Student: Balakrishna Mupparaju

Assignment: Week5 & 6

Activity 5.01: Reading Tabular Data from a Web Page and Creating DataFrames

In [4]:

```
# Import necessary libraries
from bs4 import BeautifulSoup  # To parse HTML files
import pandas as pd  # For managing tabular data and creating DataFrames

# Open and read the local HTML file containing GDP data
fd = open("/Users/balakrishnamupparaju/Downloads/The-Data-Wrangling-Workshop
soup = BeautifulSoup(fd)  # Parse the file's contents using BeautifulSoup
fd.close()  # Close the file to release system resources

# Find all tables within the HTML document
all_tables = soup.find_all("table")
print("Total number of tables are {} ".format(len(all_tables)))  # Print tot
```

```
Total number of tables are 9
```

In [16]:

```
# Select a specific table based on its class
data_table = soup.find("table", {"class": '"wikitable"|}'})
print("Type of the selected table is: {}".format(type(data_table))) # Confir

# Extract the first row (likely header-like or meta-data row) of the table
sources = data_table.tbody.findAll('tr', recursive=False)[0]

# Extract all <td> elements (columns) from the first row
sources_list = [td for td in sources.findAll('td')]
print("Number of source columns found: {}".format(len(sources_list)))  # Pri

# Extract the second row (likely the first row of data) in the table
data = data_table.tbody.findAll('tr', recursive=False)[1].findAll('td', recu

# Initialize an empty list to store nested tables found within the main tabl
data_tables = []
for td in data:
    data_tables.append(td.findAll('table'))  # Find any nested tables in eac
print("Number of nested tables found: {}".format(len(data_tables)))  # Print

# Extract source names from the header row by finding anchor tags (<a>)
source_names = [source.findAll('a')[0].getText() for source in sources_list]
print("Source names extracted: {}".format(source_names))  # Print the source
```

```
Type of the selected table is: <class 'bs4.element.Tag'>
Number of source columns found: 3
Number of nested tables found: 3
Source names extracted: ['International Monetary Fund', 'World Bank', 'Unite
d Nations']
```

In [17]:

```
# ----- FIRST TABLE -----
# Extract headers (column names) from the first nested table
header1 = [th.getText().strip() for th in data_tables[0][0].findAll('thead')
print("Headers for the first table: {}".format(header1))  # Print the header

# Extract all rows of data from the first nested table
rows1 = data_tables[0][0].findAll('tbody')[0].findAll('tr')
data_rows1 = [[td.get_text().strip() for td in tr.findAll('td')] for tr in r

# Create a DataFrame for the first table
df1 = pd.DataFrame(data_rows1, columns=header1)
print("First DataFrame preview:")
print(df1.head())  # Display the first few rows of the DataFrame
```

```
Headers for the first table: ['Rank', 'Country', 'GDP(US$MM)']
First DataFrame preview:
  Rank        Country  GDP(US$MM)
0           World[19]  79,865,481
1    1  United States  19,390,600
2    2      China[n 1]  12,014,610
3    3          Japan   4,872,135
4    4        Germany   3,684,816
```

In [18]:

```
# ----- SECOND TABLE -----
# Extract headers from the second nested table
header2 = [th.getText().strip() for th in data_tables[1][0].findAll('thead')
print("Headers for the second table: {}".format(header2))  # Print the heade

# Extract all rows of data from the second nested table
rows2 = data_tables[1][0].findAll('tbody')[0].findAll('tr')

# Define a helper function to clean and extract the desired text
def find_right_text(i, td):
    if i == 0:  # First column logic
        return td.getText().strip()
    elif i == 1:  # Second column logic
        return td.getText().strip()
    else:  # For other columns, extract text after the ♠ symbol
        index = td.text.find("♠")
        return td.text[index+1:].strip()

# Apply the helper function to process all data rows
data_rows2 = [[find_right_text(i, td) for i, td in enumerate(tr.findAll('td'

# Create a DataFrame for the second table
df2 = pd.DataFrame(data_rows2, columns=header2)
print("Second DataFrame preview:")
print(df2.head())  # Display the first few rows of the DataFrame
```

```
Headers for the second table: ['Rank', 'Country', 'GDP(US$MM)']
Second DataFrame preview:
  Rank             Country  GDP(US$MM)
0                    World  80,683,787
1    1       United States  19,390,604
2       European Union[23]  17,277,698
3    2            China[n 4]  12,237,700
4    3               Japan   4,872,137
```

In [19]:

```
# ------ THIRD TABLE ------
# Extract headers from the third nested table
header3 = [th.getText().strip() for th in data_tables[2][0].findAll('thead')
print("Headers for the third table: {}".format(header3))  # Print the header

# Extract all rows of data from the third nested table
rows3 = data_tables[2][0].findAll('tbody')[0].findAll('tr')

# Apply the helper function to process all data rows
data_rows3 = [[find_right_text(i, td) for i, td in enumerate(tr.findAll('td'

# Create a DataFrame for the third table
df3 = pd.DataFrame(data_rows3, columns=header3)
print("Third DataFrame preview:")
print(df3.head())  # Display the first few rows of the DataFrame
```

```
Headers for the third table: ['Rank', 'Country', 'GDP(US$MM)']
Third DataFrame preview:
  Rank        Country  GDP(US$MM)
0            World[24]  75,648,448
1    1  United States  18,624,475
2    2      China[n 4]  11,218,281
3    3          Japan   4,936,211
4    4        Germany   3,477,796
```

Activity 6.01: Handling Outliers and Missing Data

In this activity, we will identify and get rid of outliers. Here, we have a CSV file. The goal here is to clean the data by using the knowledge that we have learned about so far and come up with a nicely formatted DataFrame. Identify the type of outliers and their effect on the data and clean the messy data.

In [28]:

```
# Import required libraries
import pandas as pd  # For working with tabular data
import numpy as np  # For numerical operations
import matplotlib.pyplot as plt  # For visualizing data

# Enable inline plotting for visualizing graphs directly in notebooks
%matplotlib inline

# Step 1: Load the dataset from the given CSV file
df = pd.read_csv("/Users/balakrishnamupparaju/Downloads/The-Data-Wrangling-W
print("Loaded dataset preview:")
print(df.head())  # Display the first 5 rows of the dataset
```

```
Loaded dataset preview:
   id first_name last_name                       email gender  \
0   1      Sonny      Dahl            sdahl0@mysql.com   Male   
1   2        NaN       NaN           dhoovart1@hud.gov    NaN   
2   3        Gar     Armal      garmal2@technorati.com    NaN   
3   4    Chiarra     Nulty       cnulty3@newyorker.com    NaN   
4   5        NaN       NaN  sleaver4@elegantthemes.com    NaN   

        ip_address   visit  
0    135.36.96.183  1225.0  
1  237.165.194.143   919.0  
2   166.43.137.224   271.0  
3   139.98.137.108  1002.0  
4    46.117.117.27  2434.0  
```

In [29]:

```
# Step 2: Check for duplicate entries in important columns
print("First name has duplicates: {}".format(any(df.first_name.duplicated())
print("Last name has duplicates: {}".format(any(df.last_name.duplicated())))
print("Email has duplicates: {}".format(any(df.email.duplicated())))
```

```
First name has duplicates: True
Last name has duplicates: True
Email has duplicates: False
```

In [30]:

```
# Step 3: Check for missing (NaN) values in critical columns
print("Email column contains missing values: {}".format(df.email.isnull().va
print("IP Address column contains missing values: {}".format(df.ip_address.i
print("Visit column contains missing values: {}".format(df.visit.isnull().va
```

```
Email column contains missing values: False
IP Address column contains missing values: False
Visit column contains missing values: True
```

In [32]:

```
# Step 4: Record the size of the original dataset for comparison
size_prev = df.shape  # Get the shape of the DataFrame (rows, columns)


# Step 5: Remove rows where the 'visit' column has NaN or non-numeric values
df = df[np.isfinite(df['visit'])]  # Filter rows with finite numeric values
size_after = df.shape  # Get the new shape after cleaning
```

In [33]:

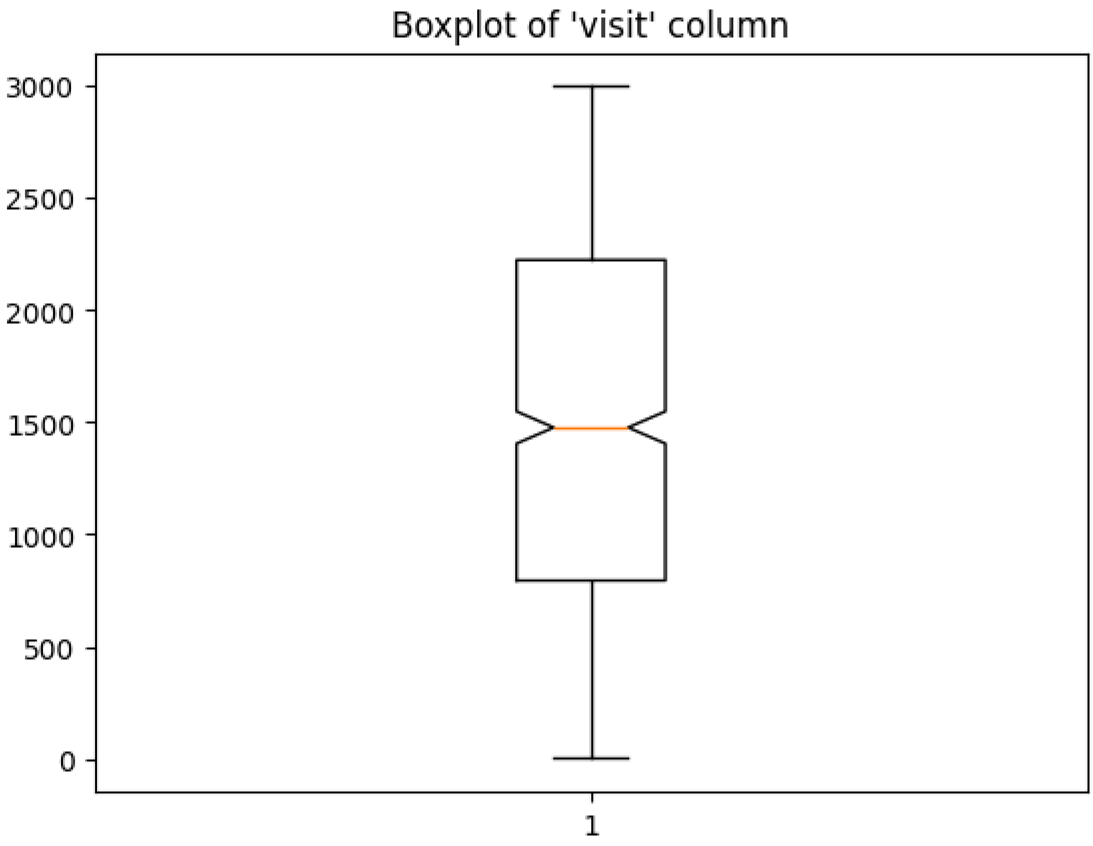
```
# Step 6: Print size comparison of the DataFrame before and after cleaning
print("Original data size: {} rows.".format(size_prev[0]))
print("Data size after removing NaN/invalid 'visit' values: {} rows.".format
```

```
Original data size: 1000 rows.
Data size after removing NaN/invalid 'visit' values: 974 rows.
```

In [34]:

```
# Step 7: Visualize the 'visit' column to identify potential outliers
plt.boxplot(df.visit, notch=True)
plt.title("Boxplot of 'visit' column")
plt.show()

#Boxplot Visualization: Helps in identifying the distribution and spotting o
```



```
In [35]: # Step 8: Remove outliers based on 'visit' value thresholds
         # Keep rows where 'visit' values are within a reasonable range (100 to 2900)
         df1 = df[(df['visit'] <= 2900) & (df['visit'] >= 100)]
         print("Data size after removing outliers: {} rows.".format(df1.shape[0]))
```

```
Data size after removing outliers: 923 rows.
```

```
In [36]: # Final output preview
         print("Cleaned dataset preview:")
         print(df1.head())
```

```
Cleaned dataset preview:
   id first_name last_name                        email gender  \
0   1      Sonny      Dahl             sdahl0@mysql.com   Male   
1   2        NaN       NaN            dhoovart1@hud.gov    NaN   
2   3        Gar     Armal       garmal2@technorati.com    NaN   
3   4    Chiarra     Nulty        cnulty3@newyorker.com    NaN   
4   5        NaN       NaN  sleaver4@elegantthemes.com    NaN   

        ip_address   visit  
0    135.36.96.183  1225.0  
1  237.165.194.143   919.0  
2   166.43.137.224   271.0  
3   139.98.137.108  1002.0  
4    46.117.117.27  2434.0  
```

```
In [ ]: 
```

3. Insert data into a SQL Lite database

In [2]:

```
# contacts_sqlite.py

import sqlite3
import pandas as pd

# ---------------------------
# Step 1: Connect to SQLite (in-memory or file-based)
# ---------------------------
conn = sqlite3.connect("contacts.db")  # Saves as contacts.db file
cursor = conn.cursor()

# ---------------------------
# Step 2: Create the contacts table
# ---------------------------
cursor.execute("""
CREATE TABLE IF NOT EXISTS contacts (
    name TEXT,
    address TEXT,
    city TEXT,
    state TEXT,
    zip TEXT,
    phone_number TEXT
)
""")

# ---------------------------
# Step 3: Insert sample data (10 rows)
# ---------------------------
sample_data = [
    ("Alice Johnson", "123 Maple St", "Springfield", "IL", "62704", "217-555
    ("Bob Smith", "456 Oak St", "Madison", "WI", "53703", "608-555-5678"),
    ("Carla Lopez", "789 Pine Ave", "Denver", "CO", "80203", "303-555-9012")
    ("David Lee", "101 Elm Dr", "Austin", "TX", "73301", "512-555-3456"),
    ("Emma Patel", "202 Birch Blvd", "Raleigh", "NC", "27601", "919-555-7890
    ("Frank Miller", "303 Cedar Ln", "Boise", "ID", "83702", "208-555-1122")
    ("Grace Kim", "404 Aspen Way", "Seattle", "WA", "98101", "206-555-3344")
    ("Henry Young", "505 Willow Rd", "Phoenix", "AZ", "85001", "602-555-5566
    ("Isabel Chen", "606 Sycamore Ct", "Boston", "MA", "02108", "617-555-778
    ("Jake Nguyen", "707 Magnolia Pl", "Portland", "OR", "97201", "503-555-9
]

cursor.executemany("INSERT INTO contacts VALUES (?, ?, ?, ?, ?, ?)", sample_
conn.commit()

# ---------------------------
# Step 4: Query the table and display as DataFrame
# ---------------------------
df = pd.read_sql_query("SELECT * FROM contacts", conn)
print("\n All Contacts:")
print(df)
```

```python
# Optional: Close the connection
conn.close()
```

```
All Contacts:
             name          address         city state    zip phone_number
0   Alice Johnson     123 Maple St  Springfield    IL  62704  217-555-1234
1       Bob Smith       456 Oak St      Madison    WI  53703  608-555-5678
2     Carla Lopez     789 Pine Ave       Denver    CO  80203  303-555-9012
3       David Lee       101 Elm Dr       Austin    TX  73301  512-555-3456
4      Emma Patel   202 Birch Blvd      Raleigh    NC  27601  919-555-7890
5    Frank Miller     303 Cedar Ln        Boise    ID  83702  208-555-1122
6       Grace Kim     404 Aspen Way      Seattle    WA  98101  206-555-3344
7     Henry Young    505 Willow Rd      Phoenix    AZ  85001  602-555-5566
8     Isabel Chen  606 Sycamore Ct       Boston    MA  02108  617-555-7788
9     Jake Nguyen  707 Magnolia Pl     Portland    OR  97201  503-555-9900
10  Alice Johnson     123 Maple St  Springfield    IL  62704  217-555-1234
11      Bob Smith       456 Oak St      Madison    WI  53703  608-555-5678
12    Carla Lopez     789 Pine Ave       Denver    CO  80203  303-555-9012
13      David Lee       101 Elm Dr       Austin    TX  73301  512-555-3456
14     Emma Patel   202 Birch Blvd      Raleigh    NC  27601  919-555-7890
15   Frank Miller     303 Cedar Ln        Boise    ID  83702  208-555-1122
16      Grace Kim     404 Aspen Way      Seattle    WA  98101  206-555-3344
17    Henry Young    505 Willow Rd      Phoenix    AZ  85001  602-555-5566
18    Isabel Chen  606 Sycamore Ct       Boston    MA  02108  617-555-7788
19    Jake Nguyen  707 Magnolia Pl     Portland    OR  97201  503-555-9900
```

```
In [ ]:
```