**Data Cleaning - Resampling Time Series Data**

First, we have to identify if there are any missing or junk values in our DataFrames.

Challenge

Can you investigate all 4 DataFrames and find if there are any missing values?

If yes, find how many missing or NaN values there are. Then, find the row where the missing values occur.

Finally, remove any rows that contain missing values.

.

.

..

...

..

.

.

**Solution: Finding the missing values**

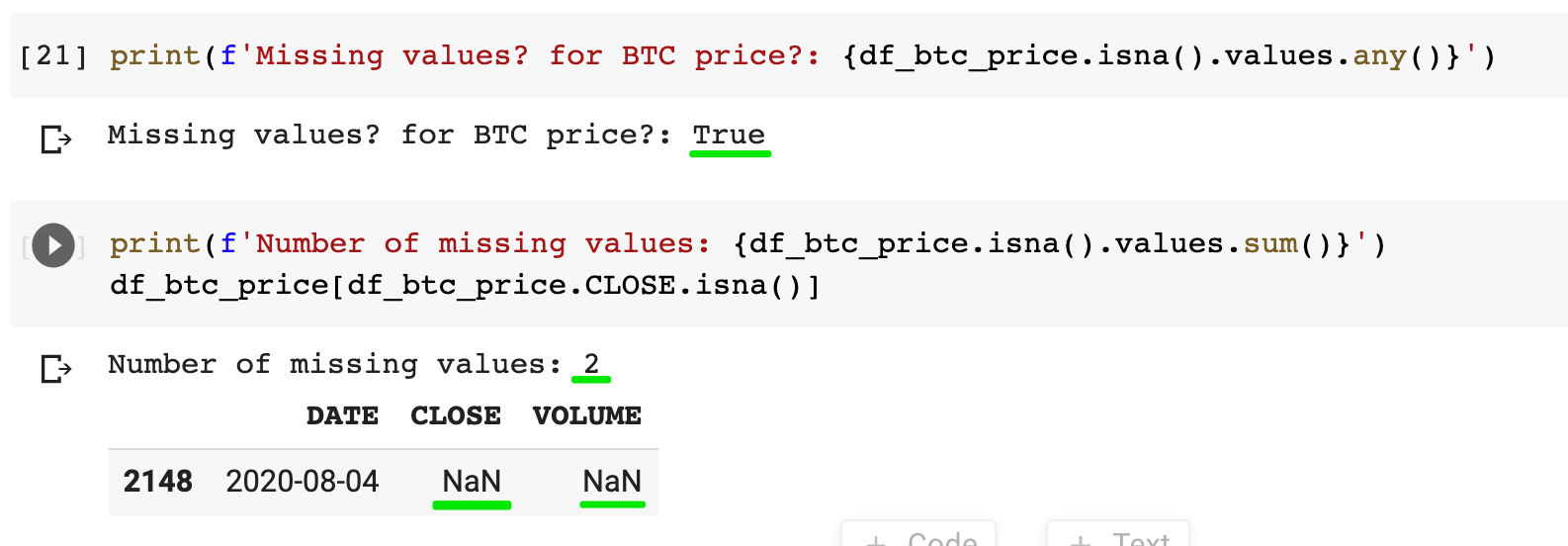
For 3 of the DataFrames there are no missing values. We can verify this using the .isna() method. This will return a whole series of booleans, but we can chain .values.any() to see if any value in the series is True.

1. print(f'Missing values for Tesla?: {df\_tesla.isna().values.any()}')
2. print(f'Missing values for U/E?: {df\_unemployment.isna().values.any()}')
3. print(f'Missing values for BTC Search?: {df\_btc\_search.isna().values.any()}')

However, for the Bitcoin price data, there seems to be a problem. There's a missing value somewhere.

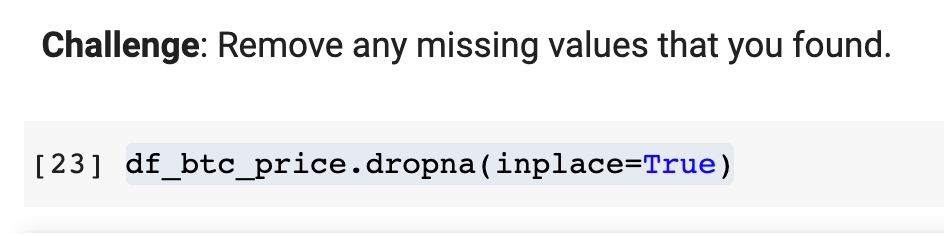
The number of missing values can be found by using .sum() to add up the number of occurrences of True in the series. This shows that there are 2 missing values.

To find the row where the missing values occur, we can create a subset of the DataFrame using .isna() once again (If you've arrived at this answer using a different approach, that's fine too. There are a number of ways to solve this challenge.)



To remove a missing value we can use .dropna(). The inplace argument allows to overwrite our DataFrame and means we don't have to write:

1. df\_btc\_price = df\_btc\_price.dropna()



Challenge

Our DataFrames contain time-series data. Do you remember how to check the data type of the entries in the DataFrame? Have a look at the data types of the MONTH or DATE columns. Convert any strings you find into Datetime objects. Do this for all 4 DataFrames. Double-check if your type conversion was successful.

.

.

..

...

..

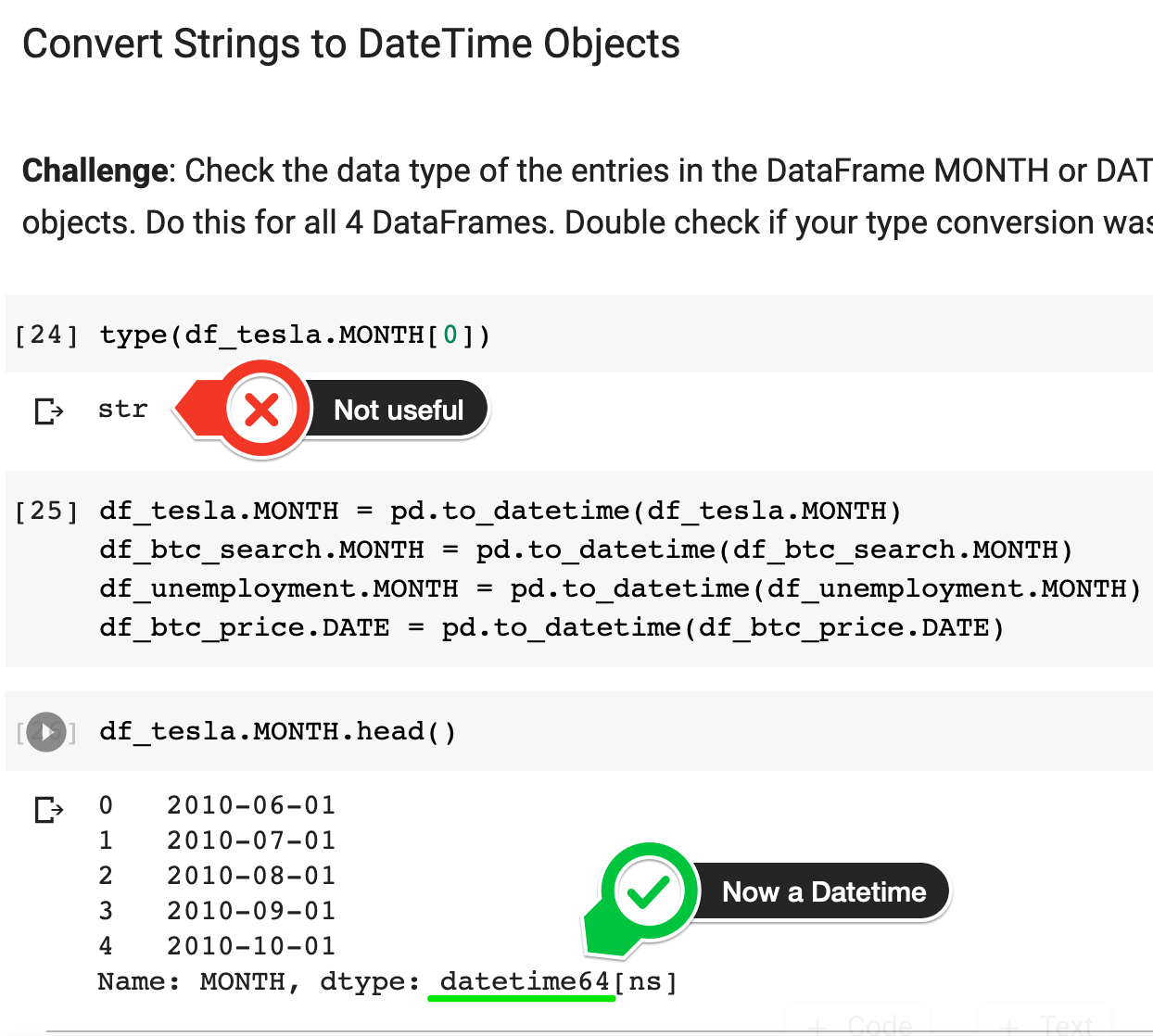
.

.

**Solution: Converting Strings to DateTime Objects**

All the date data in our columns are in the form of strings. To convert this into a Datetime object we're going to use the Pandas .to\_datetime() function.

1. df\_tesla.MONTH = pd.to\_datetime(df\_tesla.MONTH)
2. df\_btc\_search.MONTH = pd.to\_datetime(df\_btc\_search.MONTH)
3. df\_unemployment.MONTH = pd.to\_datetime(df\_unemployment.MONTH)
4. df\_btc\_price.DATE = pd.to\_datetime(df\_btc\_price.DATE)



Resampling Time Series Data

Next, we have to think about how to make our Bitcoin price and our Bitcoin search volume comparable. Our Bitcoin price is daily data, but our Bitcoin Search Popularity is monthly data.

To convert our daily data into monthly data, we're going to use the [.resample()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.resample.html) function. The only things we need to specify is which column to use (i.e., our DATE column) and what kind of sample frequency we want (i.e., the "rule"). We want a monthly frequency, so we use 'M'.  If you ever need to resample a time series to a different frequency, you can find a list of different options [here](https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#dateoffset-objects) (for example 'Y' for yearly or 'T' for minute).

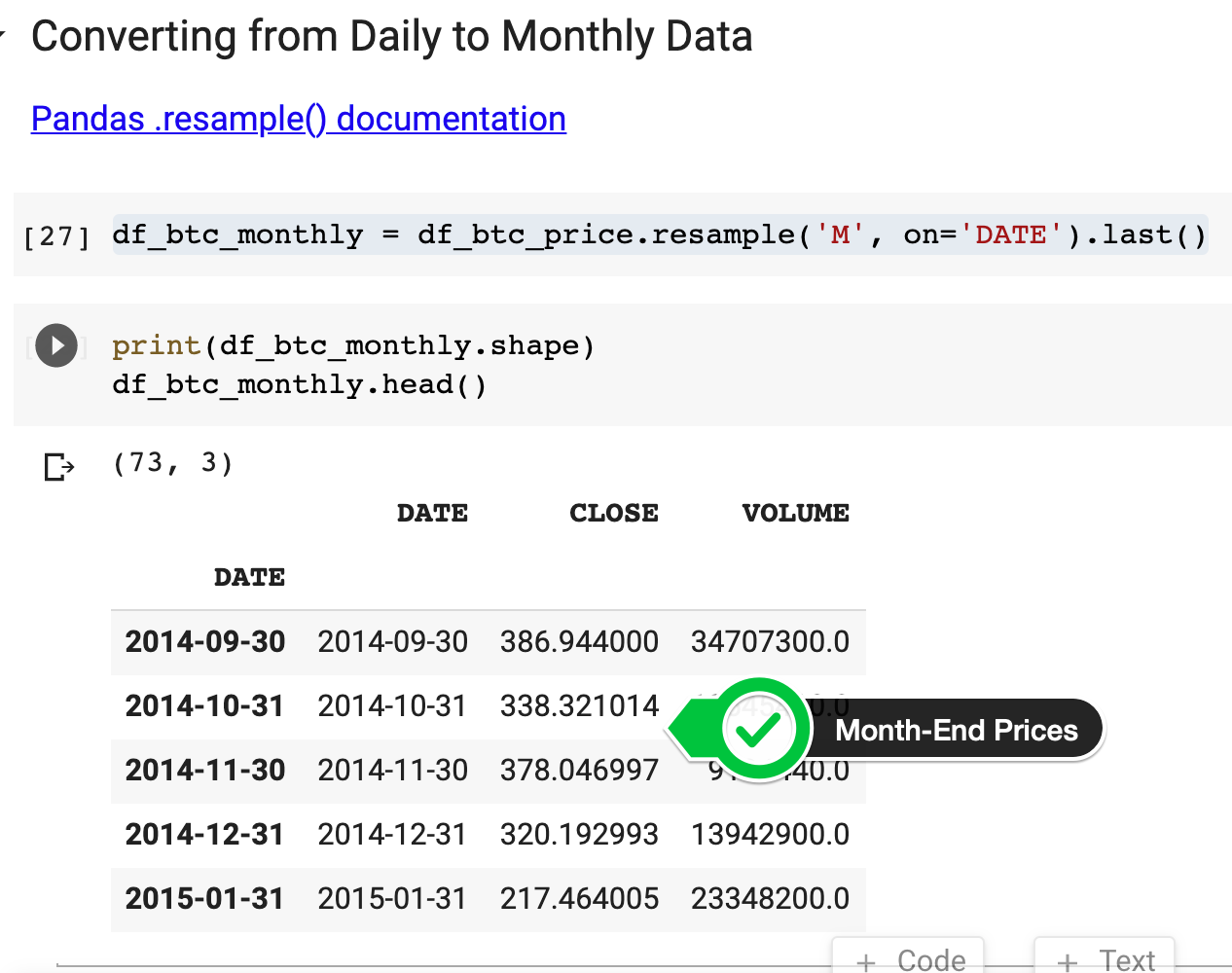
After resampling, we need to figure out how the data should be treated. In our case, we want the last available price of the month - the price at month-end.

1. df\_btc\_monthly = df\_btc\_price.resample('M', on='DATE').last()

If we wanted the average price over the course of the month, we could use something like:

1. df\_btc\_monthly = df\_btc\_price.resample('M', on='DATE').mean()

This is what our data looks like now:



We have 73 rows in our price data - the same as our search data. Nice! 😎