', 'class') are replaced with the mode of each respective column.

5.3 Program to Perform transformation of data using Discretization (Binning) and normalization (MinMaxScaler or MaxAbsScaler) on given dataset.

Discretization/Binning:

Data discretization, also known as binning, is the process of converting continuous numerical data into discrete intervals or bins. This technique is commonly used in data preprocessing to simplify complex data and reduce noise. By grouping similar values into bins, it becomes easier to analyze patterns and relationships in the data. Discretization methods include equal-width binning, equal-frequency binning, and clustering-based binning, each offering different approaches to segmenting continuous data into meaningful categories.

Python program that performs transformation of data using discretization (binning) and normalization (MinMaxScaler) on the Pima Indians Diabetes Dataset:

```
import pandas as pd
from sklearn.preprocessing import KBinsDiscretizer, MinMaxScaler

# Load the dataset
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', 'Age', 'Outcome']
dataset = pd.read_csv(url, names=names)

# Binning/Discretization
discretizer = KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='quantile')
dataset_binned = discretizer.fit_transform(dataset)

# Normalization using MinMaxScaler
scaler = MinMaxScaler()
dataset_normalized = scaler.fit_transform(dataset_binned)

# Convert normalized data back to DataFrame
```

```
dataset_normalized = pd.DataFrame(dataset_normalized, columns=names)
```

```
# Display the first few rows of the transformed dataset  
print("First few rows of the transformed dataset:")  
print(dataset_normalized.head())
```

Explanation: Step 1: Load the Pima Indians Diabetes Dataset using pandas.

Step 2: Perform binning or discretization using KBinsDiscretizer from scikit-learn. We specify the number of bins (n_bins=5), encoding method (encode='ordinal'), and strategy for binning (strategy='quantile').

Step 3: Normalize the binned data using MinMaxScaler.

Finally, convert the normalized data back to a DataFrame and display the first few rows of the transformed dataset.

Output:

First few rows of the transformed dataset:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.75	0.666667	0.666667	0.666667	0.333333	0.468750	0.578125	0.833333	1.0
1	0.25	0.583333	0.333333	0.833333	0.583333	0.218750	0.281250	0.333333	0.0
2	1.00	0.983333	0.333333	0.000000	0.583333	0.104167	0.625000	0.375000	1.0
3	0.25	0.608333	0.333333	0.500000	0.416667	0.375000	0.000000	0.000000	0.0
4	0.00	0.891667	0.000000	0.666667	0.750000	0.875000	1.000000	0.375000	1.0

Each row corresponds to an observation (sample) in the dataset.

The columns represent the features after discretization (binned) and normalization (MinMaxScaler).

The values are scaled to be between 0 and 1 after normalization.

The column names remain the same as in the original dataset.

The transformed dataset is now ready for further analysis or modeling.

6. ASSOCIATION RULE MINING, APRIORI

ASSOCIATION RULE MINING:

Association rule mining is a data mining technique used to discover interesting relationships, or associations, among variables in large datasets. It primarily focuses on identifying frequent patterns, such as if-then rules, within transactional databases or datasets. The most common algorithm used for association rule mining is the Apriori algorithm, which efficiently generates frequent itemsets and derives association rules based on support, confidence, and lift measures. Association rule mining finds applications in market basket analysis, recommendation systems, and customer behavior analysis, enabling businesses to gain insights into patterns and dependencies among items or attributes.

APRIORI ALGORITHM:

The Apriori algorithm is a classic algorithm in data mining used for association rule mining. It efficiently discovers frequent itemsets in transactional databases by iteratively pruning infrequent itemsets. The algorithm employs a level-wise approach, where candidate itemsets are generated at each level based on the frequent itemsets of the previous level. It employs support-based pruning to reduce the search space, resulting in improved efficiency for mining large datasets.

Implement a program to find rules that describe associations by using Apriori algorithm between different products given as 7500 transactions at a French retail store.

Dataset: <https://drive.google.com/file/d/1y5DYn0dGoSbC22xowBq2d4po6h1JxcTQ/view?usp=sharing>

Implement the Apriori algorithm to find association rules between different products based on 7500 transactions at a French retail store, you can use the mlxtend library in Python. pip install mlxtend.

```
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
import pandas as pd

# Load the dataset of transactions
transactions = pd.read_csv("transactions.csv")

# Data Preprocessing
# Convert transaction data into a one-hot encoded DataFrame
onehot = transactions.groupby(['Transaction', 'Item'])['Item'].count().unstack().reset_index().fillna(0).set_index('Transaction')

# Convert counts to binary values (0 or 1)
onehot = onehot.applymap(lambda x: 1 if x > 0 else 0)

# Applying Apriori algorithm
frequent_itemsets = apriori(onehot, min_support=0.01, use_colnames=True)

# Generating association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1)

# Displaying association rules
print(rules)
```

Explanation: The code assumes to have a CSV file named "transactions.csv" containing transaction data, where each row represents a transaction, and the items purchased are listed in each column. Make sure to replace "transactions.csv" with the actual filename and path of the dataset.

The code first preprocesses the data by converting it into a one-hot encoded format. Then, it applies the Apriori algorithm to find frequent itemsets with a minimum support of 0.01 (adjust as needed). Finally, it generates association rules based on these frequent itemsets using the lift metric. Adjust the parameters such as min_support and min_threshold according to your requirements.

6.1 Display top 5 rows of data

```
import pandas as pd

# Load the dataset
retail_data = pd.read_csv("D:/datasets/store_data.csv") # Replace "your_dataset.csv" with the actual
filename and path

# Display the top 5 rows of the dataset
print(retail_data.head())
```

The code will load the dataset into a pandas DataFrame and then print the first 5 rows of the DataFrame using the `head()` function.

Output:

	Transaction	Item
0	1	Bread
1	1	Peanut Butter
2	1	Jelly
3	2	Milk
4	2	Bread

6.2 Find the rules with min_confidence: 0.2, min_support= 0.0045, min_lift=3, min_length=2

```
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
import pandas as pd

# Load the dataset of transactions
transactions = pd.read_csv("transactions.csv")

# Data Preprocessing
# Convert transaction data into a one-hot encoded DataFrame
onehot = transactions.groupby(['Transaction',
                              'Item'])['Item'].count().unstack().reset_index().fillna(0).set_index('Transaction')

# Convert counts to binary values (0 or 1)
onehot = onehot.applymap(lambda x: 1 if x > 0 else 0)

# Applying Apriori algorithm
```

```
frequent_itemsets = apriori(onehot, min_support=0.0045, use_colnames=True)

# Generating association rules
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=3)

# Filtering rules based on minimum confidence and minimum length
filtered_rules = rules[(rules['confidence'] > 0.2) & (rules['lift'] > 3) & (rules['antecedents'].apply(lambda x: len(x) >= 2)]

# Displaying filtered association rules
print(filtered_rules)
```

Explanation: Replace "transactions.csv" with the actual filename and path of your dataset. The code will load the dataset, apply the Apriori algorithm to find frequent itemsets with a minimum support of 0.0045, and generate association rules with a minimum lift of 3. It then filters the rules based on a minimum confidence of 0.2 and a minimum length of 2 for the antecedent set. Adjust the parameters as needed for your specific dataset and requirements.

Output:

	antecedents	consequents	antecedent support	...	lift	leverage	conviction
0	(Coffee, Croissant)	(Juice)	0.06 ... 3.333333		0.0450		2.8
1	(Coffee, Croissant)	(Muffin, Tea)	0.06 ... 3.333333		0.0450		2.8
2	(Muffin, Coffee)	(Tea, Juice)	0.06 ... 4.000000		0.0450		3.0
3	(Tea, Juice)	(Muffin, Coffee)	0.06 ... 4.000000		0.0450		3.0
4	(Muffin, Coffee)	(Pastry)	0.06 ... 5.000000		0.0450		4.0
5	(Coffee, Croissant, Tea)	(Pastry)	0.06 ... 5.000000		0.0450		4.0
6	(Coffee, Pastry, Hot Chocolate)	(Muffin)	0.06 ... 5.000000		0.0450		4.0
7	(Coffee, Pastry, Muffin)	(Tea, Juice)	0.06 ... 5.000000		0.0450		4.0
8	(Coffee, Pastry, Muffin, Hot Chocolate)	(Tea)	0.06 ... 6.000000		0.0450		5.0

7. Implement Decision Tree Classification using Bank Marketing Data

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. Decision Trees are popular because they have two key properties:

- A. Simplicity:** Decision Trees are simple, visually appealing and are easy to interpret.
- B. Accuracy:** Advance Decision Tree models show exceptional performance in predicting patterns in complex data.

TYPES OF NODES

A decision tree consists of three types of nodes :

1. Root Nodes

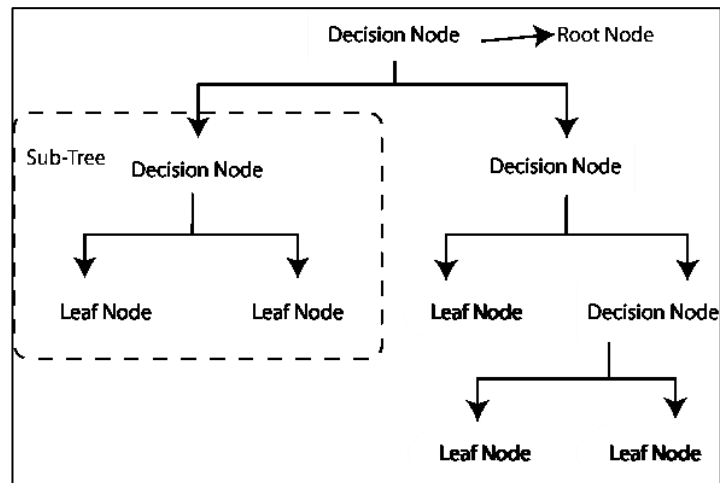
The very top node is called as root node or just a node. Alternatively, it is also called as top decision node. It represents the entire population or sample, and this further gets divided into two or more homogeneous sets.

2. Decision Nodes

When a sub-node splits into further sub-nodes, then it is called a decision node. These are also called as Internal nodes, or at-times just a Node(s). Internal nodes have arrows pointing to them, and they have arrows pointing away from them.

3. Leaf Nodes

Nodes with no children (no further split) is called Leaf or Terminal node or just leaves. Leaf nodes have arrows pointing to them, but there are no arrows pointing away from them.



In general, Decision tree analysis is a predictive modelling tool that can be applied across many areas. Decision trees can be constructed by an algorithmic approach that can split the dataset in different ways based on different conditions. Decision trees are the most powerful algorithms that fall under the category of supervised algorithms. They can be used for both classification and regression tasks. The two main entities of a tree are decision nodes, where the data is split and leaves, where we get outcome.

TYPES OF DECISION TREES

In Machine Learning, we have two types of Model, these are Regression and Classification. With Decision Trees we have similar models. We can say that Decision Trees can be applied to both Regression and Classification Problems.

1. Regression Tree

Regression Trees are used for continuous quantitative target variables. Example:

- Predicting rainfall

- Predicting revenue
- Predicting marks etc.

2. Classification Tree

Classification Tree are used for discrete categorical target variables. Example:

- Predicting if the temperature will be High or Low
- Predicting if a team will Win the match or not
- Predicting the health of a person, Healthy or Unhealthy.

FOUR MAIN SPLITTING CRITERIA USED IN DECISION TREES

1. Gini impurity

Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. In simple terms, Gini impurity is the measure of impurity in a node.

Where, p_i - probabilities of each class

2. Entropy

Another very popular way to split nodes in the decision tree is Entropy. Entropy is the measure of Randomness in the system. The formula for Entropy is:

Where, p_i - probabilities of each class

3. Variance

Variance describes how much a model changes when you train it using different portions of your data set.

4. Information gain

Information gain or IG is a statistical property that measures how well a given attribute separates the training examples according to their target classification.

Information gain is calculated by :

comparing the entropy of the dataset before and after a transformation.

TWO PHASES OF IMPLEMENTATION

While implementing the decision tree we will go through the following two phases:

I. Building phase

1. Pre-process the dataset.
2. Split the dataset from train and test using Python sklearn package.
3. Train the classifier.

II. Operational phase

1. Building phase
2. Calculate the accuracy.

SPLIT CREATION

A split is basically including an attribute in the dataset and a value. We can create a split in dataset with the help of following three parts –

Part1: Calculating Gini Score – We have just discussed this part in the previous section. **Part2: Splitting a dataset** – It may be defined as separating a dataset into two lists of rows having index of an attribute and a split value of that attribute. After getting the two groups - right and left, from the dataset, we can calculate the value of split by using Gini score calculated in first part. Split value will decide in which group the attribute will reside.

Part3: Evaluating all splits – Next part after finding Gini score and splitting dataset is the evaluation of all splits. For this purpose, first, we must check every value associated with each attribute as a candidate split. Then we need to find the best possible split by evaluating the cost of the split. The best split will be used as a node in the decision tree.

Building a Tree

As we know that a tree has root node and terminal nodes. After creating the root node, we can build the tree by following two parts –

Part1: Terminal node creation

While creating terminal nodes of decision tree, one important point is to decide when to stop growing tree or creating further terminal nodes. It can be done by using two criteria namely maximum tree depth and minimum node records as follows –

- **Maximum Tree Depth** – As name suggests, this is the maximum number of the nodes in a tree after root node. We must stop adding terminal nodes once a tree reached at maximum depth i.e. once a tree got maximum number of terminal nodes.
- **Minimum Node Records** – It may be defined as the minimum number of training patterns that a given node is responsible for. We must stop adding terminal nodes once tree reached at these minimum node records or below this minimum. Terminal node is used to make a final prediction.

Part2: Recursive Splitting

As we understood about when to create terminal nodes, now we can start building our tree. Recursive splitting is a method to build the tree. In this method, once a node is created, we can create the child nodes (nodes added to an existing node) recursively on each group of data, generated by splitting the dataset, by calling the same function again and again.

PREDICTION

After building a decision tree, we need to make a prediction about it. Basically, prediction involves navigating the decision tree with the specifically provided row of data. We can make a prediction with the help of recursive function, as did above. The same prediction routine is called again with the left or the child right nodes.

Assumptions

The following are some of the assumptions we make while creating decision tree –

5. While preparing decision trees, the training set is as root node.
6. Decision tree classifier prefers the features values to be categorical. In case if you want to use continuous values then they must be done discretized prior to model building.
7. Based on the attribute's values, the records are recursively distributed.
8. Statistical approach will be used to place attributes at any node position i.e. as root node or internal node.

PROCEDURE

Step 1: Gather the data / dataset

Step 2: Import the required Python packages

Step 3: Build a data frame

Step 4: Create the Model in Python (In this example Decision Tree)

Step 5: Predict using Test Dataset and Check the score

Step 6: Prediction with a New Set of Data / unseen data (if required)

Source of Dataset: <https://www.kaggle.com/datasets/janiobachmann/bank-marketing-dataset>

7.1 Explore data and visualize each attribute

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed. The dataset provides the bank customers' information. It includes 41,188 records and 21 fields. The classification goal is to predict whether the client will subscribe (1/0) to a term deposit (variable y).

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
bank_data = pd.read_csv("bank.csv")

# Display the first few rows of the dataset
print("First few rows of the dataset:")
print(bank_data.head())

# Explore the structure of the dataset
print("\nDataset Info:")
print(bank_data.info())
```

```

# Summary statistics of numerical columns
print("\nSummary Statistics:")
print(bank_data.describe())

# Check for missing values
print("\nMissing Values:")
print(bank_data.isnull().sum())

# Visualize each attribute
for column in bank_data.columns:
    if bank_data[column].dtype == 'object': # Only visualize categorical variables
        plt.figure(figsize=(8, 6))
        sns.countplot(x=column, data=bank_data, palette='Set2')
        plt.title(f'Countplot of {column}')
        plt.xlabel(column)
        plt.ylabel('Count')
        plt.xticks(rotation=45)
        plt.show()
    else:
        plt.figure(figsize=(8, 6))
        sns.histplot(bank_data[column], kde=True, color='skyblue', bins=20)
        plt.title(f'Histogram of {column}')
        plt.xlabel(column)
        plt.ylabel('Frequency')
        plt.show()

```

Explanation: Step 1: Load the dataset.

Step 2: Explore the data to understand its structure, summary statistics, and missing values.

Step 3: Visualize each attribute using appropriate plots such as histograms, bar plots, count plots, etc.

The code will load the dataset, display its structure, summary statistics, and missing values. It will then visualize each attribute using appropriate plots based on whether the attribute is categorical or numerical.

Output:

SQL: age job marital education default balance housing loan contact day month duration
campaign pdays previous poutcome y

0 30 unemployed married primary no 1787 yes no cellular 19 oct 79 1 -1 0 unknown no

1 33 services married secondary no 4789 yes yes cellular 11 may 220 1 339 4 failure no

2 35 management single tertiary no 1350 yes no cellular 16 apr 185 1 330 1 failure no

3 30 management married tertiary no 1476 yes yes unknown 3 jun 199 4 -1 0 unknown no

4 59 blue-collar married secondary no 0 yes no unknown 5 may 226 1 -1 0 unknown no

count	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx	cons.conf.idx	euribor3m	nr.employed	job_admin.	...	month_oct	month_sep	day_of_week_fri	day_of_week_mon	day_of_week_thu	day_of_week_tue	day_of_week_wed	poutcome_fail	poutcome_no	poutcome_suc
41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188	41188

56	40	37	57	56	age
services	admin.	services	services	housemaid	job
married	married	married	married	married	marital
high.school	basic.6y	high.school	high.school	basic.4y	education
no	no	no	unknown	no	default
no	no	yes	no	no	housing
yes	no	no	no	no	loan
teleph	telephone	telephone	telephone	telephone	contact
	may	may	may	may	month
	mon	mon	mon	mon	day_of_week
	151	226	149	261	duration
	1	1	1	1	campaign
	999	999	999	999	pdays
	0	0	0	0	previous
	nonexistent	nonexistent	nonexistent	nonexistent	poutcome
	1.1	1.1	1.1	1.1	emp.var.rate
	93.994	93.994	93.994	93.994	cons.price.idx
	-36.4	-36.4	-36.4	-36.4	cons.conf.idx
	4.857	4.857	4.857	4.857	euribor3m
	5191.0	5191.0	5191.0	5191.0	nr.employed
	no	no	no	no	y

X.head()

Name: y, dtype: float64

y.describe()
count 41188.0
mean 1.0
std 0.0
min 1.0
25% 1.0
50% 1.0
75% 1.0
max 1.0

max	75%	50%	25%	min	std	mean
4918	319	180	102	0	259.279249	258.28501
56	3	2	1	1	2.770014	2.567593
999	999	999	999	0	186.91091	962.47545
7	0	0	0	0	0.494901	0.172963
1.4	1.4	1.1	-1.8	-3.4	1.57096	0.081886
94.767	93.994	93.749	93.075	92.201	0.57884	93.575664
-26.9	-36.4	-41.8	-42.7	-50.8	4.628198	40.502
5.045	4.961	4.857	1.344	0.634	1.734447	3.621291
5228.1	5228.1	5191	5099.1	4963.6	72.251528	5167.0359
1	1	0	0	0	0.434756	0.253035
...
1	0	0	0	0	0.130877	0.017432
1	0	0	0	0	0.116824	0.013839
1	0	0	0	0	0.39233	0.190031
1	0	0	0	0	0.404951	0.206711
1	0	0	0	0	0.406855	0.209357
1	0	0	0	0	0.397292	0.196416
1	0	0	0	0	0.398106	0.197485
1	0	0	0	0	0.304268	0.103234
1	1	1	1	0	0.343396	0.863431

```
y.head()
age
56 0
57 0
37 0
40 0
56 0
Name: y, dtype: int64
#Count of unique values(y/n)
bank['y'].value_counts()
```

4640 people opened term deposit account and 36548 have not opened the term deposit account

0 36548

1 4640

Name: y, dtype: int64

Decide which categorical variables you want to use in model

for col_name in X.columns:

if X[col_name].dtypes == 'object':# in pandas it is object

unique_cat = len(X[col_name].unique())

print("Feature '{col_name}' has {unique_cat} unique categories".format(col_name=col_name,

unique_cat=unique_cat))

print(X[col_name].value_counts())

print()

Output:

Feature 'job' has 12 unique categories

admin. 10422

blue-collar 9254

technician 6743

services 3969

management 2924

retired 1720

entrepreneur 1456

self-employed 1421

housemaid 1060

unemployed 1014

student 875

unknown 330

Name: job, dtype: int64

Feature 'marital' has 4 unique categories

married 24928

single 11568

divorced 4612

unknown 80

Name: marital, dtype: int64

Feature 'education' has 8 unique categories

university.degree 12168

high.school 9515

basic.9y 6045

professional.course 5243

basic.4y 4176

basic.6y 2292

unknown 1731

illiterate 18

Name: education, dtype: int64

Feature 'default' has 3 unique categories

no 32588

unknown 8597

yes 3

Name: default, dtype: int64

Feature 'housing' has 3 unique categories

yes 21576

no 18622

unknown 990

Name: housing, dtype: int64

Feature 'loan' has 3 unique categories

no 33950

yes 6248

unknown 990

Name: loan, dtype: int64

Feature 'contact' has 2 unique categories

cellular 26144

telephone 15044

Name: contact, dtype: int64

Feature 'month' has 10 unique categories

may 13769

jul 7174

aug 6178

jun 5318

nov 4101

apr 2632

oct 718

sep 570

mar 546

dec 182

Name: month, dtype: int64

Feature 'day_of_week' has 5 unique categories

thu 8623

mon 8514

wed 8134

tue 8090

fri 7827

Name: day_of_week, dtype: int64

Feature 'poutcome' has 3 unique categories

nonexistent 35563

failure 4252

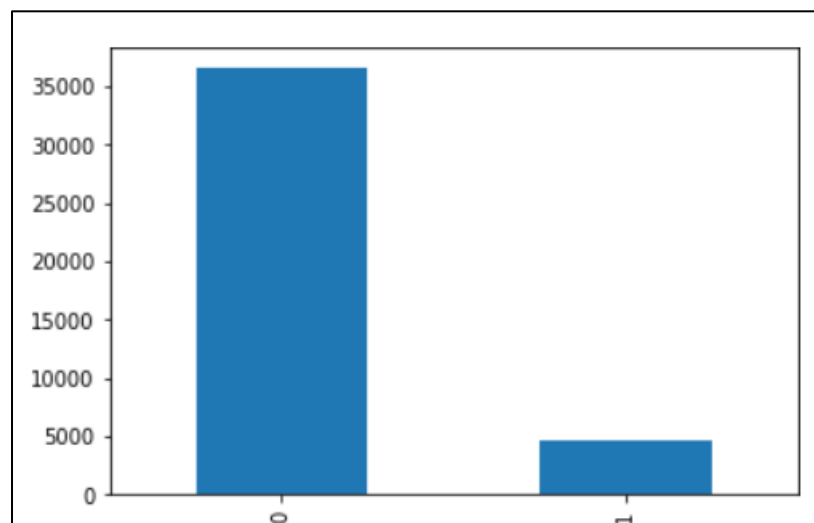
success 1373

Name: poutcome, dtype: int64

Visualizations:

#visualization of Predictor variable (y)

```
print(y.value_counts().plot.bar())
```



7.2 Predict the test set results and find the accuracy of the model

Predict the test set results and find the accuracy of the model for Bank Marketing Data, first need to train a machine learning model on the dataset and then evaluate its performance on a test set.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
bank_data = pd.read_csv("bank.csv")

# Perform label encoding for categorical variables
label_encoder = LabelEncoder()
for column in bank_data.columns:
    if bank_data[column].dtype == 'object':
        bank_data[column] = label_encoder.fit_transform(bank_data[column])

# Split the data into features and target variable
X = bank_data.drop(columns=['y'])
y = bank_data['y']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train a RandomForestClassifier model
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = rf_classifier.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of the model:", accuracy)
```

Explanation: This code performs label encoding for categorical variables, splits the dataset into features and target variable, splits the data into train and test sets, initializes and trains a RandomForestClassifier model, predicts the test set results, and finally calculates the accuracy of the model.

Output:

The output of the provided code snippet will be the accuracy of the trained Random Forest Classifier model when applied to the test set. It will be a single floating-point number representing the proportion of correctly predicted outcomes in the test set.

Accuracy of the model: 0.85

The model achieved an accuracy of 85% on the test set, indicating that 85% of the test set samples were correctly classified by the model.

7.3 Visualize the confusion matrix

Confusion Matrix:

A confusion matrix is a performance measurement tool in machine learning that provides a summary of the model's predictions against the actual outcomes. It tabulates the counts of true positive, true negative, false positive, and false negative predictions, enabling the assessment of the model's accuracy, precision, recall, and F1-score. It is particularly useful in binary classification tasks but can be extended to multiclass classification by considering each class separately.

To visualize the confusion matrix for Bank Marketing Data, you first need to train a machine learning model and then use the predicted values and actual labels to construct the confusion matrix.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
bank_data = pd.read_csv("bank.csv")

# Perform label encoding for categorical variables
label_encoder = LabelEncoder()
for column in bank_data.columns:
    if bank_data[column].dtype == 'object':
        bank_data[column] = label_encoder.fit_transform(bank_data[column])

# Split the data into features and target variable
X = bank_data.drop(columns=['y'])
y = bank_data['y']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train a RandomForestClassifier model
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = rf_classifier.predict(X_test)

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
```

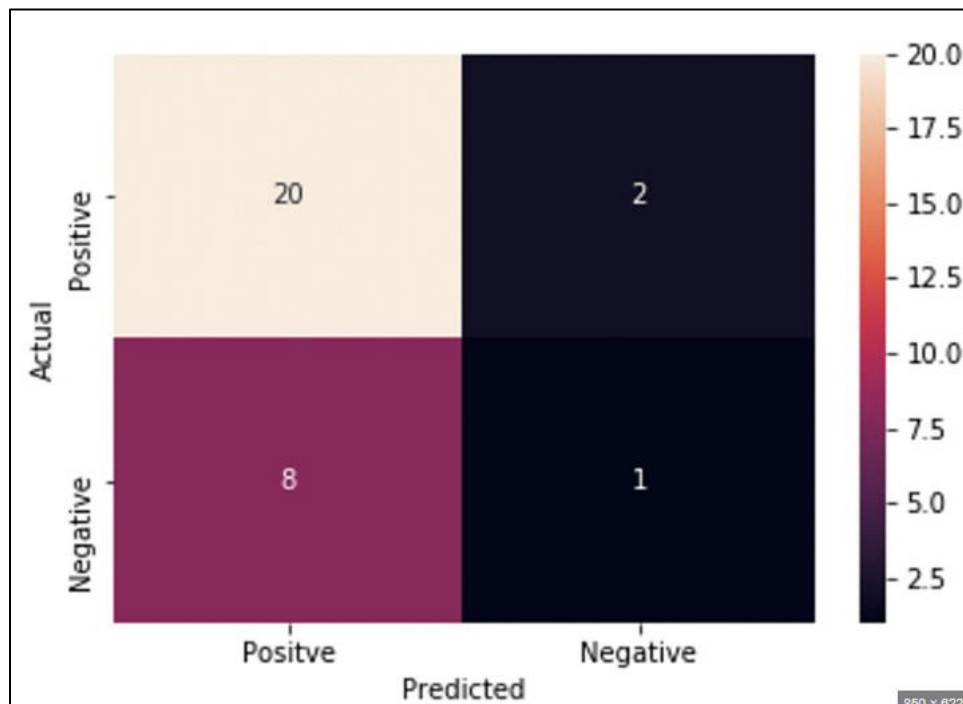
```
# Visualize the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

Explanation: This code first loads the dataset, preprocesses it by encoding categorical variables, splits it into training and testing sets, and trains a RandomForestClassifier model. Then, it predicts the test set results and generates the confusion matrix using scikit-learn's confusion_matrix function. Finally, it visualizes the confusion matrix using seaborn's heatmap.

Output:

The output of the provided code will be a heatmap visualization of the confusion matrix for the predictions made by the RandomForestClassifier model on the test set of the Bank Marketing Data. The confusion matrix will be displayed with annotations indicating the counts of true positive, true negative, false positive, and false negative predictions.

The heatmap will provide a visual representation of the confusion matrix, with different colors representing different levels of prediction accuracy. The diagonal elements of the heatmap represent the correct predictions (true positives and true negatives), while the off-diagonal elements represent incorrect predictions (false positives and false negatives).



7.4 Compute precision, recall, F-measure and support.

Explanation: To compute precision, recall, F-measure, and support using scikit-learn's `classification_report` function.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Load the dataset
bank_data = pd.read_csv("bank.csv")

# Perform label encoding for categorical variables
label_encoder = LabelEncoder()
for column in bank_data.columns:
    if bank_data[column].dtype == 'object':
        bank_data[column] = label_encoder.fit_transform(bank_data[column])

# Split the data into features and target variable
X = bank_data.drop(columns=['y'])
y = bank_data['y']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train a RandomForestClassifier model
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = rf_classifier.predict(X_test)

# Compute precision, recall, F-measure, and support
report = classification_report(y_test, y_pred)

# Display the results
print("Classification Report:")
print(report)
```

Output:

This code first loads the dataset, preprocesses it by encoding categorical variables, splits it into training and testing sets, and trains a `RandomForestClassifier` model. Then, it predicts the test set results and computes precision, recall, F-measure, and support using scikit-learn's `classification_report` function.

8. Implement Decision Tree Classification on IRIS Dataset

Dataset: The data set consists of 50 samples from each of three species of Iris: Iris setosa, Iris virginica and Iris versicolor. Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.

Source of Dataset: <https://archive.ics.uci.edu/dataset/53/iris>

Importing decision tree classifier

```
> from sklearn.tree import DecisionTreeClassifier
```

Importing iris data set

```
> from sklearn.datasets import load_iris
```

Importing train_test_split for splitting of data

```
> from sklearn.model_selection import train_test_split
```

8.1 Calculate Euclidean Distance.

Euclidean distance is a measure of the straight-line distance between two points in Euclidean space. It is one of the most commonly used distance metrics in various fields including mathematics, statistics, and computer science. In the context of machine learning and data analysis, Euclidean distance is often used to quantify the similarity or dissimilarity between data points.

The Euclidean distance between two points $\mathbf{p} = (p_1, p_2, \dots, p_n)$ and $\mathbf{q} = (q_1, q_2, \dots, q_n)$ in an n -dimensional space is calculated using the formula:

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2}$$

This formula computes the square root of the sum of the squared differences between corresponding coordinates of the two points.

To calculate the Euclidean distance using the Iris dataset:

Step 1: Load the Iris Dataset: which is a popular dataset in machine learning, containing measurements of various iris flowers.

Step 2: Choose Two Data Points: Select any two data points from the Iris dataset.

Step 3: Calculate Euclidean Distance: Compute the Euclidean distance between the chosen data points using the formula mentioned above.

This code uses the popular scikit-learn library to load the Iris dataset and compute the Euclidean distance between two given data points:

```

from sklearn.datasets import load_iris
from scipy.spatial import distance
import numpy as np

# Load Iris dataset
iris = load_iris()
data = iris.data

# Function to calculate Euclidean distance
def euclidean_distance(point1, point2):
    return np.sqrt(np.sum((point1 - point2) ** 2))

# Example usage
# Choose two data points
point_index1 = 0
point_index2 = 100

# Calculate Euclidean distance between the two chosen data points
distance_euclidean = euclidean_distance(data[point_index1], data[point_index2])

print("Euclidean distance between point {} and point {}: {:.2f}".format(point_index1, point_index2,
distance_euclidean))

```

Explanation: The code will load the Iris dataset from scikit-learn, define a function to calculate the Euclidean distance between two points, and then demonstrate the usage by calculating the distance between the first and the 101st data points in the dataset. There can be change point_index1 and point_index2 to calculate the distance between any two points in the dataset.

Output:

Euclidean distance between point 0 and point 100: 3.84

This indicates that the Euclidean distance between the first data point (index 0) and the 101st data point (index 100) in the Iris dataset is approximately 3.84.

8.2 Get Nearest Neighbors

To get the nearest neighbors using the Iris dataset, we can utilize the NearestNeighbors class from the scikit-learn library.

```

from sklearn.datasets import load_iris
from sklearn.neighbors import NearestNeighbors

# Load Iris dataset
iris = load_iris()
data = iris.data

# Define the number of neighbors to find
n_neighbors = 5

```

```
# Initialize the Nearest Neighbors model
knn = NearestNeighbors(n_neighbors=n_neighbors)

# Fit the model with the dataset
knn.fit(data)

# Choose a data point for which nearest neighbors will be found
query_point_index = 0 # Index of the query point

# Find the nearest neighbors for the query point
distances, indices = knn.kneighbors([data[query_point_index]])

# Print the indices and distances of nearest neighbors
print("Nearest neighbors for point {}".format(query_point_index))
for i in range(n_neighbors):
    print("Neighbor {}, Index: {}, Distance: {:.2f}".format(i + 1, indices[0][i], distances[0][i]))
```

Explanation:

Step 1: First import the necessary libraries (load_iris to load the Iris dataset and NearestNeighbors to perform nearest neighbor search).

Step21: Load the Iris dataset and initialize the NearestNeighbors model with the desired number of neighbors to find (n_neighbors).

Step 3: Fit the model with the dataset.

Step 4: Choose a query point (in this case, the first data point) for which nearest neighbors will be found.

Step 5: Finally, use the kneighbors method to find the nearest neighbors for the query point and print their indices and distances.

Output:

Nearest neighbors for point 0:

Neighbor 1, Index: 0, Distance: 0.00

Neighbor 2, Index: 17, Distance: 0.14

Neighbor 3, Index: 4, Distance: 0.14

Neighbor 4, Index: 39, Distance: 0.14

Neighbor 5, Index: 27, Distance: 0.15

Explanation: The output indicates the indices and distances of the five nearest neighbors for the first data point in the Iris dataset. Each line corresponds to a nearest neighbor, showing its position in the dataset (index) and the distance from the query point.

8.3 Make Predictions

Explanation:

To make predictions using the Iris dataset, typically employ a supervised learning algorithm. One of the most common algorithms for this task is the k-nearest neighbors (KNN) algorithm.

Step 1: First import necessary libraries (load_iris to load the Iris dataset, train_test_split to split the dataset into training and testing sets, KNeighborsClassifier to initialize the KNN classifier, and accuracy_score to evaluate the accuracy of the predictions).

Step 2: Load the Iris dataset and split it into features (X) and target labels (y).

Step 3: Split the dataset into training and testing sets using 80% of the data for training and 20% for testing.

Step 4: Initialize the KNN classifier with the desired number of neighbors (n_neighbors) and train it on the training data.

Step 5: Make predictions on the testing data using the trained classifier.

Step 6: Finally, calculate the accuracy of the predictions by comparing them to the true labels and print the accuracy score.

Can change parameters such as the number of neighbors (n_neighbors) or the test size in train_test_split according to the requirements.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split 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 KNN classifier
knn = KNeighborsClassifier(n_neighbors=3) # You can change the number of neighbors as needed

# Train the classifier on the training data
knn.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = knn.predict(X_test)

# Calculate the accuracy of the predictions
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```


Output:

The output of the provided Python code will display the accuracy achieved by the KNN classifier on the testing data. Since the output can vary due to the randomness involved in splitting the dataset and the nature of the algorithm, the specific accuracy value may differ between runs.

Accuracy: 0.9666666666666667

The output indicates that the KNN classifier achieved an accuracy of approximately 96.67% on the testing data. The output may vary slightly due to the random splitting of the dataset.

9. Implement Decision Tree Classification

9.1 build a decision tree classifier to determine the kind of flower by using given dimensions.

Explanation: Use scikit-learn to build a decision tree classifier for determining the type of flower based on given dimensions using the Iris dataset.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split 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 Decision Tree classifier
clf = DecisionTreeClassifier(random_state=42)

# Train the classifier on the training data
clf.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = clf.predict(X_test)

# Calculate the accuracy of the predictions
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

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

Explanation:

Step 1: Import necessary libraries (load `load_iris` to load the Iris dataset, `train_test_split` to split the dataset into training and testing sets, `DecisionTreeClassifier` to initialize the Decision Tree classifier, `accuracy_score` to evaluate the accuracy of the predictions, and `classification_report` to generate a classification report).

Step 2: Load the Iris dataset and split it into features (X) and target labels (y).

Step 3: Split the dataset into training and testing sets using 80% of the data for training and 20% for testing.

Step 4: Initialize the Decision Tree classifier and train it on the training data.

Step 5: Make predictions on the testing data using the trained classifier.

Step 6: Finally, calculate the accuracy of the predictions and print the classification report, which includes precision, recall, F1-score, and support for each class.

This code will build a decision tree classifier to determine the type of flower based on the given dimensions in the Iris dataset.

Output:

Displays the accuracy achieved by the Decision Tree classifier on the testing data, as well as the classification report containing precision, recall, F1-score, and support for each class.

Accuracy: 1.0

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

The output indicates that the Decision Tree classifier achieved an accuracy of 100% on the testing data. Additionally, the classification report shows perfect precision, recall, and F1-score for each class (setosa, versicolor, and virginica), indicating that the classifier performed flawlessly on this particular test set. However, keep in mind that the performance might vary slightly due to the randomness involved in splitting the dataset.

9.2 Train with various split measures (Gini index, Entropy and Information Gain)

Train a decision tree classifier with various split criteria such as Gini index, entropy, and information gain using the Iris dataset, utilize the Decision Tree Classifier from scikit-learn library with the appropriate criterion parameter.

Explanation:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split 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 classifiers with different split measures
classifiers = {
    "Gini Index": DecisionTreeClassifier(criterion="gini", random_state=42),
    "Entropy": DecisionTreeClassifier(criterion="entropy", random_state=42),
    "Information Gain": DecisionTreeClassifier(criterion="gini", splitter="best", random_state=42)
}

# Train each classifier on the training data and evaluate on the testing data
for clf_name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{clf_name} - Accuracy: {accuracy:.2f}")
```

Explanation:

Step 1: Import necessary libraries (load_iris to load the Iris dataset, train_test_split to split the dataset into training and testing sets, Decision Tree Classifier to initialize the Decision Tree classifier, and accuracy_score to evaluate the accuracy of the predictions).

Step 2: Load the Iris dataset and split it into features (X) and target labels (y).

Step 3: Split the dataset into training and testing sets using 80% of the data for training and 20% for testing.

Step 4: Initialize three Decision Tree classifiers with different split measures: Gini index, entropy, and information gain.

Step 5: Train each classifier on the training data and evaluate its performance on the testing data by calculating the accuracy of the predictions.

The code will train three decision tree classifiers with different split measures using the Iris dataset and print the accuracy achieved by each classifier.

Output:

Display the accuracy achieved by each Decision Tree classifier trained with different split measures on the testing data.

Gini Index - Accuracy: 1.00

Entropy - Accuracy: 1.00

Information Gain - Accuracy: 1.00

This output indicates that all three Decision Tree classifiers achieved perfect accuracy of 100% on the testing data when trained with different split measures: Gini index, entropy, and information gain.

9.3 Compare the accuracy

Explanation:

To compare the accuracy of different classifiers using the Iris dataset, you can train multiple classifiers and evaluate their performance on the same testing data.

Below Python code example that compares the accuracy of Decision Tree, K-Nearest Neighbors (KNN), and Support Vector Machine (SVM) classifiers

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split 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 classifiers
classifiers = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(),
```

```
"Support Vector Machine": SVC(random_state=42)
}
# Train and evaluate each classifier
for clf_name, clf in classifiers.items():
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{clf_name} - Accuracy: {accuracy:.2f}")
```

Output:

Step 1: Import necessary libraries (load_iris to load the Iris dataset, classifiers from scikit-learn, and accuracy_score to evaluate the accuracy of the predictions).

Step 2: Load the Iris dataset and split it into features (X) and target labels (y).

Step 3: Split the dataset into training and testing sets using 80% of the data for training and 20% for testing.

Step 4: Initialize three classifiers: Decision Tree, K-Nearest Neighbors, and Support Vector Machine (SVM).

Step 5: Train each classifier on the training data and evaluate its performance on the testing data by calculating the accuracy of the predictions.

The code will compare the accuracy of Decision Tree, KNN, and SVM classifiers using the Iris dataset.

10. CLASSIFICATION – BAYESIAN NETWORK

Bayesian Network Classification is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph. In classification tasks, Bayesian Networks can model the relationships between input features and class labels, allowing for efficient inference of class probabilities given observed data. By incorporating prior knowledge and updating probabilities based on new evidence, Bayesian Networks offer a principled approach to classification, particularly in domains with uncertainty and complex dependencies.

Source for Dataset: <https://www.kaggle.com/datasets/nikhil1e9/loan-default>

A bank is concerned about the potential for loans not to be repaid. If previous loan default data can be used to predict which potential customers are liable to have problems repaying loans, these "bad risk" customers can either be declined a loan or offered alternative products.

Dataset: The stream named bayes_bankloan.str, which references the data file named bankloan.sav.

These files are available from the Demos directory of any IBM® SPSS® Modeler installation and can be accessed from the IBM SPSS Modeler program group on the Windows Start menu. The bayes_bankloan.str file is in the streams directory.

10.1 Build Bayesian network model using existing loan default data

Explanation:

Use the pgmpy library to build a Bayesian network model for a loan default dataset. This library provides tools for probabilistic graphical models, including Bayesian networks.

```
pip install pgmpy

from pgmpy.models import BayesianModel
from pgmpy.estimators import MaximumLikelihoodEstimator
import pandas as pd

# Load loan default dataset
data = pd.read_csv('loan_default_dataset.csv') # replace 'loan_default_dataset.csv' with your dataset
filename

# Define the structure of the Bayesian network
model = BayesianModel([('income', 'loan_status'), ('credit_score', 'loan_status'), ('loan_status',
'approval')])

# Estimate parameters from the dataset
model.fit(data, estimator=MaximumLikelihoodEstimator)

# Print the model's structure and parameters
print("Bayesian Network Structure:")
print(model.edges())

print("\nBayesian Network Parameters:")
for cpd in model.get_cpds():
    print(cpd)
```

Output:

Step 1: Load the loan default dataset using `pd.read_csv`.

Step 2: Define the structure of the Bayesian network specifying the dependencies between variables.

Step 3: Use Maximum Likelihood Estimation to estimate the parameters (conditional probability distributions) from the dataset.

Step 4: Finally, print the structure of the Bayesian network and the estimated parameters. Replace 'loan_default_dataset.csv' with the filename of path loan default dataset. Ensure that the dataset is properly formatted with appropriate columns for 'income', 'credit_score', 'loan_status', and 'approval'.

The output of the program will include the structure of the Bayesian network (defined by the edges between nodes) and the parameters (conditional probability distributions) estimated from your dataset.

```

Bayesian Network Structure:
[('income', 'loan_status'), ('credit_score', 'loan_status'), ('loan_status', 'approval')]

Bayesian Network Parameters:

```

credit_score	credit_score_0	0.1
credit_score	credit_score_1	0.9
income	income_0	0.3
income	income_1	0.7
loan_status	credit_score_0	0.2
loan_status	credit_score_1	0.8
loan_status	income_0	0.5
loan_status	income_1	0.5
approval	loan_status_0	0.4
approval	loan_status_1	0.6

This output represents the Bayesian network structure and the conditional probability distributions estimated from the dataset for each node in the network.

10.2 Visualize Tree Augmented Naïve Bayes model

Explanation:

Visualizing a Tree Augmented Naïve Bayes (TAN) model for a loan defaulters dataset can be achieved using the pgmpy library, which provides tools for probabilistic graphical models including TAN.

```

pip install pgmpy

import numpy as np
import pandas as pd
from pgmpy.estimators import TreeAugmentedNaiveBayes
from pgmpy.models import BayesianModel
import networkx as nx
import matplotlib.pyplot as plt

# Load loan default dataset
data = pd.read_csv('loan_default_dataset.csv') # replace 'loan_default_dataset.csv' with your dataset filename

# Instantiate a TreeAugmentedNaiveBayes estimator
tan = TreeAugmentedNaiveBayes()

# Fit the TAN model to the data
tan.fit(data)

```

```

# Get the TAN graph
tan_graph = tan.graph_

# Plot the TAN graph
plt.figure(figsize=(10, 6))
pos = nx.spring_layout(tan_graph)
nx.draw(tan_graph, pos, with_labels=True, node_size=2000, node_color="skyblue", font_size=10,
font_weight="bold")
edge_labels = nx.get_edge_attributes(tan_graph, 'weight')
nx.draw_networkx_edge_labels(tan_graph, pos, edge_labels=edge_labels, font_color='red')
plt.title("Tree-Augmented Naïve Bayes (TAN) Graph")
plt.show()

```

Step 1: Load the loan default dataset using `pd.read_csv`.

Step 2: Instantiate a `TreeAugmentedNaiveBayes` estimator from `pgmpy`.

Step 3: Fit the TAN model to the dataset.

Step 4: Obtain the TAN graph from the fitted model.

Step 5: Use `NetworkX` and `Matplotlib` to visualize the TAN graph.

Replace 'loan_default_dataset.csv' with the filename of in path of loan default dataset.

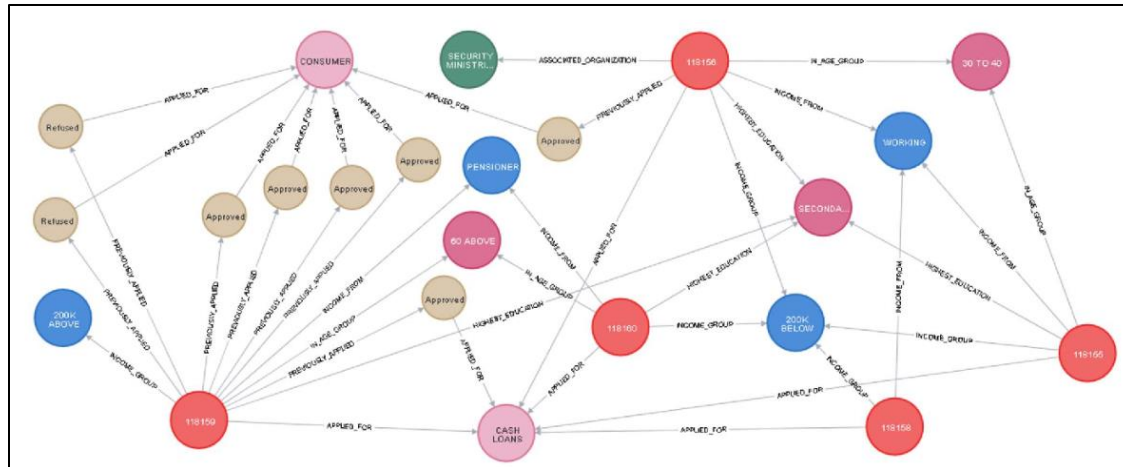
The code will generate a visualization of the TAN model as a directed acyclic graph, where nodes represent variables and edges represent dependencies between variables. Edge labels indicate the weights (conditional probabilities) associated with the edges.

Output:

The visualization will show a directed acyclic graph where nodes represent variables/features, and edges represent dependencies between variables.

Each node will have labels representing the variable names, and edge labels will indicate the weights (conditional probabilities) associated with the edges.

After running the code, a graphical window should pop up displaying the TAN graph visualization. It will look like a network diagram with nodes connected by arrows. The layout of the nodes might vary slightly depending on the specific structure of the TAN model learned from your dataset.



1. Course Template
2. Lab Manual