Unit 3 - Missing values and Data statistics

1. Find rows with missing values
2. Remove missing values using dropna()
3. Fill missing values using fillna()
4. Fill missing values using interpolate()
5. A note on slicing - copy()
6. GroupBy()

In [8]:
```
import pandas as pd
import numpy as np
```

In [9]:
```
url = 'https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/vaccinations/vaccinations.csv'
vacc_df = pd.read_csv(url)
vacc_df.shape
```

Out[9]:
```
(36848, 12)
```

1. Find rows with missing values

`null` / `na` - no value

`NaN` - **N**ot **a** **N**umber - the value is missing. This value will be ignored in calculations such as `.mean()`

`isnull()` is a pandas function, so either use it on a dataframe or call it through pd

In [29]:
```
vacc_df.isnull().sum()
```

Out[29]:
```
location                                   0
iso_code                                   0
date                                       0
total_vaccinations                     15328
people_vaccinated                      16203
people_fully_vaccinated                19253
daily_vaccinations_raw                 18696
daily_vaccinations                       277
total_vaccinations_per_hundred         15328
people_vaccinated_per_hundred          16203
people_fully_vaccinated_per_hundred    19253
daily_vaccinations_per_million           277
month                                      0
dtype: int64
```

call it through pandas:

In [4]:
```
pd.isnull(vacc_df).sum()
```

Out[4]:
```
location                                   0
iso_code                                   0
date                                       0
total_vaccinations                      9949
people_vaccinated                      10735
people_fully_vaccinated                13585
daily_vaccinations_raw                 12189
daily_vaccinations                       241
total_vaccinations_per_hundred          9949
people_vaccinated_per_hundred          10735
people_fully_vaccinated_per_hundred    13585
daily_vaccinations_per_million           241
dtype: int64
```

View specific columns:

In [5]:
```
vacc_df[['daily_vaccinations', 'total_vaccinations']].notnull().sum()
```

Out[5]:
```
daily_vaccinations    24660
total_vaccinations    14952
dtype: int64
```

In [6]:
```
vacc_df[['daily_vaccinations']].isnull().sum()
```

Out[6]:
```
daily_vaccinations    241
dtype: int64
```

Using numpy: `isnan` is a numpy function

In [7]:
```
np.isnan(vacc_df[['daily_vaccinations']]).sum()
```

Out[7]:
```
daily_vaccinations    241
dtype: int64
```

2. Remove missing values using dropna()

Look at Zimbabwe for example. Zimbabwe contains missing values:

In [8]:
```
zimbabwe = vacc_df.loc[vacc_df.location == 'Zimbabwe']
#zimbabwe.head(10)
```

In [9]:
```
zimbabwe[['total_vaccinations']].isnull().sum()
```

Out[9]:
```
total_vaccinations    3
dtype: int64
```

In [10]:
```
zimbabwe['total_vaccinations'].notnull().sum()
```

Out[10]:
```
107
```

We can see the difference when counting the number of values per row:

In [11]:
```
zimbabwe.count()
```

Out[11]:
```
location                               110
iso_code                               110
date                                   110
total_vaccinations                     107
people_vaccinated                      107
people_fully_vaccinated                 78
daily_vaccinations_raw                 105
daily_vaccinations                     109
total_vaccinations_per_hundred         107
people_vaccinated_per_hundred          107
people_fully_vaccinated_per_hundred     78
daily_vaccinations_per_million         109
dtype: int64
```

Remove all rows that contain one or more missing values:

In [12]:
```
zimbabwe.dropna()
```

Out[12]:

	location	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations
24823	Zimbabwe	ZWE	2021-03-22	43574.0	43294.0	280.0	845.0	845.0
24824	Zimbabwe	ZWE	2021-03-23	45197.0	44135.0	1062.0	1623.0	807.0
24825	Zimbabwe	ZWE	2021-03-24	51893.0	49404.0	2489.0	6696.0	1755.0
24826	Zimbabwe	ZWE	2021-03-25	58987.0	54892.0	4095.0	7094.0	2712.0
24827	Zimbabwe	ZWE	2021-03-26	67662.0	61093.0	6569.0	8675.0	3711.0
...	...	...	...	...	...	...	...	...
24896	Zimbabwe	ZWE	2021-06-03	1048504.0	684164.0	364340.0	8290.0	13588.0
24897	Zimbabwe	ZWE	2021-06-04	1056238.0	685564.0	370674.0	7734.0	11349.0
24898	Zimbabwe	ZWE	2021-06-05	1061951.0	686636.0	375315.0	5713.0	8498.0
24899	Zimbabwe	ZWE	2021-06-06	1068107.0	687321.0	380786.0	6156.0	8019.0
24900	Zimbabwe	ZWE	2021-06-07	1073971.0	688696.0	385275.0	5864.0	7699.0

78 rows × 12 columns

Note: `dropna()`, like most other functions in the pandas API returns a new DataFrame (a copy of the original with changes) as the result, so you should assign it back if you want to save changes:

In [13]:
```
zimbabwe.head()
```

Out[13]:

	location	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations
24791	Zimbabwe	ZWE	2021-02-18	0.0	0.0	NaN	NaN	NaN
24792	Zimbabwe	ZWE	2021-02-19	NaN	NaN	NaN	NaN	328.0
24793	Zimbabwe	ZWE	2021-02-20	NaN	NaN	NaN	NaN	328.0
24794	Zimbabwe	ZWE	2021-02-21	NaN	NaN	NaN	NaN	328.0
24795	Zimbabwe	ZWE	2021-02-22	1314.0	1314.0	NaN	NaN	328.0

assign it back:

In [7]:
```
zimbabwe = zimbabwe.dropna()
zimbabwe
```

Out[7]:

	location	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations
36719	Zimbabwe	ZWE	2021-03-22	43574.0	43294.0	280.0	845.0	845.0
36720	Zimbabwe	ZWE	2021-03-23	45197.0	44135.0	1062.0	1623.0	807.0
36721	Zimbabwe	ZWE	2021-03-24	51893.0	49404.0	2489.0	6696.0	1755.0
36722	Zimbabwe	ZWE	2021-03-25	58987.0	54892.0	4095.0	7094.0	2712.0
36723	Zimbabwe	ZWE	2021-03-26	67662.0	61093.0	6569.0	8675.0	3711.0
...	...	...	...	...	...	...	...	...
36843	Zimbabwe	ZWE	2021-07-24	2116664.0	1438890.0	677774.0	44604.0	49319.0
36844	Zimbabwe	ZWE	2021-07-25	2127402.0	1447342.0	680060.0	10738.0	48838.0
36845	Zimbabwe	ZWE	2021-07-26	2178709.0	1491493.0	687216.0	51307.0	50153.0
36846	Zimbabwe	ZWE	2021-07-27	2216835.0	1522150.0	694685.0	38126.0	45643.0
36847	Zimbabwe	ZWE	2021-07-28	2275416.0	1562285.0	713131.0	58581.0	46563.0

127 rows × 12 columns

Remove all values for a specific column - using `subset`

In [15]:
```
zimbabwe.dropna(subset = ['total_vaccinations'])
```

Out[15]:

	location	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations
24791	Zimbabwe	ZWE	2021-02-18	0.0	0.0	NaN	NaN	NaN
24795	Zimbabwe	ZWE	2021-02-22	1314.0	1314.0	NaN	NaN	328.0
24796	Zimbabwe	ZWE	2021-02-23	4041.0	4041.0	NaN	2727.0	808.0
24797	Zimbabwe	ZWE	2021-02-24	7872.0	7872.0	NaN	3831.0	1312.0
24798	Zimbabwe	ZWE	2021-02-25	11007.0	11007.0	NaN	3135.0	1572.0
...	...	...	...	...	...	...	...	...
24896	Zimbabwe	ZWE	2021-06-03	1048504.0	684164.0	364340.0	8290.0	13588.0
24897	Zimbabwe	ZWE	2021-06-04	1056238.0	685564.0	370674.0	7734.0	11349.0
24898	Zimbabwe	ZWE	2021-06-05	1061951.0	686636.0	375315.0	5713.0	8498.0
24899	Zimbabwe	ZWE	2021-06-06	1068107.0	687321.0	380786.0	6156.0	8019.0
24900	Zimbabwe	ZWE	2021-06-07	1073971.0	688696.0	385275.0	5864.0	7699.0

107 rows × 12 columns

For more columns:

In [16]:
```
zimbabwe.dropna(subset = ['total_vaccinations', 'daily_vaccinations_per_million']).head()
```

Out[16]:

	location	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations
24795	Zimbabwe	ZWE	2021-02-22	1314.0	1314.0	NaN	NaN	328.0
24796	Zimbabwe	ZWE	2021-02-23	4041.0	4041.0	NaN	2727.0	808.0
24797	Zimbabwe	ZWE	2021-02-24	7872.0	7872.0	NaN	3831.0	1312.0
24798	Zimbabwe	ZWE	2021-02-25	11007.0	11007.0	NaN	3135.0	1572.0
24799	Zimbabwe	ZWE	2021-02-26	12579.0	12579.0	NaN	1572.0	1750.0

A summary of the functions so far:

- `.isnull()` - display rows that contain missing values
- `.notnull()` - display rows that don't contain missing values
- `.dropna()` - Remove rows with missing values according to parameters:
  - `.dropna()` (default) - drops rows if at least one column has NaN
  - `.dropna(subset = ['column_name'])` - drop rows that contain missing values in the subset of column names
  - `.dropna(how='all')` - drops rows only if all of its columns have NaNs
  - `.dropna(thresh = k)` - k how many non-null values you want to keep (k=3 means the row should contain at least 3 non-null values)
  - `.dropna(axis=1)` - drop columns instead of rows

See documentation here.

3. Fill missing values using fillna()

Use `.fillna()` to fill missing dataframe values with:

- Whatever value you choose
- Mean, median, mode

This is called *imputation*

Replace all NaNs with 0s

In [17]:
```
vacc_df.fillna(0, inplace = False )
vacc_df
```

Out[17]:

	location	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations
0	Afghanistan	AFG	2021-02-22	0.0	0.0	NaN	NaN	NaN
1	Afghanistan	AFG	2021-02-23	NaN	NaN	NaN	NaN	1367.0
2	Afghanistan	AFG	2021-02-24	NaN	NaN	NaN	NaN	1367.0
3	Afghanistan	AFG	2021-02-25	NaN	NaN	NaN	NaN	1367.0
4	Afghanistan	AFG	2021-02-26	NaN	NaN	NaN	NaN	1367.0
...	...	...	...	...	...	...	...	...
24896	Zimbabwe	ZWE	2021-06-03	1048504.0	684164.0	364340.0	8290.0	13588.0
24897	Zimbabwe	ZWE	2021-06-04	1056238.0	685564.0	370674.0	7734.0	11349.0
24898	Zimbabwe	ZWE	2021-06-05	1061951.0	686636.0	375315.0	5713.0	8498.0
24899	Zimbabwe	ZWE	2021-06-06	1068107.0	687321.0	380786.0	6156.0	8019.0
24900	Zimbabwe	ZWE	2021-06-07	1073971.0	688696.0	385275.0	5864.0	7699.0

24901 rows × 12 columns

`inplace = False` is the default. This doesn't change the vacc_df dataframe.

To change it you need:

`vacc_df.fillna(0 , inplace = True)`

or to assign:

`vacc_df = vacc_df.fillna(0)`

But we won't do that! This is where some **business understanding** comes in: it's not a good idea to fill a column like `total_vaccinations` with 0s.

See what happens:

In [18]:
```
vacc_df.fillna(0).head(10)
```

Out[18]:

	location	iso_code	date	total_vaccinations	people_vaccinated	people_fully_vaccinated	daily_vaccinations_raw	daily_vaccinations
0	Afghanistan	AFG	2021-02-22	0.0	0.0	0.0	0.0	0.0
1	Afghanistan	AFG	2021-02-23	0.0	0.0	0.0	0.0	1367.0
2	Afghanistan	AFG	2021-02-24	0.0	0.0	0.0	0.0	1367.0
3	Afghanistan	AFG	2021-02-25	0.0	0.0	0.0	0.0	1367.0
4	Afghanistan	AFG	2021-02-26	0.0	0.0	0.0	0.0	1367.0
5	Afghanistan	AFG	2021-02-27	0.0	0.0	0.0	0.0	1367.0
6	Afghanistan	AFG	2021-02-28	8200.0	8200.0	0.0	0.0	1367.0
7	Afghanistan	AFG	2021-03-01	0.0	0.0	0.0	0.0	1580.0
8	Afghanistan	AFG	2021-03-02	0.0	0.0	0.0	0.0	1794.0
9	Afghanistan	AFG	2021-03-03	0.0	0.0	0.0	0.0	2008.0

So we'll use 0's only for the daily_vaccinations columns, and perhaps for some other columns (which?)

In [19]:
```
vacc_df['daily_vaccinations'].fillna(0, inplace = True)
```

checkout some of the data to see that it works

In [20]:
```
vacc_df.iloc[0:3,[0,2,7]]
```

Out[20]:

	location	date	daily_vaccinations
0	Afghanistan	2021-02-22	0.0
1	Afghanistan	2021-02-23	1367.0
2	Afghanistan	2021-02-24	1367.0

Other options - using central measures:

In [21]:
```
# Using median
vacc_df['daily_vaccinations'].fillna(vacc_df['daily_vaccinations'].median(), inplace=True)

# Using mean
#vacc_df['daily_vaccinations'].fillna(vacc_df['daily_vaccinations'].mean(), inplace=True)

# Using mode
#vacc_df['daily_vaccinations'].fillna(vacc_df['daily_vaccinations'].mode(), inplace=True)
```

What about `total_vaccinations` - there are some `NaN` s there as well:

In [22]:
```
vacc_df.iloc[52:62,[0,2,3]]
```

Out[22]:

	location	date	total_vaccinations
52	Afghanistan	2021-04-15	NaN
53	Afghanistan	2021-04-16	NaN
54	Afghanistan	2021-04-17	NaN
55	Afghanistan	2021-04-18	NaN
56	Afghanistan	2021-04-19	NaN
57	Afghanistan	2021-04-20	NaN
58	Afghanistan	2021-04-21	NaN
59	Afghanistan	2021-04-22	240000.0
60	Afghanistan	2021-04-23	NaN
61	Afghanistan	2021-04-24	NaN

For the `total_vaccinations` we'll use `ffill` which fills the missing values with first non-missing value that occurs before it.

Yes, `bfill` exists as well. (I does what you think it does :)

In [23]:
```
vacc_df[['total_vaccinations']].fillna(method='ffill')[52:62]
#vacc_df['total_vaccinations'][52:62]
```

Out[23]:

	total_vaccinations
52	120000.0
53	120000.0
54	120000.0
55	120000.0
56	120000.0
57	120000.0
58	120000.0
59	240000.0
60	240000.0
61	240000.0

The first value for some country might be NaN

Business understanding: this isn't good enough! We need to aggregate by country!!

Use `groupby()` and `apply` (This is more advanced and we will return to it shortly)

We will create a new column here, `newTotal` - so we can see the difference in `total_vaccinations`

In [24]:
```
vacc_df['newTotal'] = vacc_df.groupby('location')[['total_vaccinations']].apply(lambda x: x.fillna(method='ffi
vacc_df.iloc[52:62,[0,2,3,12]]
```

Out[24]:

	location	date	total_vaccinations	newTotal
52	Afghanistan	2021-04-15	NaN	120000.0
53	Afghanistan	2021-04-16	NaN	120000.0
54	Afghanistan	2021-04-17	NaN	120000.0
55	Afghanistan	2021-04-18	NaN	120000.0
56	Afghanistan	2021-04-19	NaN	120000.0
57	Afghanistan	2021-04-20	NaN	120000.0
58	Afghanistan	2021-04-21	NaN	120000.0
59	Afghanistan	2021-04-22	240000.0	240000.0
60	Afghanistan	2021-04-23	NaN	240000.0
61	Afghanistan	2021-04-24	NaN	240000.0

4. Fill missing values using interpolate()

In [25]:
```
vacc_df['newTotal2'] = vacc_df[['total_vaccinations']].interpolate(method ='linear')
vacc_df.iloc[52:62,[0,2,3,12, 13]]
```

Out[25]:

	location	date	total_vaccinations	newTotal	newTotal2
52	Afghanistan	2021-04-15	NaN	120000.0	184000.000000
53	Afghanistan	2021-04-16	NaN	120000.0	192000.000000
54	Afghanistan	2021-04-17	NaN	120000.0	200000.000000
55	Afghanistan	2021-04-18	NaN	120000.0	208000.000000
56	Afghanistan	2021-04-19	NaN	120000.0	216000.000000
57	Afghanistan	2021-04-20	NaN	120000.0	224000.000000
58	Afghanistan	2021-04-21	NaN	120000.0	232000.000000
59	Afghanistan	2021-04-22	240000.0	240000.0	240000.000000
60	Afghanistan	2021-04-23	NaN	240000.0	253921.157895
61	Afghanistan	2021-04-24	NaN	240000.0	267842.315789

A summary of the functions so far:

- `.fillna()` - fill missing values according to parameters:
  - `.fillna('k')` - with value k, create a new dataframe
  - `.fillna('k', inplace = True)` - with value k, into the existing dataframe
  - `.fillna(method='ffill')` - fill with first non-missing value that occurs before it
  - `.fillna(method='bfill')` - fill with first non-missing value that occurs after it
- `interpolate` - fill using some interpolation technique

See documentation:

* Missing data handling documentation

5. A note on slicing

Slicing is taking only part of a dataframe. For example - the slice we named zimbabwe:

In [26]:
```
zimbabwe = vacc_df.loc[vacc_df.location == 'Zimbabwe']
```

When we change data in a slice, we are changing the ORIGINAL dataframe. This will cause a warning to appear:

In [27]:
```
zimbabwe.fillna(0, inplace=True)
```

```
C:\Users\lihi\anaconda3\lib\site-packages\pandas\core\frame.py:4317: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  return super().fillna(
```

The warning will disappear if you rerun the command, but it can still be scary. Best way to avoid it is to create a `copy` of the dataframe:

In [28]:
```
zimbabwe = vacc_df.loc[vacc_df.location == 'Zimbabwe'].copy()
zimbabwe.fillna(0, inplace=True)
```

This works fine, no warnings. But - this won't change the original dataframe (which might be a good thing, if you didn't plan to change it, or a bad thing, if you did)

What about changes in the original dataframe? They will not change the copy.

If you do want your copy to change, use a shallow copy:

In [29]:
```
small_example = pd.Series([1, 2], index=["a", "b"])
small_example
```

Out[29]:
```
a    1
b    2
dtype: int64
```

In [30]:
```
my_deep_copy = small_example.copy()
my_deep_copy
```

Out[30]:
```
a    1
b    2
dtype: int64
```

In [31]:
```
my_shallow_copy = small_example.copy(deep=False)
my_shallow_copy
```

Out[31]:
```
a    1
b    2
dtype: int64
```

Make a change to the dataframe - where will it appear?

In [32]:
```
small_example[0] = -100
small_example
```

Out[32]:
```
a   -100
b      2
dtype: int64
```

In [33]:
```
my_deep_copy
```

Out[33]:
```
a    1
b    2
dtype: int64
```

In [34]:
```
my_shallow_copy
```

Out[34]:
```
a   -100
b      2
dtype: int64
```

A summary:

- `.copy()` - creates a copy of the slice of the dataframe
- `.copy(deep=False)` - updates to the original dataframe will show in the copy

6. Groupby()

How do we fill missing values for `total_vaccinations` according to the mean of each country?

How do we fill missing values for `daily_vaccinations` according to the mean of each country each month?

For this, we need to use groupby

groupby documentation

Group according to something + select some columns + do something on the result

The `mean` of `daily_vaccinations` according to `location`:

In [8]:
```
vacc_df.groupby('location')[['daily_vaccinations']].mean()
```

Out[8]:

location	daily_vaccinations
Afghanistan	7.706303e+03
Africa	3.087572e+05
Albania	5.650316e+03
Algeria	3.289048e+03
Andorra	4.382229e+02
...	...
Wallis and Futuna	7.283193e+01
World	1.641670e+07
Yemen	3.834722e+03
Zambia	3.400229e+03
Zimbabwe	1.299056e+04

231 rows × 1 columns

Note that this format means `location` is now the index

Try running the below commands:

In [13]:
```
df_by_loc = vacc_df.groupby('location')[['daily_vaccinations']].mean()
#df_by_loc[["location"]]   #this will result in an error
#df_by_loc[["daily_vaccinations"]]   #this is OK
```

If you plan to continue using this data and need the index as an attribute:

Two possible solutions:

set `as_index=False`

add `reset_index()`

In [36]:
```
#vacc_df.groupby('location', as_index = False)[['daily_vaccinations']].mean()
vacc_df.groupby('location')[['daily_vaccinations']].mean().reset_index()
```

Out[36]:

	location	daily_vaccinations
0	Afghanistan	7.706303e+03
1	Africa	3.087572e+05
2	Albania	5.650316e+03
3	Algeria	3.289048e+03
4	Andorra	4.382229e+02
...	...	...
226	Wallis and Futuna	7.283193e+01
227	World	1.641670e+07
228	Yemen	3.834722e+03
229	Zambia	3.400229e+03
230	Zimbabwe	1.299056e+04

231 rows × 2 columns

Groupby 2 or more columns is possible

For example: create a dataframe with the mean daily vaccinations per country per month

First, change the `date` into a `datetime` object and extract the month

In [22]:
```
vacc_df['date'] = pd.to_datetime(vacc_df['date'])
vacc_df['month'] = pd.DatetimeIndex(vacc_df['date']).month
```

Now, groupby both `location` and `month`

In [23]:
```
vacc_df.groupby(['location','month'])[['daily_vaccinations', 'total_vaccinations']].mean().reset_index()
```

Out[23]:

	location	month	daily_vaccinations	total_vaccinations
0	Afghanistan	2	1367.000000	4.100000e+03
1	Afghanistan	3	2770.774194	5.400000e+04
2	Afghanistan	4	7320.200000	1.800000e+05
3	Afghanistan	5	9220.580645	5.682665e+05
4	Afghanistan	6	8096.633333	7.211901e+05
...	...	...	...	...
1372	Zimbabwe	3	2153.000000	4.566529e+04
1373	Zimbabwe	4	11928.800000	2.844643e+05
1374	Zimbabwe	5	17447.483871	7.858874e+05
1375	Zimbabwe	6	9797.433333	1.138461e+06
1376	Zimbabwe	7	31209.178571	1.715205e+06

1377 rows × 4 columns

what will happen if we swith the order of the indexes?

try running the following:

In [56]:
```
#vacc_df.groupby(['month', 'location'])[['daily_vaccinations', 'total_vaccinations']].mean().reset_index()
```

Still the same, but using a lambda function:

In [24]:
```
vacc_df.groupby(['location', 'month'])[['daily_vaccinations', 'total_vaccinations']].apply(lambda x: x.mean()).
```

Out[24]:

	location	month	daily_vaccinations	total_vaccinations
0	Afghanistan	2	1367.000000	4.100000e+03
1	Afghanistan	3	2770.774194	5.400000e+04
2	Afghanistan	4	7320.200000	1.800000e+05
3	Afghanistan	5	9220.580645	5.682665e+05
4	Afghanistan	6	8096.633333	7.211901e+05
...	...	...	...	...
1372	Zimbabwe	3	2153.000000	4.566529e+04
1373	Zimbabwe	4	11928.800000	2.844643e+05
1374	Zimbabwe	5	17447.483871	7.858874e+05
1375	Zimbabwe	6	9797.433333	1.138461e+06
1376	Zimbabwe	7	31209.178571	1.715205e+06

1377 rows × 4 columns

So now we are ready to answer the questions:

How do we fill missing values for `total_vaccinations` according to the mean of each country?

We now understand this:

In [25]:
```
vacc_df['newTotal'] = vacc_df.groupby('location')[['total_vaccinations']].apply(lambda x: x.fillna(method='ffi
vacc_df[['location', 'new_total', 'total_vaccinations']].iloc[[10,15],:]
```

Out[25]:

	location	new_total	total_vaccinations
10	Afghanistan	8200.0	NaN
15	Afghanistan	8200.0	NaN

same results, different synthax (don't worry if you don't understand this):

In [27]:
```
#vacc_df['new_total'] = vacc_df['total_vaccinations'].fillna(vacc_df.groupby(['location'])['total_vaccinations
#vacc_df[['location', 'new_total', 'total_vaccinations']].iloc[[10,15],:]
```

How do we fill missing values for `daily_vaccinations` according to the mean of each country each month?

In [31]:
```
vacc_df.groupby(['location', 'month'])[['daily_vaccinations']].apply(lambda x: x.fillna(x.mean())).reset_index
```

Out[31]:

	location	month	level_2	daily_vaccinations
0	Afghanistan	2	0	1367.0
1	Afghanistan	2	1	1367.0
2	Afghanistan	2	2	1367.0
3	Afghanistan	2	3	1367.0
4	Afghanistan	2	4	1367.0
...	...	...	...	...
36843	Zimbabwe	7	36843	49319.0
36844	Zimbabwe	7	36844	48838.0
36845	Zimbabwe	7	36845	50153.0
36846	Zimbabwe	7	36846	45643.0
36847	Zimbabwe	7	36847	46563.0

36848 rows × 4 columns

We now have more rows than with the mean function, since

`fillna()` is not an aggregation function

You can also group different columns using different functions:

In [12]:

```
vacc_df.groupby('location').agg({'daily_vaccinations': ['first', 'last' , 'mean', 'median', 'max'], 'total_vacc
```

Out[12]:

	daily_vaccinations					total_vaccinations
	first	last	mean	median	max	max
location						
Afghanistan	1367.0	33708.0	8.001713e+03	6571.0	33708.0	1.381416e+06
Africa	500.0	1163209.0	3.131704e+05	263476.0	1163209.0	6.832872e+07
Albania	64.0	9915.0	5.650316e+03	5521.0	17565.0	1.138771e+06
Algeria	30.0	20914.0	1.858461e+04	20914.0	20914.0	3.421279e+06
Andorra	66.0	1762.0	4.382229e+02	257.0	1762.0	8.234900e+04
...	...	...	...	...	...	...
Wallis and Futuna	272.0	7.0	7.283193e+01	21.0	343.0	9.158000e+03
World	0.0	37256430.0	1.651060e+07	15587002.0	43389267.0	4.066651e+09
Yemen	4276.0	939.0	3.834722e+03	3265.0	10240.0	3.114830e+05
Zambia	106.0	13059.0	3.491349e+03	2115.5	13814.0	4.137740e+05
Zimbabwe	328.0	46563.0	1.340446e+04	11950.5	50153.0	2.275416e+06

231 rows × 6 columns

> A summary:
>
> - `.groupby()` - group according to the columns specified
> - `.reset_index()` or set `as_index=False` - adds the current index as a column, adds a new numerical index
> - `pd.to_datetime(df['date'])` - changes the attribute type to datetime
> - `pd.DatetimeIndex(df['date']).month` - extracts the month from the datatime attribute
> - `apply` - applies a function on each row (axis =0) in the dataframe. Change to (axis = 1) to apply the function on each column documentation
> - `lambda` - small anonymous function
> - `agg` - apply multiple functions at once, one for each specified column documentation

This was a lot of information.

Keep your balance. Practice. You will make it.

Photo by Martin Sanchez on Unsplash