Coronavirus World Data Analysis

KATE expects your code to define variables with specific names that correspond to certain things we are interested in.

KATE will run your notebook from top to bottom and check the latest value of those variables, so make sure you don't overwrite them.

- Remember to uncomment the line assigning the variable to your answer and don't change the variable or function names.
- Use copies of the original or previous DataFrames to make sure you do not overwrite them by mistake.

You will find instructions below about how to define each variable.

Once you're happy with your code, upload your notebook to KATE to check your feedback.

First of all, run the following cell to:

- import `pandas` with an alias of `pd`
- read a CSV containing the data to work with
- convert the `date` column to the `datetime` format
- create a DataFrame `df` containing the data for only **1st July 2020**
- take a look at the first few rows of the DataFrame

In [1]:
```
import pandas as pd

data = pd.read_csv('data/owid-covid-data.csv')
data['date'] = pd.to_datetime(data['date'])
df = data[data['date'] == '2020-07-01']

df.head()
```

Out[1]:

	iso_code	continent	location	date	total_cases	new_cases	total_deaths	new_death
173	AFG	Asia	Afghanistan	2020-07-01	31517.0	279.0	746.0	13.
300	ALB	Europe	Albania	2020-07-01	2535.0	69.0	62.0	4.
491	DZA	Africa	Algeria	2020-07-01	13907.0	336.0	912.0	7.
613	AND	Europe	Andorra	2020-07-01	855.0	0.0	52.0	0.
727	AGO	Africa	Angola	2020-07-01	284.0	8.0	13.0	2.

5 rows × 34 columns

- `df` DataFrame now has one row of data for each country with data present for **July 1st 2020**

- however, it also has a row with a `location` of `World` which contains aggregated values for all countries
- `df.tail()`, `df.info()` and `df.shape` will allow for further exploration of the structure of the DataFrame

In [2]:
```
df.tail()
```

Out[2]:

	iso_code	continent	location	date	total_cases	new_cases	total_deaths	new_de
29411	ESH	Africa	Western Sahara	2020-07-01	380.0	172.0	1.0	
29506	YEM	Asia	Yemen	2020-07-01	1158.0	30.0	312.0	
29623	ZMB	Africa	Zambia	2020-07-01	1594.0	26.0	24.0	
29738	ZWE	Africa	Zimbabwe	2020-07-01	591.0	17.0	7.0	
29934	OWID_WRL	NaN	World	2020-07-01	10465987.0	192563.0	511041.0	57

5 rows × 34 columns

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 211 entries, 173 to 29934
Data columns (total 34 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   iso_code                         211 non-null    object
 1   continent                        210 non-null    object
 2   location                         211 non-null    object
 3   date                             211 non-null    datetime64[ns]
 4   total_cases                      210 non-null    float64
 5   new_cases                        210 non-null    float64
 6   total_deaths                     210 non-null    float64
 7   new_deaths                       210 non-null    float64
 8   total_cases_per_million          210 non-null    float64
 9   new_cases_per_million            210 non-null    float64
 10  total_deaths_per_million         210 non-null    float64
 11  new_deaths_per_million           210 non-null    float64
 12  total_tests                      73 non-null     float64
 13  new_tests                        73 non-null     float64
 14  total_tests_per_thousand         73 non-null     float64
 15  new_tests_per_thousand           73 non-null     float64
 16  new_tests_smoothed               83 non-null     float64
 17  new_tests_smoothed_per_thousand  83 non-null     float64
 18  tests_units                      85 non-null     object
 19  stringency_index                 155 non-null    float64
 20  population                       211 non-null    float64
 21  population_density               200 non-null    float64
 22  median_age                       187 non-null    float64
 23  aged_65_older                    184 non-null    float64
 24  aged_70_older                    186 non-null    float64
 25  gdp_per_capita                   184 non-null    float64
 26  extreme_poverty                  122 non-null    float64
 27  cvd_death_rate                   186 non-null    float64
 28  diabetes_prevalence              194 non-null    float64
 29  female_smokers                   141 non-null    float64
 30  male_smokers                     139 non-null    float64
 31  handwashing_facilities           92 non-null     float64
 32  hospital_beds_per_thousand       165 non-null    float64
 33  life_expectancy                  208 non-null    float64
dtypes: datetime64[ns](1), float64(29), object(4)
memory usage: 57.7+ KB
```

In [4]:
```
df.shape
```

Out[4]:
```
(211, 34)
```

Q1. Create a new DataFrame called `countries` which is the same as `df` but with the `World` row removed.

- Use the `.copy()` method to ensure you have a distinct DataFrame in memory
- Assign this new DataFrame to the variable `countries` ; do not modify `df`

See below code syntax for some guidance:

```
countries['location'] != 'World'
```

In [5]:

```
#add your code below
countries = df.copy()
#mask_1 = countries['location'] == 'World'
#countries = countries[mask_1]
#countries
countries.drop(29934, axis=0, inplace=True)
#countries.tail()
#countries
```

Q2. Check the shape of your DataFrame to confirm that `countries` has one row fewer than `df` :

Please note you have been provided with the code for this question to carry out the necessary analysis. Simply uncomment the line of code and run the code cell to produce the desired results.

In [6]:

```
print(df.shape, countries.shape)
```

```
(211, 34) (210, 34)
```

Q3. Define a DataFrame based on the `countries` DataFrame, but which only contains the columns in `cols` (defined below) and assign this to a variable called `countries_dr`

- Order this DataFrame by `'total_deaths_per_million'` , with the highest numbers at the top.

See below code syntax for some guidance:

```
DataFrame_name[column_names].sort_values(by=..., ascending=False)
```

In [7]:

```
#cols = ['continent', 'location', 'total_deaths_per_million']

#add your code below
countries_dr = countries[['continent', 'location', 'total_deaths_per_million
countries_dr
```

Out[7]:

	continent	location	total_deaths_per_million
23306	Europe	San Marino	1237.551
2917	Europe	Belgium	841.615
613	Europe	Andorra	673.008
28347	Europe	United Kingdom	644.168
25362	Europe	Spain	606.633
...	...	...	...
23111	North America	Saint Vincent and the Grenadines	0.000
23926	Africa	Seychelles	0.000
15734	Africa	Lesotho	0.000
10808	Europe	Gibraltar	0.000
12195	Asia	Hong Kong	NaN

210 rows × 3 columns

Q4. Using the `countries` DataFrame we created earlier, find the sum of `total_tests` for countries in `Africa`, assigning the result, *as an integer*, to `africa_tests`.

- Use `.sum()` method calculate the sum for `total_tests` column
- Use `.astype(int)` method or `int()` function to convert results to an integer

See below code syntax for some guidance:

```
countries['continent'] == 'Africa'
```

In [8]:

```
#add your code below
#countries
africa_mask = countries['continent'] == 'Africa'

africa_tests = countries[africa_mask]


africa_tests = africa_tests['total_tests'].sum()

africa_tests.astype(int)
```

Out[8]:

```
3445134
```

Q5. How many countries in Africa have no value recorded for the number of `total_tests` column? Assign the result to `africa_missing_test_data`.

- You may find the pandas `.isna()` method and python `len()` function useful

See below code syntax for some guidance:

```
len(DataFrame_name[column_name].isna())
```

In [9]:
```
#add your code below
africa_mask = countries['continent'] == 'Africa'
africa_missing = countries[africa_mask]
#africa_missing['total_tests']
#africa_missing_test_data = len(africa_missing['total_tests'].isna())
africa_missing_test_data = africa_missing['total_tests'].isna().sum()

(africa_missing_test_data)
```

Out[9]:
```
45
```

Q6. How many countries have a higher value for `total_tests` than the `United Kingdom` ? Assign your answer to a variable called `countries_more_tests` .

Remember to work from the `countries` DataFrame rather than `df` . You should avoid modifying any existing DataFrames.

In [10]:
```
#add your code below

countries_2 = countries.set_index('location')
uk_count = countries_2.at['United Kingdom', 'total_tests']

mask_high = countries_2['total_tests']>uk_count
countries_more_tests = len(countries_2[mask_high])
countries_more_tests
```

Out[10]:
```
3
```

Q7. Create a DataFrame called `beds_dr` which is based on the `countries` DataFrame, but contains only the columns `hospital_beds_per_thousand` and `total_deaths_per_million` .

- Your answer should only include rows where there are values present in both of these columns
- You may find the `.dropna()` method useful

See below code syntax for some guidance:

```
DataFrame_name.dropna()
```

In [11]:

```
#add your code below
#countries
countries_3 = countries[['hospital_beds_per_thousand', 'total_deaths_per_mil
beds_dr = countries_3.dropna()
beds_dr
```

Out[11]:

	hospital_beds_per_thousand	total_deaths_per_million
173	0.50	19.163
300	2.89	21.544
491	1.90	20.798
952	3.80	30.635
1081	5.00	28.919
...	...	...
29136	0.80	1.794
29332	2.60	0.000
29506	0.70	10.461
29623	2.00	1.305
29738	1.70	0.471

164 rows × 2 columns

Q8. Refer to the `beds_dr` DataFrame. What is the average `total_deaths_per_million` for entries in `beds_dr` where `hospital_beds_per_thousand` is greater than the mean?

- Save the results to a new variable called `dr_high_bed_ratio`

See below code syntax for some guidance:

```
beds_dr['hospital_beds_per_thousand'] > beds_dr['hospital_beds_per_thousand'].mean()
```

In [12]:

```
#add your code below
#beds_dr
mean_beds = beds_dr['hospital_beds_per_thousand'].mean()
#mean_beds
mask_1 = beds_dr['hospital_beds_per_thousand']>mean_beds
beds = beds_dr[mask_1]
#beds
dr_high_bed_ratio = beds['total_deaths_per_million'].mean()
dr_high_bed_ratio
```

Out[12]:

```
98.18423728813559
```

Q9. Refer to the `beds_dr` DataFrame. What is the average `total_deaths_per_million` for entries in `beds_dr` where `hospital_beds_per_thousand` is less than the mean?

- Save the results to a new variable called `dr_low_bed_ratio`

See below code syntax for some guidance:

```
beds_dr['hospital_beds_per_thousand'] < beds_dr['hospital_beds_per_thousand'].mean()
```

In [13]:

```
#add your code below
#mean_beds
mask_2 = beds_dr['hospital_beds_per_thousand']<mean_beds
beds_1 = beds_dr[mask_2]
beds_1
#beds_dr
dr_low_bed_ratio = beds_1['total_deaths_per_million'].mean()
dr_low_bed_ratio
```

Out[13]:

```
56.294057142857135
```

Q10. Refer to the `countries` DataFrame. Create a new DataFrame called `no_new_cases` which contains only rows from `countries` with zero `new_cases` .

Please note you have been provided with the code for this question to carry out the necessary analysis. Simply uncomment the lines of code and run the code cell to produce the desired results.

In [14]:

```
#add your code below
no_new_cases = countries[countries['new_cases'] == 0]
no_new_cases.head()
```

Out[14]:

	iso_code	continent	location	date	total_cases	new_cases	total_deaths	new_deaths
613	AND	Europe	Andorra	2020-07-01	855.0	0.0	52.0	0.0
836	AIA	North America	Anguilla	2020-07-01	3.0	0.0	0.0	0.0
952	ATG	North America	Antigua and Barbuda	2020-07-01	66.0	0.0	3.0	0.0
1381	ABW	North America	Aruba	2020-07-01	103.0	0.0	3.0	0.0
2080	BHS	North America	Bahamas	2020-07-01	104.0	0.0	11.0	0.0

5 rows × 34 columns

Q11. Refer to the `no_new_cases` DataFrame. Which country in `no_new_cases` DataFrame has had the highest number of `total_cases` ?

- Save the results to a new variable called `highest_no_new`

See below code syntax for some guidance:

```
no_new_cases['total_cases'] == no_new_cases['total_cases'].max()
```

In [15]:
```
#add your code below

highest_no_new1 = no_new_cases.loc[no_new_cases['total_cases'] == no_new_cas
highest_no_new = highest_no_new1.values[0]
highest_no_new
```

Out[15]:
```
'Cameroon'
```

Q12. Refer to the `countries` DataFrame. What is the sum of the `population` of all countries which have had zero `total_deaths` ?

- Assign your answer to `sum_populations_no_deaths` variable
- Your answer should be in millions, rounded to the nearest whole number, and converted to an integer

In [16]:
```
#add your code below
#countries.head()
mask_no_deaths = countries['total_deaths'] == 0
countries_2 = countries[mask_no_deaths]
countries_3 = countries_2['population'].sum()
countries_3 = countries_3.astype(int)
countries_3
sum_populations_no_deaths = countries_3/1000000
sum_populations_no_deaths = sum_populations_no_deaths.round().astype(int)
sum_populations_no_deaths
```

Out[16]:
```
192
```

Q13. Create a function called `country_metric` which accepts the following three parameters:

- a DataFrame (which can be assumed to be of a similar format to `countries`)
- a location (i.e. a string which will be found in the `location` column of the DataFrame)
- a metric (i.e. a string which will be found in any column (other than `location`) in the DataFrame)

The function should return only the value from the first row for a given `location` and `metric` . *You may find `.iloc[]` useful.*

See below code syntax for some guidance:

```
def country_metric(df, location, metric):
```

In [17]:

```
#add your code below
#countries.head()
def country_metric(df, location, metric):
    return df[df['location'] == location].iloc[0][metric]
```

Q.14 Use your function to collect the value for `Vietnam` for the metric `aged_70_older` , assigning the result to `vietnam_older_70` .

Please note you have been provided with the code for this question to carry out the necessary analysis. Simply uncomment the lines of code and run the code cell to produce the desired results.

In [18]:

```
#add your code below
vietnam_older_70 = country_metric(countries, 'Vietnam', 'aged_70_older')
vietnam_older_70
```

Out[18]:

```
4.718
```

Q.15 Create another function called `countries_average` , which accepts the following three parameters:

- a DataFrame "df" (which can be assumed to be such as `countries`)
- a list of countries "countries" (which can be assumed to all be found in the `location` column of the DataFrame)
- a string "metric" (which can be assumed to be a column (other than `location`) which will be found in the DataFrame) . For instance, this string value can be `life_expectancy` .

Note that for the test on KATE for this question to pass, you need to make sure the function accepts the three parameters in the following order: `countries_average(df, countries, metric)` . (You can call your parameters however you like as long as the type of these parameters are what was described above).

The function should return the average value for the given metric for the given list of countries.

You may find `.isin()` method useful while filtering for list of countries.

In [19]:

```
#add your code below
def countries_average(df, countries, metric):
    filtered_df = df[df['location'].isin(countries)]
    average_value = filtered_df[metric].mean()

    return average_value
```

Q16. Use your `countries_average` function to find out the average `life_expectancy` of countries in the `g7` list defined below. Assign the result to the variable `g7_avg_life_expectancy` .

Please note you have been provided with the code for this question to carry out the necessary analysis. Simply uncomment the lines of code and run the code cell to produce the desired results.

In [20]:

```
g7 = ['United States', 'Italy', 'Canada', 'Japan', 'United Kingdom', 'German
```

In [21]:

```
#add your code below
g7 = ['United States', 'Italy', 'Canada', 'Japan', 'United Kingdom', 'German
g7_avg_life_expectancy = countries_average(df, g7, 'life_expectancy')
g7_avg_life_expectancy
```

Out[21]:

```
82.10571428571428
```

Q.17 Refer to the `countries` DataFrame. Find the country with lowest value for `life_expectancy` in the `countries` DataFrame, and create a string which is formatted as follows:

'{country} has a life expectancy of {diff} years lower than the G7 average.'

Assign your string to the variable `headline` and ensure it is formatted exactly as above, with:

- use `f-strings` to format the string
- {country} being replaced by the value in the `location` column of the DataFrame
- {diff} being replaced by a float **rounded to one decimal place**, of the value from the `life_expectancy` column subtracted from `g7_avg_life_expectancy` . Please note that {diff} should be a positive value

```
diff = <G7 countries average life expectancy> - <value of the lowest life expectancy country>
```

See below code syntax for some guidance:

```
lowest = countries[countries['life_expectancy'] == countries['life_
expectancy'].min()].iloc[0]
country = lowest['location']
life_exp = lowest['life_expectancy']
```

In [56]:

```
#add your code below



lowest = countries[countries['life_expectancy'] == countries['life_expectanc
country = lowest['location']
life_exp = lowest['life_expectancy']
diff = g7_avg_life_expectancy - life_exp
diff = round(diff, 1)
headline = f'{country} has a life expectancy of {diff} years lower than the
headline
```

Out[56]:

```
'Central African Republic has a life expectancy of 28.8 years lower than th
e G7 average.'
```

In [ ]:

In [ ]: