Bilenkin540Weeks_5_&_6_Exercises

April 16, 2025

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

[56]:
```
# Importing necessary libraries
import requests
from bs4 import BeautifulSoup
import pandas as pd
```

1. Open the page in a separate Chrome/Firefox tab and use something like an Inspect Element tool to view the source HTML and understand its structure.

Manually inspected HTML page and identified header row using Inspect Element tool. Identified the following:

First column name Country/Territory The other related columns are IMF, World Bank, and United Nations, each with two subcolumns for Year and Estimate.

2. Read the page using bs4.

[57]:
```
import requests
from bs4 import BeautifulSoup

# URL of the Wikipedia page
url = 'https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(nominal)'

# Fetching the page content
response = requests.get(url)

# Parsing the HTML content with BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')
```

3. Find the table structure you will need to deal with (how many tables are there?).

[58]:
```
import pandas as pd

url = "https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(nominal)"
tables = pd.read_html(url)

# Printing the total number of tables found
print(f"Total number of tables found: {len(tables)}")
```

```
Total number of tables found: 7
```

To determine how many tables are on the Wikipedia page, I used both soup.find_all("table") and pandas.read_html() to detect the tables. While the book's solution mentions finding 9 tables, I found only 7 tables using read_html(), which extracts only well-structured tabular data. This discrepancy is most likely due to changes that have occurred since the book was published. If I'm not mistaken, the book was released in 2020, and we are now in 2025. It's likely that the Wikipedia page has been updated over the past five years, or that each method interprets HTML content differently. As of today, the first table (index 0) contains the relevant GDP information from the IMF, World Bank, and United Nations.

4. Find the right table using bs4.

[59]:
```
# Finding all tables with the 'wikitable' class using bs4
tables = soup.find_all('table', {'class': 'wikitable'})

# Checking the number of tables to ensure we have the right one
print(f"Number of tables found: {len(tables)}")

# Accessing the first table in the list
table = tables[0]

# Inspecting the table's caption to make sure it's the GDP table
caption = table.find('caption')
if caption:
    print(f"Table Caption: {caption.text.strip()}")
else:
    print("No caption found for the table.")
```

```
Number of tables found: 1
Table Caption: GDP forecast or estimate (million US$) by country
```

5. Separate the source names and their corresponding data.

[60]:
```
# Extracting table rows (excluding header)
rows = table.find_all('tr')[1:]  # Skip the header row

# Creating list to store the cleaned data
data = []

# Extracting the data for each row (country and GDP values for IMF, World Bank,␣
 ↪United Nations)
for row in rows:
    cols = row.find_all('td')
    if len(cols) >= 6:  # Ensuring there are enough columns (Country, IMF,␣
 ↪World Bank, United Nations)
        country = cols[0].text.strip()  # Extracting country name
        imf = cols[1].text.strip()  # Extracting IMF GDP value
        wb = cols[3].text.strip()  # Extracting World Bank GDP value
        un = cols[5].text.strip()  # Extracting United Nations GDP value
```

```
        # Cleaning up footnotes and commas in the GDP data
        imf = imf.split(' ')[0].replace(',', '')  # Removing footnote and commas
        wb = wb.split(' ')[0].replace(',', '')  # Removing footnote and commas
        un = un.split(' ')[0].replace(',', '')  # Removing footnote and commas

        # Appending the cleaned data to the list
        data.append([country, imf, wb, un])

# Displaying the first few rows of the cleaned data
print(data[:5])  # Showing the first 5 rows of extracted data
```

```
[['World', '115494312', '105435540', '100834796'], ['United States', '30338000',
'27360935', '25744100'], ['China', '19535000', '17794782', '17963170'],
['Germany', '4922000', '4456081', '4076923'], ['Japan', '4390000', '4212945',
'4232173']]
```

6. Get the source names from the list of sources you have created.

[61]:
```
# Source names (already identified in the headers)
sources = ['IMF', 'World Bank', 'United Nations']

# Displaying the source names
print(sources)
```

```
['IMF', 'World Bank', 'United Nations']
```

7. Separate the header and data from the data that you separated before for the first source only, and then create a DataFrame using that.

[62]:
```
# Extracting the data for IMF (first source)
imf_data = [row[1] for row in data]  # Extracting the IMF column data

# Createing a DataFrame for the IMF data
df_imf = pd.DataFrame(imf_data, columns=['IMF'])

# Displaying the IMF DataFrame
print(df_imf.head())
```

```
         IMF
0  115494312
1   30338000
2   19535000
3    4922000
4    4390000
```

8. Repeating the last task for the other two data sources.

[63]:
```
# Creating an empty list to hold unique data
unique_data = []
seen_countries = set()
```

```
# Looping through the rows and adding data if the country hasn't been seen
for idx, row in enumerate(rows[1:]):  # Skipping header row
    cols = row.find_all('td')
    if len(cols) >= 6:
        country = cols[0].text.strip()
        if country not in seen_countries:
            seen_countries.add(country)
            imf = cols[1].text.strip()
            wb = cols[3].text.strip()
            un = cols[5].text.strip()
            unique_data.append([idx + 1, country, imf, wb, un])  # Adding Rank, Country, and GDP data

# Creating the DataFrame from unique data
df_unique = pd.DataFrame(unique_data, columns=['Rank', 'Country', 'IMF', 'World Bank', 'United Nations'])

# Renaming the IMF column to GDP(US$MM) and removing the other columns (World Bank and United Nations)
df_unique = df_unique[['Rank', 'Country', 'IMF']]  # Keeping only Rank, Country, and IMF columns
df_unique = df_unique.rename(columns={'IMF': 'GDP(US$MM)'})  # Renaming IMF to GDP(US$MM)

# Displaying the first 5 rows of the updated DataFrame
print(df_unique.head())
```

```
   Rank        Country   GDP(US$MM)
0     1          World  115,494,312
1     2  United States   30,338,000
2     3          China   19,535,000
3     4        Germany    4,922,000
4     5          Japan    4,390,000
```

Activity 6.01: Handling Outliers and Missing Data

1. Read the visit_data.csv file.

[64]:
```
import pandas as pd

# Reading the visit_data.csv file
file_path = r'C:\Users\maxim\OneDrive\Desktop\BU\DSC 540\visit_data.csv'
df = pd.read_csv(file_path)

# Display the first five rows and basic info to undestand the dataset structure
print(df.head())
print(df.info())
```

```
   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
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   id          1000 non-null   int64
 1   first_name  704 non-null    object
 2   last_name   704 non-null    object
 3   email       1000 non-null   object
 4   gender      495 non-null    object
 5   ip_address  1000 non-null   object
 6   visit       974 non-null    float64
dtypes: float64(1), int64(1), object(5)
memory usage: 54.8+ KB
None
```

2. Check for duplicates.

[65]:
```
# Checking for duplicates
duplicate_rows = df[df.duplicated()]
print(f"Number of duplicate rows: {duplicate_rows.shape[0]}")
```

```
Number of duplicate rows: 0
```

3. Check whether any essential column contains NaN.

[66]:
```
# Checking for missing values in important columns
missing_data_values = df[['first_name', 'last_name', 'gender', 'visit']].
  ↪isnull().sum()
print("Missing values in essential columns:\n", missing_data_values)
```

```
Missing values in essential columns:
 first_name    296
last_name     296
gender        505
visit          26
dtype: int64
```

4. Get rid of the outliers.

[67]:
```
import numpy as np

# Cleaning the data
df_cleaned = df.dropna(subset=['visit'])  # dropping rows where 'visit' is NaN

# Getting rid of outliers using the IQR method for the 'visit' column
Q1 = df_cleaned['visit'].quantile(0.25)
Q3 = df_cleaned['visit'].quantile(0.75)
IQR = Q3 - Q1

# Defining the lower and upper bounds for the data
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filtering out the outliers
df_no_outliers = df_cleaned[(df_cleaned['visit'] >= lower_bound) &␣
 ↪(df_cleaned['visit'] <= upper_bound)]

# Displaying result
print(f"Data size after removing outliers: {df_no_outliers.shape}")
```

```
Data size after removing outliers: (974, 7)
```

5. Report the size difference.

[68]:
```
# Original data size
original_size = df.shape

# Cleaned data size after removing outliers
cleaned_size = df_no_outliers.shape

# Calculating the difference in size
size_difference = original_size[0] - cleaned_size[0]

# Printing the results
print(f"Original data size: {original_size}")
print(f"Cleaned data size: {cleaned_size}")
print(f"Number of rows removed (outliers): {size_difference}")
```

```
Original data size: (1000, 7)
Cleaned data size: (974, 7)
Number of rows removed (outliers): 26
```

6. Create a box plot to check for outliers.

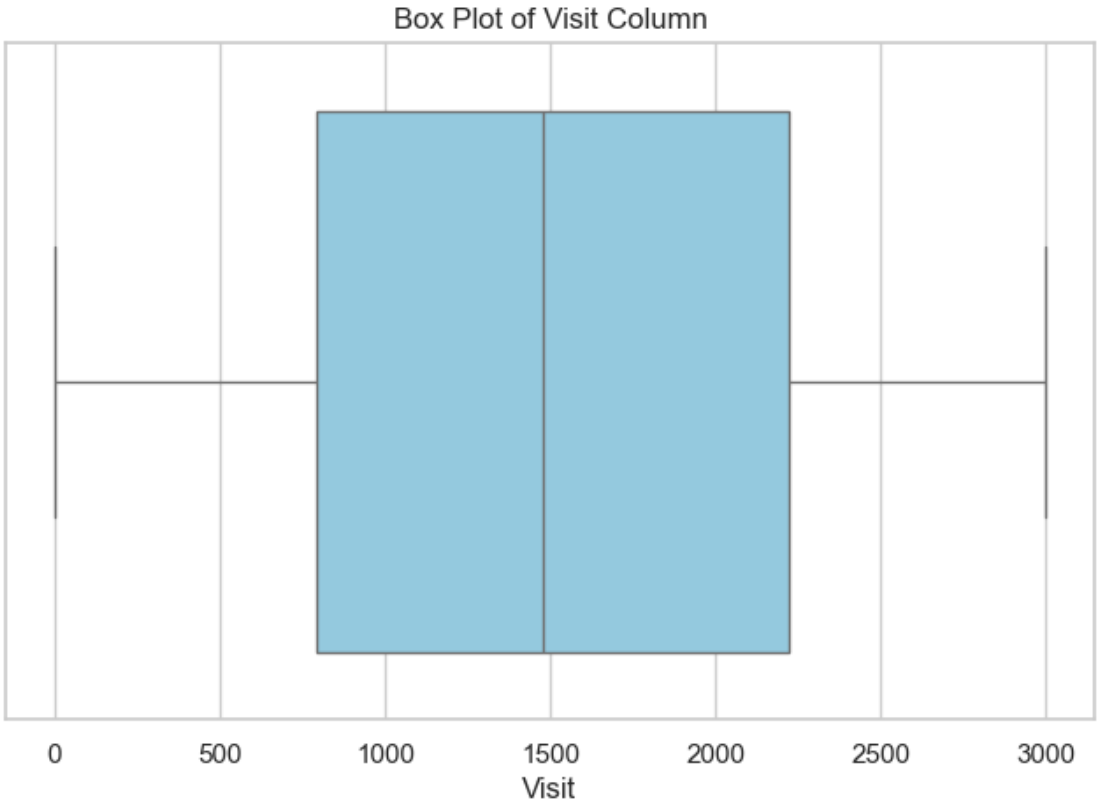
[69]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# Setting plot style
sns.set(style="whitegrid")

# Creating a box plot for the 'visit' column
plt.figure(figsize=(8, 5))
sns.boxplot(x=df['visit'], color='skyblue')
plt.title('Box Plot of Visit Column')
plt.xlabel('Visit')
plt.show()

# Calculation and printing Mean and Median (formatted to two decimal places)
mean_visit = df['visit'].mean()
median_visit = df['visit'].median()

print(f"Mean: {mean_visit:.2f}")
print(f"Median: {median_visit:.2f}")
```




```
Mean: 1497.98
Median: 1477.00
```

As we can see from the box plot above, there are still some outliers present. Most of the data

is concentrated between approximately 700 and 2,300, giving us an interquartile range (IQR) of about 1,600. The distribution appears to be slightly positively skewed, as the mean (1497.98) is marginally higher than the median (1477.00). This suggests a few outliers on the right side (higher end) of the data distribution.

7. Get rid of any outliers.

[70]:
```
df_cleaned = df[(df['visit'] >= 100) & (df['visit'] <= 2900)]
print("Final row count after removing outliers:", len(df_cleaned))
```

```
Final row count after removing outliers: 923
```

The original dataset had 1,000 rows. After removing missing values, 26 rows were dropped, leaving 974 rows. From the box plot and the calculation of the mean and median, we observed that the distribution still contained some outliers, as the mean was slightly higher than the median. By removing the outliers—values below the lower bound of 100 and above the upper bound of 2,900—we ended up with 923 rows.

3. Insert data into a SQL Lite database – create a table with the following data below:

a. Name, Address, City, State, Zip, Phone Number

[71]:
```
import sqlite3

# Connecting to (or create) the database
conn = sqlite3.connect("contacts.db")
cursor = conn.cursor()

# Creating the table with specified columns
cursor.execute('''
CREATE TABLE IF NOT EXISTS contacts (
    name TEXT,
    address TEXT,
    city TEXT,
    state TEXT,
    zip TEXT,
    phone_number TEXT
)
''')

# Committing the changes to the database
conn.commit()
```

b. Add at least 10 rows of data and submit your code with a query generating your results.

[72]:
```
import pandas as pd

# Sample data to insert
data = [
```

```
    ("Alice Johnson", "123 Main St", "Springfield", "IL", "62701",␣
 ↪"217-555-1234"),
    ("Bob Smith", "456 Elm St", "Chicago", "IL", "60601", "312-555-5678"),
    ("Carol White", "789 Oak St", "Peoria", "IL", "61602", "309-555-9012"),
    ("David Lee", "321 Pine St", "Naperville", "IL", "60540", "630-555-3456"),
    ("Eva Green", "654 Cedar St", "Rockford", "IL", "61101", "815-555-7890"),
    ("Frank Black", "987 Maple St", "Aurora", "IL", "60505", "630-555-2345"),
    ("Grace Hall", "159 Birch St", "Joliet", "IL", "60431", "815-555-6789"),
    ("Henry Young", "753 Walnut St", "Evanston", "IL", "60201", "847-555-1122"),
    ("Isla Moore", "852 Cherry St", "Elgin", "IL", "60120", "847-555-3344"),
    ("Jake King", "951 Poplar St", "Waukegan", "IL", "60085", "847-555-5566")
]

# Inserting data into the table
cursor.executemany('INSERT INTO contacts VALUES (?, ?, ?, ?, ?, ?)', data)
conn.commit()

# Querying and displaying the results
df = pd.read_sql_query("SELECT * FROM contacts", conn)
print(df)
```

```
              name         address          city state    zip  phone_number
0    Alice Johnson     123 Main St   Springfield    IL  62701  217-555-1234
1        Bob Smith      456 Elm St       Chicago    IL  60601  312-555-5678
2      Carol White      789 Oak St        Peoria    IL  61602  309-555-9012
3        David Lee     321 Pine St    Naperville    IL  60540  630-555-3456
4        Eva Green    654 Cedar St      Rockford    IL  61101  815-555-7890
..             …            …            …   …    …          …
235    Frank Black    987 Maple St        Aurora    IL  60505  630-555-2345
236     Grace Hall    159 Birch St        Joliet    IL  60431  815-555-6789
237    Henry Young   753 Walnut St      Evanston    IL  60201  847-555-1122
238     Isla Moore   852 Cherry St         Elgin    IL  60120  847-555-3344
239      Jake King   951 Poplar St      Waukegan    IL  60085  847-555-5566

[240 rows x 6 columns]
```