Artificial Intelligence Lab

Practical File

Name	Ashish Chauhan
Roll Number	05619051723
Branch	IIOT–B1-23
School	University School of Automation & Robotics
University	Guru Gobind Singh Indraprastha University

INDEX

S No.	Experiment	Sign.

Experiment – 1

Aim: For given 2 jugs with capacities 5 liters and 3 liters and an infinite water supply. Determine how the total amount of water in both the jugs may reach 4 liters.

Program:

```
x=0
y=0
print(f"Initial State: (3 Litre jug : {x} , 5 litre jug : {y} )")

while y!=4:
    if y==0 :
        #Filling of 5 litre jug
        y=5
        print(f"Fill 5 Litre jug: 3 litre jug: {x}, 5 Litre jug : {y}")

    elif x<3:
        #pouring water from 5 litre to 3 litre jug
        transfer = min (3-x , y)
        x+= transfer
        y-=transfer
        print(f"Pour from 5 litre jug to 3 litre jug : 3 litre jug : {x} , 5 litre jug : {y}")

    elif x==3:
        #Emptying of 3 litre jug
        x=0
        print(f"Empty 3 litre jug : 3 Litre jug : {x} , 5 Litre jug : {y}")

print(f"Goal reached : 3 litre jug : {x} , 5 litre jug : {y}")
```

Output:

```
Initial State: (3 Litre jug : 0 , 5 litre jug : 0 )
Fill 5 Litre jug: 3 litre jug: 0, 5 Litre jug : 5
Pour from 5 litre jug to 3 litre jug : 3 litre jug : 3 , 5 litre jug : 2
Empty 3 litre jug : 3 Litre jug : 0 , 5 Litre jug : 2
Pour from 5 litre jug to 3 litre jug : 3 litre jug : 2 , 5 litre jug : 0
Fill 5 Litre jug: 3 litre jug: 2, 5 Litre jug : 5
Pour from 5 litre jug to 3 litre jug : 3 litre jug : 3 , 5 litre jug : 4
Goal reached : 3 litre jug : 3 , 5 litre jug : 4
```

Description:

- Fill the 3-liter jug completely.
- Pour the 3 liters from the 3-liter jug into the 5-liter jug.
- Fill the 3-liter jug again completely.
- Carefully pour water from the 3-liter jug into the 5-liter jug untilthe 5-liter jug is full. Since the 5-liter jug already has 3 liters, youcan only add 2 more liters before it's full.
- You'll have exactly 1 liter of water left in the 3-liter jug.
- Empty the 5-liter jug.
- Pour the remaining 1 liter of water from the 3-liter jug into theempty 5-liter jug.
- Fill the 3-liter jug completely again.
- Pour the 3 liters from the 3-liter jug into the 5-liter jug, whichalready contains 1 liter.

- Now, the 5-liter jug contains exactly 4 liters of water.

- This method uses the 3-liter jug first and allows you to measure out 4liters accurately in the 5-liter jug.

Experiment – 2

Aim: 8- Puzzle Problem

Program:

```python
def print_matrix (matrix):
    for row in matrix:
        print(row)
Tabnine: Edit | Test | Explain | Document | Ask
def find_zero (matrix):
    for row_index, row in enumerate(matrix):
        for col_index, value in enumerate(row):
            if value ==0:
                return (row_index, col_index)
    return None
Tabnine: Edit | Test | Explain | Document | Ask
def operation (A, i, j, action):
    if action == 1 and i > 0:
        A[i-1][j], A[i][j] = A[i][j], A[i-1][j]
    elif action ==2 and i < 2:
        A[i+1][j], A[i][j] = A[i][j], A[i+1][j]
    elif action == 3 and j > 0:
        A[i][j-1], A[i][j] = A[i][j], A[i][j-1]
    elif action == 4 and j <2 :
        A[i][j+1], A[i][j] = A[i][j], A[i][j+1]
    else:
        print("Invalid Input")
    return A
A = [[1, 2, 3], [4, 0, 5], [7, 8, 6]]
G = [[1,2,3], [4,5,6],[7,8,0]]
print("Actual Matrix")
print_matrix(A)
print("Goalstate Matrix")
print_matrix(G)
while A != G:
    print_matrix(A)
    print("Choose 1.Up 2. Down 3. Left 4. Right")
    action = int(input("Choose the action you want to perform: "))
    if action in [1,2,3,4]:
        i,j = find_zero(A)
        A = operation (A, i, j, action)
    else:
        print("Invalid Attempt")
print("Matrix is Transformed")
```

Output:

```
Actual Matrix
[1, 2, 3]
[4, 0, 5]
[7, 8, 6]
Goalstate Matrix
[1, 2, 3]
[4, 5, 6]
[7, 8, 0]
[1, 2, 3]
[4, 0, 5]
[7, 8, 6]
Choose 1.Up 2. Down 3. Left 4. Right
Choose the action you want to perform: 4
[1, 2, 3]
[4, 5, 0]
[7, 8, 6]
Choose 1.Up 2. Down 3. Left 4. Right
Choose the action you want to perform: 2
Matrix is Transformed
```

Description:

- Display each row of the matrix to show its current state.
- Iterate through the matrix to locate the position of the 0 element.
- If the user chooses to move the zero up, swap the zero with the element directly above it.
- If the user chooses to move the zero down, swap the zero with the element directly below it.
- If the user chooses to move the zero left, swap the zero with the element directly to the left.
- If the user chooses to move the zero right, swap the zero with the element directly to the right.
- If the input is invalid, display an error message.
- Display the initial matrix and the goal matrix to the user.
- Continue printing the matrix and asking for user input until the current matrix matches the goal matrix.
- After each operation, display the current state of the matrix.
- Ask the user to choose an action (up, down, left, or right) to move the zero.
- Locate the zero and execute the chosen operation to update the matrix.
- If the user input is not valid, display an error message and prompt again.

- Once the matrix matches the goal state, print a success message indicating that the matrix has been transformed.

Experiment – 3

Aim – To Understand the Concept Breadth First Search

Program:

```
graph = {
  'P' : ['A','B', 'C'],
  'A' : ['D'],
  'B' : ['C'],
  'C' : ['B', 'A'],
  'D' : ['P'],
}

visited = []
queue = []

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)

  while queue:
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)
print("Following is the Breadth-First Search")
bfs(visited, graph, 'A')
```

Output

```
Following is the Breadth-First Search
A D P B C

...Program finished with exit code 0
Press ENTER to exit console.
```

Description

- The graph is defined as a dictionary where keys represent nodes and values are lists of adjacent nodes.
- Empty lists visited and queue are created to keep track of visited nodes and nodes to be explored, respectively.
- The function takes the visited list, the graph dictionary, and the starting node as input.
- The starting node is added to the visited list and the queue.
- While the queue is not empty:
- The first element m is popped from the queue.
- M is printed as a visited node.
- For each neighbor of m in the graph.
- If neighbor is not in the visited list.
- Neighbor is added to the visited list and the queue
- The BFS function is called with the initial visited list the graph, and the node ‘A’. The function prints the nodes visited in BFS order.

Experiment – 4

Aim: To understand the concept of depth search function

Program:

```
graph = {
    'A' : ['D','B'],
    'B' : ['A'],
    'C' : ['D','A','B'],
    'D' : ['C'],
}

visited = []

def dfs(graph, start, visited=None):
  if visited is None:
    visited = set()

  visited.add(start)
  print(start, end=" ")

  for neighbor in graph[start]:

    if neighbor not in visited:
      dfs(graph, neighbor, visited)
dfs(graph, 'A')
```

Output

```
A D C B

...Program finished with exit code 0
Press ENTER to exit console.
```

Description

- The graph is defined as a dictionary where keys represent nodes and values are lists of adjacent nodes.
- An empty list visited is created to keep track of visited nodes.
- The function takes the graph dictionary, the starting node, and an optional visited set as input.
- If visited is not provided, it creates a new set.
- The current node is added to the visited set.
- The current node is printed.
- For each neighbor of the current node:
- If the neighbor is not in the visited set, recursively calls the DFS function with the neighbor.
- The DFS function is called with the initial visited list, the graph, and the starting node ‘A’. The function prints the nodes visited in DFS order.

Experiment – 5

Aim: To implement graph colouring algorithm in python

Program:

```python
def graph_coloring(graph):
    colors = {}
    available_colors = set(range(len(graph)))

    for node in graph:
        neighbor_colors = set(colors.get(neighbor) for neighbor in graph[node] if neighbor in colors)
        available_colors_for_node = available_colors - neighbor_colors
        if not available_colors_for_node:
            return None
        colors[node] = min(available_colors_for_node)
    return colors

graph = {
    'A': ['B', 'C' ],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B', 'E'],
    'E': ['B', 'D'],
    'F': ['C' , 'A']
}

coloring = graph_coloring(graph)

if coloring:
    print("Graph is colorable:")
    for node, color in coloring.items():
        print(f"{node}: {color}")
else:
    print("Graph is not colorable")
```

Output:

```
Graph is colorable:
A: 0
B: 1
C: 1
D: 0
E: 2
F: 2
```

Description:

- Takes a graph as input and returns a dictionary of assigned colours or None if the graph is not colourable.
- Stores assigned colours for each node.
- Contains all available colours.
- Find the colours of adjacent nodes.
- Calculate available colours for the current node. If no colours are available, the graph is not colourable.
- Assign the minimum available colour to the node.
- If the graph is colourable, return the assigned colours. Otherwise, return None.

Experiment No: 6

Aim: To perform A* Search Algorithm in a provided path/graph

Program:

```python
import heapq
def a_star_search(graph, start, goal):
    open_set = [(0, start)]
    closed_set = set()
    came_from = {}
    g_scores = {start: 0}
    f_scores = {start: heuristic(start, goal)}  # Dictionary to store the estimated total cost (g_score + h_score) for each node

    while open_set:
        current_node = heapq.heappop(open_set)[1]
        if current_node == goal:
            return reconstruct_path(came_from, start, goal)
        closed_set.add(current_node)
        for neighbor, distance in graph[current_node].items():
            tentative_g_score = g_scores[current_node] + distance
            if neighbor in closed_set and tentative_g_score >= g_scores[neighbor]:
                continue
            if neighbor not in open_set or tentative_g_score < g_scores[neighbor]:
                came_from[neighbor] = current_node
                g_scores[neighbor] = tentative_g_score
                f_scores[neighbor] = g_scores[neighbor] + heuristic(neighbor, goal)
                heapq.heappush(open_set, (f_scores[neighbor], neighbor))
    return None
def reconstruct_path(came_from, start, goal):
    current = goal
    path = [current]
    while current != start:
        current = came_from[current]
        path.append(current)
    path.reverse()
    return path
graph = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'D': 5},
    'C': {'A': 4, 'E': 2},
    'D': {'B': 5, 'E': 1},
    'E': {'C': 2, 'D': 1}
}
start = 'A'
goal = 'E'
heuristic = lambda node, goal: abs(ord(node) - ord(goal))  # A simple heuristic based on the distance between nodes in the alphabet
path = a_star_search(graph, start, goal)
if path:
    print("Shortest path:", path)
else:
    print("No path found.")
```

Output:

```
Shortest path: ['A', 'C', 'E']
```

Description:

- Initializes open set, closed set, came from, g scores, and f scores.
- Iterates over open set until the goal is reached or open set is empty.
- Explores neighbors of the current node, updates g scores, f scores, and came from, and adds eligible neighbors to open set.
- Reconstructs the path using reconstruct path if the goal is reached.
- Traces back from the goal to the start using came from to reconstruct the path.
- Defines a sample graph, start node, goal node, and a heuristic function.
- Calls a star search to find the shortest path.
- Prints the path or a message indicating no path found.

Experiment No: 7

Aim: To perform operations on array using Numpy.

Theory: NumPy is a Python library for numerical computing that provides efficient handling of large, multi-dimensional arrays and matrices. It offers mathematical functions, linear algebra capabilities, and powerful tools for array manipulation. NumPy is essential for scientific computing, data analysis, and machine learning, serving as a foundation for many other libraries.

Program:

```
import numpy as np

a=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])

#Operations on the array
print("Array :\n" , a)
print("Slicing of 2-d array :\n" ,a[0:4:2,0:4:2])
print("The dimensions of array :" ,a.ndim)
print("The shape of array :" , a.shape)
print("The size of the array is: ", a.size)
print("The largest element in the array :", a.max())
print("The smallest element in the array :", a.min())
print("The mean of elements in the array :", a.mean())
print("The diagonal elements of the array :\n" , a.diagonal())
print("The sum of elements of the array :\n" , a.sum())
print("The data type of element entries in the array :" , a.dtype)

#New arrays
print("Array with only ones : \n" , np.ones([3,3], int))
print("Array with diagonal elements as 1 : \n" , np.eye(3,3,0,int))

#Using the given Array perform operations on it
x= np.array([[9,44,33,66],[22,41,30,43],[60,80,90,100],[10,11,12,13]])
print("Given Array : \n" ,x)
print("Slicing 2-d array : \n", x[2:4,1:3])
print("Sliced array : ", x[2:3])
print("Sliced array : ", x[2:3,1:3])
print("Sliced array : \n", x[0:3,0:1])
print("Sliced array : ", x[1:2,1:4:2])
```

Output:

```
Array :
 [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Slicing of 2-d array :
 [[ 1  3]
 [ 9 11]]
The dimensions of array : 2
The shape of array : (4, 4)
The size of the array is:  16
The largest element in the array : 16
The smallest element in the array : 1
The mean of elements in the array : 8.5
The diagonal elements of the array :
 [ 1  6 11 16]
The sum of elements of the array :
 136
The data type of element entries in the array : int32
Array with only ones :
 [[1 1 1]
 [1 1 1]
 [1 1 1]]
Array with diagonal elements as 1 :
 [[1 0 0]
 [0 1 0]
 [0 0 1]]
Given Array :
 [[  9  44  33  66]
 [ 22  41  30  43]
 [ 60  80  90 100]
 [ 10  11  12  13]]
Slicing 2-d array :
 [[80 90]
 [11 12]]
Sliced array :  [[ 60  80  90 100]]
Sliced array :  [[80 90]]
Sliced array :
 [[ 9]
 [22]
 [60]]
Sliced array :  [[41 43]]
```

Description: Numpy is a powerful Python library for numerical computing, offering a wide range of functions and tools for efficiently handling and manipulating numerical data. It's a cornerstone of many scientific computing and data analysis applications. Key features and uses of Numpy:

- Multidimensional Arrays: Numpy's primary data structure is the ndarray, which can represent arrays of arbitrary dimensions. This makes it ideal for handling matrices, vectors, and higher-dimensional data.
- Efficient Operations: Numpy performs mathematical operations on arrays much faster than Python's built-in lists, thanks to its optimized C implementation. This is crucial for large-scale numerical computations.
- Broadcasting: Numpy's broadcasting mechanism allows for automatic element-wise operations between arrays of different shapes, simplifying calculations and reducing code complexity.
- Linear Algebra: Numpy provides a rich set of functions for linear algebra operations, including matrix multiplication, inversion, eigenvalue decomposition, and more.
- Random Number Generation: Numpy's random module offers a variety of functions for generating random numbers from different distributions, essential for simulations and statistical analysis.
- Fourier Transforms: Numpy's fft module implements efficient algorithms for computing Fourier transforms, a fundamental tool in signal processing and image analysis.
- Integration with Other Libraries: Numpy seamlessly integrates with other popular Python libraries like SciPy, Matplotlib, and Pandas, making it a versatile tool for scientific computing and data analysis workflows.

Experiment No: 8

Aim: To use Pandas in python and create data frames using the same along with the use of sklearn.

Theory: Pandas is a Python library for data manipulation, offering DataFrames for efficient handling of structured data. It enables data cleaning, transformation, and analysis. Scikit-learn complements Pandas by providing machine learning algorithms and model evaluation tools, facilitating seamless workflows for data preprocessing, analysis, and model training in data science.

Code:

```
#Tanya Gupta IIOT B2
import pandas as pd
df = pd.DataFrame({"Name" : ["A", "B", "C"],
                   "Departments" : ["Eng", "Maths", "App"],
                   "Salary" : [45000,50000,75000]})
print(df)
type(df)
df.describe()
print("\n")
print("Elements at row 2: \n", df.loc[2])
print("Elements from row 0 to 1: \n", df.loc[[0,1]])
print("Elements at columns 'Name' and 'Salary': \n", df.loc[:, ["Name","Salary"]])
print("Elements at row 0 to 1 and column 0 and 2: \n", df.iloc[[0,1],[0,2]])


df = pd.read_csv(r"C:\Users\dell\Desktop\iris2\a08a1080b88344b0c8a7-0e7a9b0a5d22642a06d3d5b9bcbad9890c8ee534\iris.
print(df.describe())
print(type(df))
print(df.isna())
print(df.dropna())
map = {"species" :
          {"setosa" : 0,
           "versicolor" : 1,
           "virginica" : 2}}
df.replace(map,inplace=True)
print(df)
```

```
print(df)

x_train = df.iloc[:135,[0,1,2,3]]
x_test = df.iloc[135:150,[0,1,2,3]]
y_train = df.iloc[:135,[4]]
y_test = df.iloc[135:150,[4]]

print(x_train)
print(x_test)
print(y_train)
print(y_test)

import sklearn
from sklearn.neighbors import KNeighborsClassifier
myobj = KNeighborsClassifier()
mymodel = myobj.fit(x_train, y_train)
y= mymodel.predict(x_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y, y_test))
```

Output:

```
Run  scratch
C:\Users\dell\PycharmProjects\3rdsemproject\.venv\Scripts\python.exe C:\Users\dell\AppData\Roaming\JetBrains\PyCharmCE2024.2\scratches\scrat
  Name Departments  Salary
0    A         Eng   45000
1    B       Maths   50000
2    C         App   75000


Elements at row 2:
 Name              C
Departments      App
Salary         75000
Name: 2, dtype: object
Elements from row 0 to 1:
   Name Departments  Salary
0    A         Eng   45000
1    B       Maths   50000
Elements at columns 'Name' and 'Salary':
   Name  Salary
0    A   45000
1    B   50000
2    C   75000
Elements at row 0 to 1 and column 0 and 2:
```

```
Elements at row 0 to 1 and column 0 and 2:
   Name  Salary
0    A   45000
1    B   50000
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.054000      3.758667     1.198667
std        0.828066     0.433594      1.764420     0.763161
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
<class 'pandas.core.frame.DataFrame'>
     sepal_length  sepal_width  petal_length  petal_width  species
0           False        False         False        False    False
1           False        False         False        False    False
2           False        False         False        False    False
3           False        False         False        False    False
4           False        False         False        False    False
..            ...          ...           ...          ...      ...
```

```
..            ...          ...           ...          ...      ...
145         False        False         False        False    False
146         False        False         False        False    False
147         False        False         False        False    False
148         False        False         False        False    False
149         False        False         False        False    False

[150 rows x 5 columns]
     sepal_length  sepal_width  petal_length  petal_width    species
0             5.1          3.5           1.4          0.2     setosa
1             4.9          3.0           1.4          0.2     setosa
2             4.7          3.2           1.3          0.2     setosa
3             4.6          3.1           1.5          0.2     setosa
4             5.0          3.6           1.4          0.2     setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

[150 rows x 5 columns]
```

```
     sepal_length  sepal_width  petal_length  petal_width  species
0             5.1          3.5           1.4          0.2        0
1             4.9          3.0           1.4          0.2        0
2             4.7          3.2           1.3          0.2        0
3             4.6          3.1           1.5          0.2        0
4             5.0          3.6           1.4          0.2        0
..            ...          ...           ...          ...      ...
145           6.7          3.0           5.2          2.3        2
146           6.3          2.5           5.0          1.9        2
147           6.5          3.0           5.2          2.0        2
148           6.2          3.4           5.4          2.3        2
149           5.9          3.0           5.1          1.8        2

[150 rows x 5 columns]
     sepal_length  sepal_width  petal_length  petal_width
0             5.1          3.5           1.4          0.2
1             4.9          3.0           1.4          0.2
2             4.7          3.2           1.3          0.2
3             4.6          3.1           1.5          0.2
4             5.0          3.6           1.4          0.2
..            ...          ...           ...          ...
130           7.4          2.8           6.1          1.9
```

```
130           7.4          2.8           6.1          1.9
131           7.9          3.8           6.4          2.0
132           6.4          2.8           5.6          2.2
133           6.3          2.8           5.1          1.5
134           6.1          2.6           5.6          1.4

[135 rows x 4 columns]
     sepal_length  sepal_width  petal_length  petal_width
135           7.7          3.0           6.1          2.3
136           6.3          3.4           5.6          2.4
137           6.4          3.1           5.5          1.8
138           6.0          3.0           4.8          1.8
139           6.9          3.1           5.4          2.1
140           6.7          3.1           5.6          2.4
141           6.9          3.1           5.1          2.3
142           5.8          2.7           5.1          1.9
143           6.8          3.2           5.9          2.3
144           6.7          3.3           5.7          2.5
145           6.7          3.0           5.2          2.3
146           6.3          2.5           5.0          1.9
147           6.5          3.0           5.2          2.0
148           6.2          3.4           5.4          2.3
```

```
148           6.2          3.4           5.4          2.3
149           5.9          3.0           5.1          1.8
     species
0          0
1          0
2          0
3          0
4          0
..       ...
130        2
131        2
132        2
133        2
134        2

[135 rows x 1 columns]
     species
135        2
136        2
137        2
138        2
139        2
140        2
```

```
     species
135        2
136        2
137        2
138        2
139        2
140        2
141        2
142        2
143        2
144        2
145        2
146        2
147        2
148        2
149        2
C:\Users\dell\PycharmProjects\3rdsemproject\.venv\Lib\site-packages\sklearn\neighbors\_classification.py:238: DataConversionWarning: A column-v
  return self._fit(X, y)
1.0

Process finished with exit code 0
```

Description: Pandas is a powerful Python library for data manipulation and analysis. It offers:

1. **Data Structures**: The two main structures are Series (1D) and Data Frame (2D), which allow you to work with labelled data like Excel tables or SQL databases.

2. **Data Handling**: Pandas simplifies tasks like importing / exporting data, cleaning missing values, filtering rows / columns, and transforming datasets.

3. **Aggregation and Grouping**: It allows you to easily group data by columns and apply functions like sum, mean, or count to analyse large datasets.

4. **Merging and Time-Series**: Pandas supports merging/joining multiple datasets and has excellent tools for working with time-series data.

Ques)Given a classification dataset containing over 15,000 rows and at least 5 columns, the following tasks need to be performed: First, display the dataset's keys, shape, size, mean, and standard deviation. Then, split the dataset into features and target variables, followed by applying train-test splits with the ratios of 60:40, 70:30, and 80:20. If necessary, perform label encoding. Next, apply the K-Nearest Neighbors (KNN) classifier to the dataset and evaluate the model's performance using metrics such as accuracy, confusion matrix, and classification report.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Step 1: Load the Dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv"
column_names = ['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
                'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
                'pH', 'sulphates', 'alcohol', 'quality']
data = pd.read_csv(url, sep=';', names=column_names, header=0)

# Step 2: Display Data Shape, Size, Mean, Standard Deviation
print("Data Shape:", data.shape)
print("Data Size:", data.size)
print("Mean:\n", data.mean())
print("Standard Deviation:\n", data.std())

# Step 3: Dividing the dataset into features and target
features = data.drop('quality', axis=1)
target = data['quality']

# Step 4: Train-Test Split and Performance Analysis
splits = [(0.6, 0.4), (0.7, 0.3), (0.8, 0.2)]
results = {}

for train_size, test_size in splits:
    X_train, X_test, y_train, y_test = train_test_split(features, target, train_size=train_size, random_state=42)
    # Step 6: KNN Classifier
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train, y_train)

    # Step 7: Performance Analysis
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    conf_matrix = confusion_matrix(y_test, y_pred)
    class_report = classification_report(y_test, y_pred)

    results[train_size] = {
        'accuracy': accuracy,
        'confusion_matrix': conf_matrix,
        'classification_report': class_report
    }

# Print results
for train_size, metrics in results.items():
    print(f"\nTrain Size: {train_size*100:.0f}%")
    print("Accuracy:", metrics['accuracy'])
    print("Confusion Matrix:\n", metrics['confusion_matrix'])
    print("Classification Report:\n", metrics['classification_report'])

# Optional: Visualizing the Confusion Matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(target), yticklabels=np.unique(target))
plt.xlabel('Predicted Quality')
plt.ylabel('True Quality')
plt.title('Confusion Matrix')
plt.show()
```

OUTPUT

```
Data Shape: (1599, 12)
Data Size: 19188
Mean:
 fixed_acidity            8.319637
volatile_acidity         0.527821
citric_acid              0.270976
residual_sugar           2.538806
chlorides                0.087467
free_sulfur_dioxide     15.874922
total_sulfur_dioxide    46.467792
density                  0.996747
pH                       3.311113
sulphates                0.658149
alcohol                 10.422983
quality                  5.636023
dtype: float64
Standard Deviation:
 fixed_acidity            1.741096
volatile_acidity         0.179060
citric_acid              0.194801
residual_sugar           1.409928
chlorides                0.047065
free_sulfur_dioxide     10.460157
total_sulfur_dioxide    32.895324
density                  0.001887
pH                       0.154386
sulphates                0.169507
alcohol                  1.065668
quality                  0.807569
dtype: float64
```

```
Train Size: 60%
Accuracy: 0.4921875
Confusion Matrix:
 [[  0   0   2   0   0   0]
 [  0   1  10  11   1   0]
 [  0   1 166 105   6   0]
 [  0   1 102 134  11   0]
 [  0   0  19  48  14   0]
 [  0   0   2   6   0   0]]
Classification Report:
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.33      0.04      0.08        23
           5       0.55      0.60      0.57       278
           6       0.44      0.54      0.49       248
           7       0.44      0.17      0.25        81
           8       0.00      0.00      0.00         8

    accuracy                           0.49       640
   macro avg       0.29      0.23      0.23       640
weighted avg       0.48      0.49      0.47       640


Train Size: 70%
Accuracy: 0.48541666666666666
Confusion Matrix:
 [[  0   0   1   0   0   0]
 [  0   1   7   9   0   0]
 [  0   1 126  64   4   0]
 [  0   2  89  97  12   0]
 [  0   0  18  34   9   0]
 [  0   0   1   4   1   0]]
```

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

           3       0.00      0.00      0.00         1
           4       0.25      0.06      0.10        17
           5       0.52      0.65      0.58       195
           6       0.47      0.48      0.48       200
           7       0.35      0.15      0.21        61
           8       0.00      0.00      0.00         6

    accuracy                           0.49       480
   macro avg       0.26      0.22      0.23       480
weighted avg       0.46      0.49      0.46       480


Train Size: 80%
Accuracy: 0.45625
Confusion Matrix:
 [[ 0  0  1  0  0  0]
 [ 0  0  5  5  0  0]
 [ 0  0 82 44  4  0]
 [ 0  2 65 59  6  0]
 [ 0  0 14 22  5  1]
 [ 0  0  1  3  1  0]]
Classification Report:
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         1
           4       0.00      0.00      0.00        10
           5       0.49      0.63      0.55       130
           6       0.44      0.45      0.45       132
           7       0.31      0.12      0.17        42
           8       0.00      0.00      0.00         5

    accuracy                           0.46       320
   macro avg       0.21      0.20      0.19       320
weighted avg       0.42      0.46      0.43       320
```