Chattapadhyay_DSC540_FinalProject_Milestone3

DSC 540 Week 7-8 - Milestone 3

Kausik Chattapadhyay

Milestone 3

Perform at least 5 data transformation and/or cleansing steps to your website data. For example:

- Replace Headers
- Format data into a more readable format
- Identify outliers and bad data
- Find duplicates
- Fix casing or inconsistent values
- Conduct Fuzzy Matching

Dataset

HTML - I will be scrapping data from https://www.worldometers.info/coronavirus/#countries to get the covid details for all countries.

In [1]:
```
# Load the necessary libraries.

#import libraries
import os, sys
import json
import pandas as pd
import numpy as np
from numpy import int64
import requests, io
import urllib.request
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt

# Basic plotting packages
import matplotlib.pyplot as plt
# Advanced plotting
import seaborn as sns

# Interactive visualization
import plotly.express as px

import plotly.graph_objs as go
# import plotly.figure_factory as ff
from plotly.subplots import make_subplots
```

In [5]:
```
# Reading the website data

url = "https://www.worldometers.info/coronavirus/#countries"
response = requests.get(url)

class HTMLTableParser:

    def parse_url(self, url):
        response = requests.get(url)
        soup = BeautifulSoup(response.text, 'lxml')
        return [(table['id'],self.parse_html_table(table)) for table in soup.find_all('table')]

    # HTML parser method to clean messy data
    def parse_html_table(self, table):
        n_columns = 0
        n_rows=0
        column_names = []
        # Find number of rows and columns
        # we also find the column titles if we can
        for row in table.find_all('tr'):
            # Determine the number of rows in the table
            td_tags = row.find_all('td')
            if len(td_tags) > 0:
                n_rows+=1
                if n_columns == 0:
                    # Set the number of columns for our table
                    n_columns = len(td_tags)

            # Handle column names if we find them
            th_tags = row.find_all('th')
            if len(th_tags) > 0 and len(column_names) == 0:
                for th in th_tags:
                    colData = th.get_text()
                    colData = colData.replace('/','').replace(' ','').replace('\n','').replace(' ','').replace(' ','')
                    column_names.append(colData)

        # Safeguard on Column Titles

        if len(column_names) > 0 and len(column_names) != n_columns:
            raise Exception("Column titles do not match the number of columns")

        columns = column_names if len(column_names) > 0 else range(0,n_columns)
        df = pd.DataFrame(columns = columns, index= range(0,n_rows))

        row_marker = 0

        for row in table.find_all('tr'):
            column_marker = 0
            columns = row.find_all('td')
            for column in columns:
                df.iat[row_marker,column_marker] = column.get_text()
                column_marker += 1
            if len(columns) > 0:
                row_marker += 1

        # Convert to float if possible

        for col in df:
            try:
                df[col] = df[col].astype(float)
            except ValueError:
                pass

        return df
```

In [6]:
```
# Parsing Html data

hp = HTMLTableParser()
table = hp.parse_url(url)[0][1]
# Grabbing the table from the tuple
table.head(10)
```

Out[6]:

	#	CountryOther	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	SeriousCritical	...	TotalTests	Tests1Mpop	Population	Continent	1Caseevery
0		\nNorth America\n	123,329,027		1,596,552		118,266,265		3,466,210	8,730	...				North America	
1		\nAsia\n	212,704,671	+109,988	1,528,929	+322	197,200,461	+90,358	13,975,281	15,996	...				Asia	
2		\nEurope\n	244,739,871	+8,796	2,005,299	+75	240,413,530	+45,796	2,321,042	6,542	...				Europe	
3		\nSouth America\n	67,686,626		1,347,141		65,810,786	+854	528,699	10,233	...				South America	
4		\nOceania\n	13,888,348	+25	25,232		13,718,857	+48	144,259	77	...				Australia/Oceania	
5		\nAfrica\n	12,773,223		258,499		12,058,924		455,800	547	...				Africa	
6		\n\n	721		15		706		0	0	...					
7		World	675,122,487	+118,809	6,761,667	+397	647,469,529	+136,811	20,891,291	42,125	...				All	
8	1	USA	104,183,562		1,132,719		101,290,878		1,759,965	3,564	...	1,159,772,464	3,464,021	334,805,269	North America	
9	2	India	44,682,784		530,740		44,150,289		1,755	698	...	915,265,788	650,679	1,406,631,776	Asia	

10 rows × 22 columns

In [7]:
```
#We can see a few special characters ("\n", "+") to remove in the table. Let's check the dataframe.

#check bottom rows

table.tail(10)
```

Out[7]:

	#	CountryOther	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	SeriousCritical	...	TotalTests	Tests1Mpop	Population	Continent	1Case
237	230	Tokelau	5						5		...			1,378	Australia/Oceania	
238	231	China	503,302		5,272		379,053		118,977	7,557	...	160,000,000	110,461	1,448,471,400	Asia	
239		Total:	123,329,027		1,596,552		118,266,265		3,466,210	8,730	...				North America	
240		Total:	212,704,671	+109,988	1,528,929	+322	197,200,461	+90,358	13,975,281	15,996	...				Asia	
241		Total:	244,739,871	+8,796	2,005,299	+75	240,413,530	+45,796	2,321,042	6,542	...				Europe	
242		Total:	67,686,626		1,347,141		65,810,786		528,699	10,233	...				South America	
243		Total:	13,888,348	+25	25,232		13,718,857		144,259	77	...				Australia/Oceania	
244		Total:	12,773,223		258,499		12,058,924		455,800	547	...				Africa	
245		Total:	721		15		706		0	0	...					
246		Total:	675,122,487	+118,809	6,761,667	+397	647,469,529	+136,811	20,891,291	42,125	...				All	

10 rows × 22 columns

In [8]:
```
# There are some extra special characters (\n..\n) in the dataframe.
# We need to remove the extra characters. We only need country data for mapping in this tutorial.
# So we can drop the extra top and bottom rows that we do not need for data processing.
# Drop top bottom unwanted rows

df= table.drop(table.index[[0,1,2,3,4,5,6,7]]).reset_index(drop=True)

#drop tail unwanted rows

df.drop(df.tail(8).index,inplace=True)

#drop new line \n' charachter

df.replace(('\n'), '', regex=True, inplace=True)
df.replace((','), '', regex=True, inplace=True)
```

In [9]:
```
#We need to format the table before starting mapping.

# The special characters in the dataframe can be removed using a loop as below:
# drop unwanted drop unwanted special characters using a loop

for col in df.columns[0:20]:
    df[col]=df[col].str.replace('+', '').str.replace(',', '').str.replace('N/A', '').str.replace(' ', '').str.replace(' ', '').str.replace(' ', '')
```

```
/var/folders/bt/ccg5q32ys4199ny7_d3rd5gm0000gn/T/ipykernel_49964/3463145445.py:7: FutureWarning: The default value of regex will change from True to False in a future v
ersion. In addition, single character regular expressions will *not* be treated as literal strings when regex=True.
  df[col]=df[col].str.replace('+', '').str.replace(',', '').str.replace('N/A', '').str.replace(' ', '').str.replace(' ', '').str.replace(' ', '')
```

In [10]:
```
# All the extracted data is in text format and some column names are improper for data processing.
# We need to rename some column names.

df1 = df.rename(columns={'CountryOther': 'Country_Name', 'SeriousCritical': 'Serious_Critical', 'Tot CasesIMpop': 'Tot_Cases_1M_pop', 'DeathsIMpop': 'Deaths_1M_pop', 'T

df1.head()
```

Out[10]:

	#	Country_Name	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious_Critical	...	TotalTests	Tests_1M_pop	Population	Continent	1CaseeveryX
0	1	USA	104183562		1132719		101290878		1759965	3564	...	1159772464	3464021	334805269	NorthAmerica	
1	2	India	44682784		530740		44150289		1755	698	...	915265788	650679	1406631776	Asia	
2	3	France	39511451		164176		39256968		96307	869	...	271490168	4139547	65584519	Europe	
3	4	Germany	37758773		165563		37379500	24700	213710	1281	...	122332384	1458359	83883596	Europe	
4	5	Brazil	36809608		696892		35890413		222303	8318	...	63776166	296146	215353593	SouthAmerica	

5 rows × 22 columns

In [11]:
```
# However, it is still not enough for data processing. We need to check the data type of each data frame colums.

df1.dtypes
```

Out[11]:
```
#                    object
Country_Name         object
TotalCases           object
NewCases             object
TotalDeaths          object
NewDeaths            object
TotalRecovered       object
NewRecovered         object
ActiveCases          object
Serious_Critical     object
Tot CasesIMpop       object
Deaths_1M_pop        object
TotalTests           object
Tests_1M_pop         object
Population           object
Continent            object
1CaseeveryXpp1       object
1DeatheveryXpp1      object
1TesteveryXpp1       object
NewCasesIMpop        object
NewDeaths1Mpop       object
ActiveCasesIMpop     object
dtype: object
```

In [13]:
```
# The data type of each column is object in the dataframe.
# So we need to convert some data types to appropriate data types in the data frame.
# Type conversion is the conversion of object from one data type to another data type.
#convert object columns in dataframe to numeric
df1.fillna(0, inplace=True)
df1.replace(np.nan, 0, inplace=True)
df1.replace(np.inf, 0, inplace=True)

for col in df1.columns[0:20]:
    df1[col] = pd.to_numeric(df1[col], errors='ignore')

df1.dtypes
```

Out[13]:
```
#                    int64
Country_Name        object
TotalCases           int64
NewCases           float64
TotalDeaths        float64
NewDeaths          float64
TotalRecovered     float64
NewRecovered       float64
ActiveCases        float64
Serious_Critical   float64
Tot CasesIMpop     float64
Deaths_1M_pop      float64
TotalTests         float64
Tests_1M_pop       float64
Population         float64
Continent           object
1CaseeveryXpp1     float64
1DeatheveryXpp1    float64
1TesteveryXpp1     float64
NewCasesIMpop      float64
NewDeaths1Mpop      object
ActiveCasesIMpop    object
dtype: object
```

In [14]:
```
# replace null values with 0

worldometer_data = df1.replace('', np.nan).fillna(0)
worldometer_data.head()
```

Out[14]:

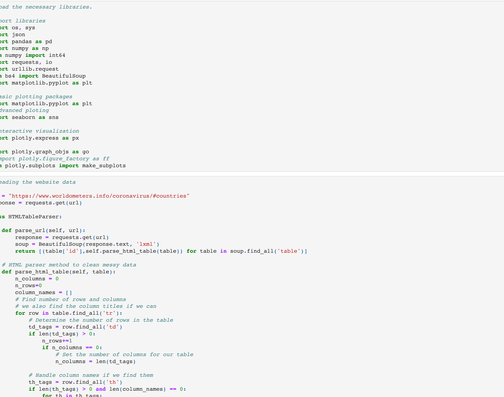
	#	Country_Name	TotalCases	NewCases	TotalDeaths	NewDeaths	TotalRecovered	NewRecovered	ActiveCases	Serious_Critical	...	TotalTests	Tests_1M_pop	Population	Continent	1Caseev
0	1	USA	104183562	0.0	1132719.0	0.0	101290878.0	0.0	1759965.0	3564.0	...	1.159772e+09	3464021.0	3.348053e+08	NorthAmerica	
1	2	India	44682784	0.0	530740.0	0.0	44150289.0	0.0	1755.0	698.0	...	9.152656e+08	650679.0	1.406632e+09	Asia	
2	3	France	39511451	0.0	164176.0	0.0	39256968.0	0.0	96307.0	869.0	...	2.714902e+08	4139547.0	6.558452e+07	Europe	
3	4	Germany	37758773	0.0	165563.0	0.0	37379500.0	24700.0	213710.0	1281.0	...	1.223324e+08	1458359.0	8.388360e+07	Europe	
4	5	Brazil	36809608	0.0	696892.0	0.0	35890413.0	0.0	222303.0	8318.0	...	6.377617e+07	296146.0	2.153536e+08	SouthAmerica	

5 rows × 22 columns

In [15]:
```
def plot_bhar_wm(col, n, min_pop=1000000, sort='descending'):
    df = worldometer_data[worldometer_data['Population']>min_pop]
    df = df.sort_values(col, ascending=True).tail(n)
    fig = px.bar(df,
                 x=col, y="Country_Name", color='Continent',
                 text=col, orientation='h', width=700,
                 color_discrete_sequence = px.colors.qualitative.Dark2)

    fig.update_layout(title=col+ '(Only countries with > 1M Pop)', xaxis_title="", yaxis_title="",
                      yaxis_categoryorder = 'total ascending',
                      uniformtext_minsize=8, uniformtext_mode='hide')
    fig.show()
```

In [16]:
```
plot_bhar_wm('Tests_1M_pop', 15, 1000000)
```



In [ ]:
```

```