Name: Kuldeep Gheghate

PRN: 123B1B118

ASSIGNMENT NO. 2(A)

Title: Write a Python program to create a dataframe by importing CSV, JSON files and perform various operations.

Aim:
To create a Python program that imports data from CSV and JSON files into a Pandas DataFrame and performs tasks such as viewing, filtering, sorting, grouping, and modifying the data.

Topic Theory:

Pandas:
Pandas is a powerful Python library used for handling and analyzing data. It simplifies working with data organized in tables, much like spreadsheets in Excel or Google Sheets. With Pandas, we can:

- Read data from different file formats such as CSV, Excel, and JSON.
- Clean and structure data for better analysis.
- Perform operations like filtering, sorting, and grouping.
- Merge and combine multiple datasets.

Data Structures in Pandas:

1. **Series:**
    - A Series is a one-dimensional data structure, similar to a list.
    - Each value in a Series has an index number for easy reference.
2. **DataFrame:**
    - A DataFrame is a two-dimensional data structure that organizes data into rows and columns.

- Each column in the DataFrame is a Series, which allows handling multiple data fields together.

Syntax Overview:

1. **Importing Pandas:**
   We start by importing the Pandas library, which provides functions to handle data efficiently.
2. **Importing CSV Files:**
   The read_csv() function is used to import data from CSV files into a DataFrame.
3. **Importing JSON Files:**
   The read_json() function is used to import data from JSON files into a DataFrame.

Algorithm:

1. **Step 1:** Import the Pandas library to handle data.
2. **Step 2:** Read data from a CSV file and load it into a DataFrame.
3. **Step 3:** Display the entire dataset.
4. **Step 4:** View specific parts of the data, like the first few rows or the last few rows.
5. **Step 5:** Get detailed information about the dataset, such as the number of rows, columns, data types, and memory usage.
6. **Step 6:** Access individual columns or rows to extract specific data or calculate values like the maximum in a column or generate summary statistics.
7. **Step 7:** Stop and conclude the program.

Conclusion:

This assignment demonstrates how to use Pandas for managing and analyzing data from CSV and JSON files. It shows how to read, inspect, and perform various operations on data within a DataFrame.

CODE:

```
import pandas as pd

# Step 1: Import the Pandas library

# Step 2: Read data from a CSV file and load it into a DataFrame
csv_data = pd.read_csv('data.csv')

# Step 3: Display the entire dataset
print("CSV Data:")
print(csv_data)

# Step 4: View specific parts of the data
print("\nFirst 5 rows of CSV Data:")
print(csv_data.head())
print("\nLast 5 rows of CSV Data:")
print(csv_data.tail())

# Step 5: Get detailed information about the dataset
print("\nCSV Data Information:")
print(csv_data.info())
print("\nCSV Data Summary Statistics:")
print(csv_data.describe())

# Step 6: Access individual columns or rows
print("\nAccessing individual columns:")
```

```
print(csv_data['column_name'])  # Replace 'column_name' with the actual column name

print("\nAccessing individual rows:")

print(csv_data.loc[0])  # Access the first row


# Step 7: Read data from a JSON file and load it into a DataFrame

json_data = pd.read_json('data.json')


# Display the JSON data

print("\nJSON Data:")

print(json_data)


# Perform operations on the JSON data

print("\nJSON Data Information:")

print(json_data.info())

print("\nJSON Data Summary Statistics:")

print(json_data.describe())
```

json Verify

```
[
  {"Name": "John", "Age": 25, "Country": "USA"},
  {"Name": "Alice", "Age": 30, "Country": "UK"},
  {"Name": "Bob", "Age": 35, "Country": "Australia"},
  {"Name": "Eve", "Age": 20, "Country": "Canada"}
]
```

csv

```
Name,Age,Country
John,25,USA
Alice,30,UK
Bob,35,Australia
Eve,20,Canada
```

OUTPUT:

```
CSV Data:
      Name  Age    Country
0     John   25        USA
1    Alice   30         UK
2      Bob   35  Australia
3      Eve   20     Canada

First 5 rows of CSV Data:
      Name  Age    Country
0     John   25        USA
1    Alice   30         UK
2      Bob   35  Australia
3      Eve   20     Canada

Last 5 rows of CSV Data:
      Name  Age    Country
0     John   25        USA
1    Alice   30         UK
2      Bob   35  Australia
3      Eve   20     Canada

CSV Data Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    4 non-null      object
 1   Age     4 non-null      int64
 2   Country  4 non-null      object
dtypes: int64(1), object(2)
memory usage: 176.0+ bytes

CSV Data Summary Statistics:
             Age
count   4.000000
mean   27.500000
std     5.916079
min    20.000000
25%    25.000000
50%    27.500000
75%    30.000000
max    35.000000

Accessing individual columns:
0      John
1     Alice
2       Bob
3       Eve
Name: Name, dtype: object

Accessing individual rows:
Name      John
Age         25
Country    USA
Name: 0, dtype: object

JSON Data:
      Name  Age    Country
0     John   25        USA
1    Alice   30         UK
2      Bob   35  Australia
3      Eve   20     Canada

JSON Data Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    4 non-null      object
 1   Age     4 non-null      int64
 2   Country  4 non-null      object
dtypes: int64(1), object(2)
memory usage: 176.0+ bytes

JSON Data Summary Statistics:

```

ASSIGNMENT NO. 2(B)

Title: Perform web scraping for any website (using scrapy/beautiful soap/selenium).

Aim:
The goal of this task is to learn how to extract data from websites using tools like Scrapy, Beautiful Soup, or Selenium. This includes understanding web page structure and retrieving relevant information for analysis.

Topic Theory:

Web Scraping:
Web scraping in Python involves automatically extracting data from websites. The process includes accessing a webpage, parsing its HTML structure, and extracting the necessary data.

Methods of Web Scraping:

1. **Requests:**
   This method is used to send HTTP requests to load web pages and retrieve content for further processing.
2. **Beautiful Soup:**
   A library that allows you to find and extract specific information from a webpage by parsing its HTML.

Parser in Beautiful Soup:
In BeautifulSoup, a parser processes HTML documents and converts them into a structured format, making it easier to extract and manipulate data.

Prettify Function:
The prettify() function in BeautifulSoup formats HTML code in a clear and readable manner by adding proper indentation and line breaks.

Algorithm:

1. **Step 1:** Set up the environment by loading the necessary libraries.
2. **Step 2:** Send an HTTP request to fetch the webpage's HTML content.
3. **Step 3:** Parse the HTML content into a structured format using BeautifulSoup.

4. **Step 4:** Search for and extract specific HTML elements using methods like find() or find_all().
5. **Step 5:** Stop.

Conclusion:

In this assignment, we explored how to use the Requests and BeautifulSoup libraries to fetch, parse, and extract data from a webpage. We also learned how to identify specific HTML elements.

CODE:

```
# Import necessary libraries
import pandas as pd # Used for data manipulation and analysis
from bs4 import BeautifulSoup # Used for web scraping and parsing HTML and XML
import requests # Used for making HTTP requests

# Use requests.get() to fetch the webpage
html_doc = requests.get('http://www.pccoepune.com') # Fetch the webpage content
print(html_doc.text) # Print the webpage content

soup = BeautifulSoup(html_doc.content, 'html.parser') # Parse the webpage content using BeautifulSoup

dataFrame=[] # Initialize an empty list to store dataframes
for i,table in enumerate(soup.find_all('table')): # find all tables in the parsed HTML
  rows=table.findAll('tr')[1:] # Find all rows in each table except the first one

  data=[] # Initialize an empty list to store data for each table
  for row in rows: # Iterate over each row
    cols=row.findAll('td') # Find all columns in each row
    col=[col.text.strip() for col in cols] # Extract text from each column and remove leading/trailing spaces

    data.append(cols) # Append the extracted data to the data list
    df=pd.DataFrame(data) # Create a pandas DataFrame from the data
    dataFrame.append(df) # Append the DataFrame to the dataFrame list
```

ASSIGNMENT NO. 2(c)

Title: Write a Python program to import any dataset from UCI/Kaggle, perform data cleaning and remove the outliers.

Aim:

The purpose of this assignment is to demonstrate how to import a dataset from UCI or Kaggle, clean the data by handling missing or incorrect values, and remove outliers to ensure that the data is ready for analysis.

Topic Theory:

1. Data Cleaning:
Data cleaning is the process of preparing raw data for analysis by fixing or removing incorrect, incomplete, or irrelevant parts of the dataset. This is an important step to ensure the accuracy and consistency of your analysis. Data cleaning tasks include:

- Handling missing data
- Fixing incorrect data
- Removing duplicate entries

2. Outliers:
Outliers are data points that differ significantly from other observations in the dataset. They can distort statistical analyses, so it is important to identify and handle them properly. Common techniques to detect and remove outliers include:

- Z-score method: Identifies outliers based on how far they are from the mean.
- IQR (Interquartile Range) method: Outliers are detected based on their distance from the first and third quartiles of the dataset.

Steps to Solve the Problem:

1. Import Dataset from UCI/Kaggle:
You can download the dataset directly from UCI or Kaggle and load it into a Pandas DataFrame using Python. Pandas is a library that helps you work with structured data.

2. Data Cleaning:
This includes tasks like:

- Removing rows with missing values or filling missing values with appropriate data (like the mean, median, or mode).
- Removing duplicate rows if they exist.
- Checking for and fixing incorrect data (e.g., incorrect data types or out-of-range values).

3. Detecting and Removing Outliers:
After the dataset is clean, the next step is to detect and handle outliers. The most common techniques for identifying outliers are:

- Using Z-scores: Identifies outliers by checking how far a value is from the mean.
- Using IQR: Identifies outliers that are significantly above or below the typical range of the data.

Algorithm:

1. **Step 1:** Import the required libraries such as Pandas, NumPy, and Matplotlib (or Seaborn for visualization).
2. **Step 2:** Load the dataset into a Pandas DataFrame from a CSV file.
3. **Step 3:** Perform data cleaning:
    - Remove or fill missing data.
    - Remove duplicate rows.
    - Fix incorrect data if necessary.
4. **Step 4:** Identify outliers using one of the common methods (Z-score or IQR).
5. **Step 5:** Remove or handle the outliers by either dropping them or transforming the data.
6. **Step 6:** Finalize the cleaned dataset and display the results.
7. **Step 7:** Stop.

Conclusion:

In this assignment, we learned how to import a dataset from UCI or Kaggle, clean the data by handling missing values and duplicates, and identify and remove outliers. These steps are essential for ensuring the dataset is ready for accurate analysis.

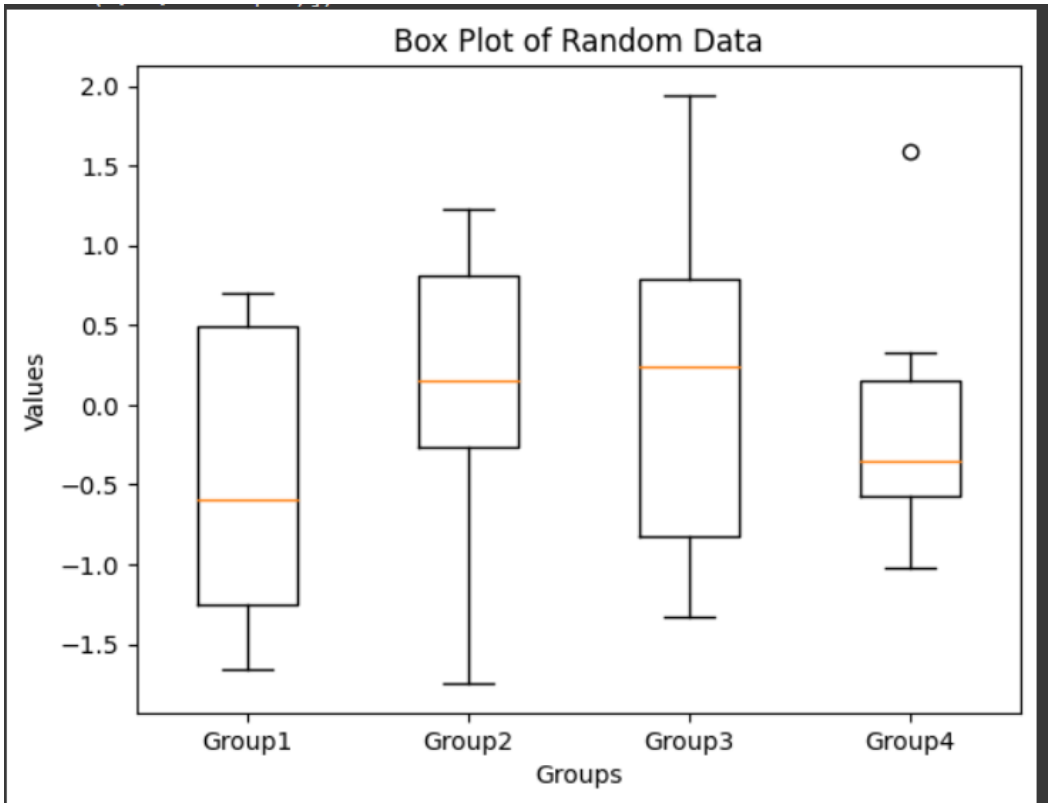
Code:

```
# prompt: import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
import numpy as np

# sample data
data = np.random.randn(10,4)
print(data)

#creat a box plot
plt.boxplot(data)
plt.title("Box Plot of Random Data")
plt.xlabel("Groups")
plt.ylabel("Values")
plt.xticks([1,2,3,4],['Group1','Group2','Group3','Group4'])
```

```
[[ 0.53727033 -0.83744552  0.31555713  1.58484375]
 [ 0.68334103  0.02475353 -0.80405714 -0.56455188]
 [-1.37300029 -1.74498022 -0.83382824  0.33011702]
 [-0.91104676  1.2249387   0.17446947 -0.102746  ]
 [-1.66148437  0.21624014  0.90777377 -0.37975559]
 [ 0.36908745  0.84420334 -0.89742469 -0.31802508]
 [-1.36968973  0.71669893  1.1937     -0.57498538]
 [-0.87428496  0.08246183  1.9414313  -1.01841184]
 [ 0.70392461 -0.3664973  -1.32887018 -0.97738673]
 [-0.3160289   1.06353127  0.43574829  0.23939072]]
([<matplotlib.axis.XTick at 0x7bd191758c10>,
  <matplotlib.axis.XTick at 0x7bd191758be0>,
  <matplotlib.axis.XTick at 0x7bd191784700>,
  <matplotlib.axis.XTick at 0x7bd1917b9870>],
 [Text(1, 0, 'Group1'),
  Text(2, 0, 'Group2'),
  Text(3, 0, 'Group3'),
  Text(4, 0, 'Group4')])
```




```python
import pandas as pd
import numpy as np

# Generate random names
names = ['Liam', 'Noah', 'Oliver', 'William', 'Elijah', 'James', 'Benjamin', 'Lucas', 'Mason', 'Ethan','Kuldeep','kunal','Gargi','kush','Natasha']
# Generate random heights (in cm)
heights = np.random.randint(70, 150++, size=15)

# Create a DataFrame
df = pd.DataFrame({'Name': names, 'Height (cm)': heights})

print(df)

# box plot
print(df.boxplot(column=['Height (cm)']))

# Quantile
Q1 = df['Height (cm)'].quantile(0.25)
Q2 = df['Height (cm)'].quantile(0.50)
Q3 = df['Height (cm)'].quantile(0.75)
print(Q1,Q2,Q3)
IQR = Q3 - Q1
print(IQR)
lower_limit = Q1 - 1.5 * IQR
upper_limit = Q3 + 1.5 * IQR
print(lower_limit,upper_limit)
```

```
         Name  Height (cm)
0        Liam          124
1        Noah          129
2      Oliver          137
3     William          117
4      Elijah          134
5       James           86
6    Benjamin          126
7       Lucas          132
8       Mason          136
9       Ethan          124
10    Kuldeep          116
11      kunal           92
12      Gargi           72
13       kush          109
14    Natasha           93
Axes(0.125,0.11;0.775x0.77)
101.0 124.0 130.5
29.5
56.75 174.75
```