Author and company name with crop mark graphicTitle and subtitle with crop mark graphicColor background

[**Requirements 3**](#_heading=h.ie6tfxh1bs8u)

[Virtual Environment 3](#_heading=h.osr64k4tm1pd)

[Decencies 3](#_heading=h.fafud2ecg7hj)

[**Installation 3**](#_heading=h.msec6dkp5ggo)

[Anaconda/Virtual Environment 3](#_heading=h.ucg3mrddm0gx)

[Dependencies 3](#_heading=h.7vmmtmji1e6c)

[**Data Processing 1 4**](#_heading=h.xu6sg3f9u4x4)

[ensure\_directory\_exist 4](#_heading=h.rjy2wsme3ddc)

[load\_data 4](#_heading=h.5099v93jopf7)

[data\_validation 6](#_heading=h.6x5jrxtss8fd)

[split\_data 8](#_heading=h.xonfnb20nkgs)

[scaler\_features 9](#_heading=h.tepf1g55mvoc)

[create\_datasets 10](#_heading=h.m8vv8ibi7oze)

[**Data Processing 1 - Output 12**](#_heading=h.57fp051odtow)

[Set Up: 12](#_heading=h.qu1z1pg0x8ls)

[Scenario 1 - First Time Running Output, split\_by\_date=True 13](#_heading=h.1cp1cpgr2gos)

[Checking Data Type 13](#_heading=h.3fzwbz4omt5x)

[Raw Data 14](#_heading=h.szt0wb34q86u)

[Validated Data 14](#_heading=h.ltxeld3vfsa4)

[Train Data 15](#_heading=h.15tmxfnh4ew3)

[Test Data 15](#_heading=h.fsgnc5p5nj3c)

[Checking Ratio 16](#_heading=h.k0d99csb48fd)

[Scenario 2 - Loaded Data, split\_by\_date=True 16](#_heading=h.cegqc7i0f8ft)

[Checking Data Type 16](#_heading=h.h0syoqcwuouv)

[Raw Data 17](#_heading=h.dlvrajdz86w4)

[Validated Data 17](#_heading=h.pwts8l486q3)

[Train Data 18](#_heading=h.ccdnuiydxonz)

[Test Data 18](#_heading=h.3opnp8mwsos)

[Checking Ratio 19](#_heading=h.l8t24um37k5z)

[Scenario 3 - First Time Running Output, split\_by\_date=False 19](#_heading=h.rikvrfr8j7f3)

[Checking Data Type 19](#_heading=h.1y7ptx2m84im)

[Raw Data 20](#_heading=h.thz8xkgjwdvj)

[Validated Data 20](#_heading=h.iladxfx7l6pl)

[Train Data 21](#_heading=h.idbqtyr1advi)

[Test Data 22](#_heading=h.f5jfmecy28kb)

[Checking Ratio 22](#_heading=h.44vg7cw58b8e)

[Scenario 4 - Loaded Data, split\_by\_date=False 23](#_heading=h.hklj9fmo9zbv)

[Checking Data Type 23](#_heading=h.7i8l0ivl2hkg)

[Raw Data 24](#_heading=h.es63jh31hhe0)

[Validated Data 24](#_heading=h.lyydulg06c2x)

[Train Data 25](#_heading=h.9seo948ctkr1)

[Test Data 26](#_heading=h.vv3ctnx34l3m)

[Checking Ratio 26](#_heading=h.5myjx76m3qga)

[Testing Summary: 26](#_heading=h.803a6cxmpkp2)

# Requirements

## Virtual Environment

· Google Colab

· Jupyter Notebook

## Decencies

· numpy

· matplotlib

· mplfinance

· pandas

· scikit-learn

· pandas-datareader

· yfinance

· pandas\_ta

# Installation

\*Note: Anaconda is required unless Google Collab is being used

## Anaconda/Virtual Environment

1. Download Anaconda: Go to the Anaconda website (https://www.anaconda.com/products/distribution) and download the appropriate version for your operating system.

2. Install Anaconda: Follow the installation instructions for the OS from the Anaconda website.

3. Open Anaconda Navigator: Launch Anaconda Navigator from your installed applications.

4. Create a New Environment (Required): Create a new environment to isolate Jupyter installation on each project. Click on "Environments" in Navigator and then "Create" to make a new environment.

5. Install Jupyter Notebook: In the selected environment, click on the environment name and select "Open Terminal". In the terminal, type: conda install jupyter.

## Dependencies

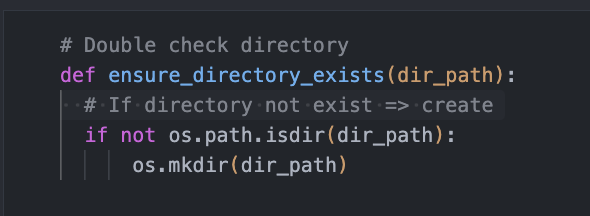
In Google Colab or Jupyter Notebook, it can directly install the required dependencies using the !pip command in code cells. Here's an example of how to install the dependencies:

**!pip install <package> or !pip install -r <text file>**

# 

# Data Processing 1

## ensure\_directory\_exist



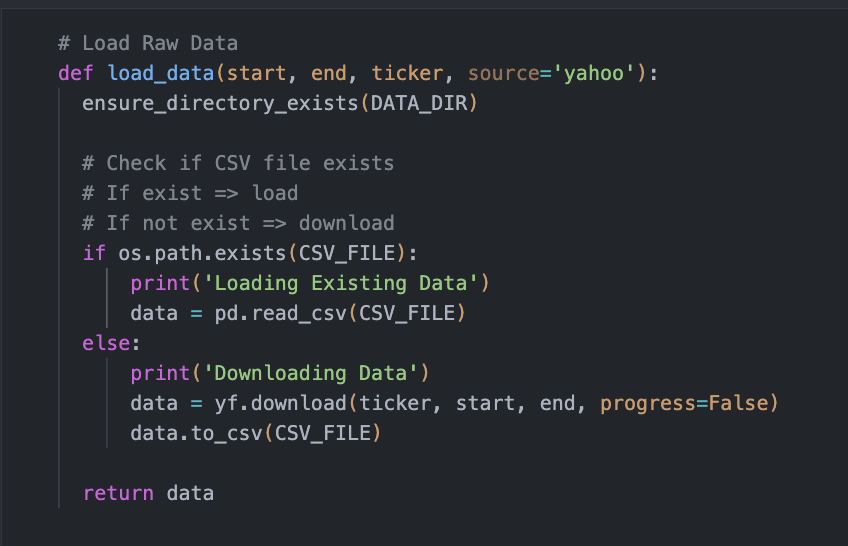
The function `ensure\_directory\_exists` takes the following parameter:

* `dir\_path`: This parameter is a string representing the path of the directory you want to ensure exists.

This function has the following features:

* Checking and Creating Directory: The primary purpose of this function is to ensure that a specified directory exists. It checks if the directory at the provided `dir\_path` exists using the `os.path.isdir` function. If the directory does not exist, it creates the directory using the `os.mkdir` function.

## load\_data



The function `load\_data` takes several parameters:

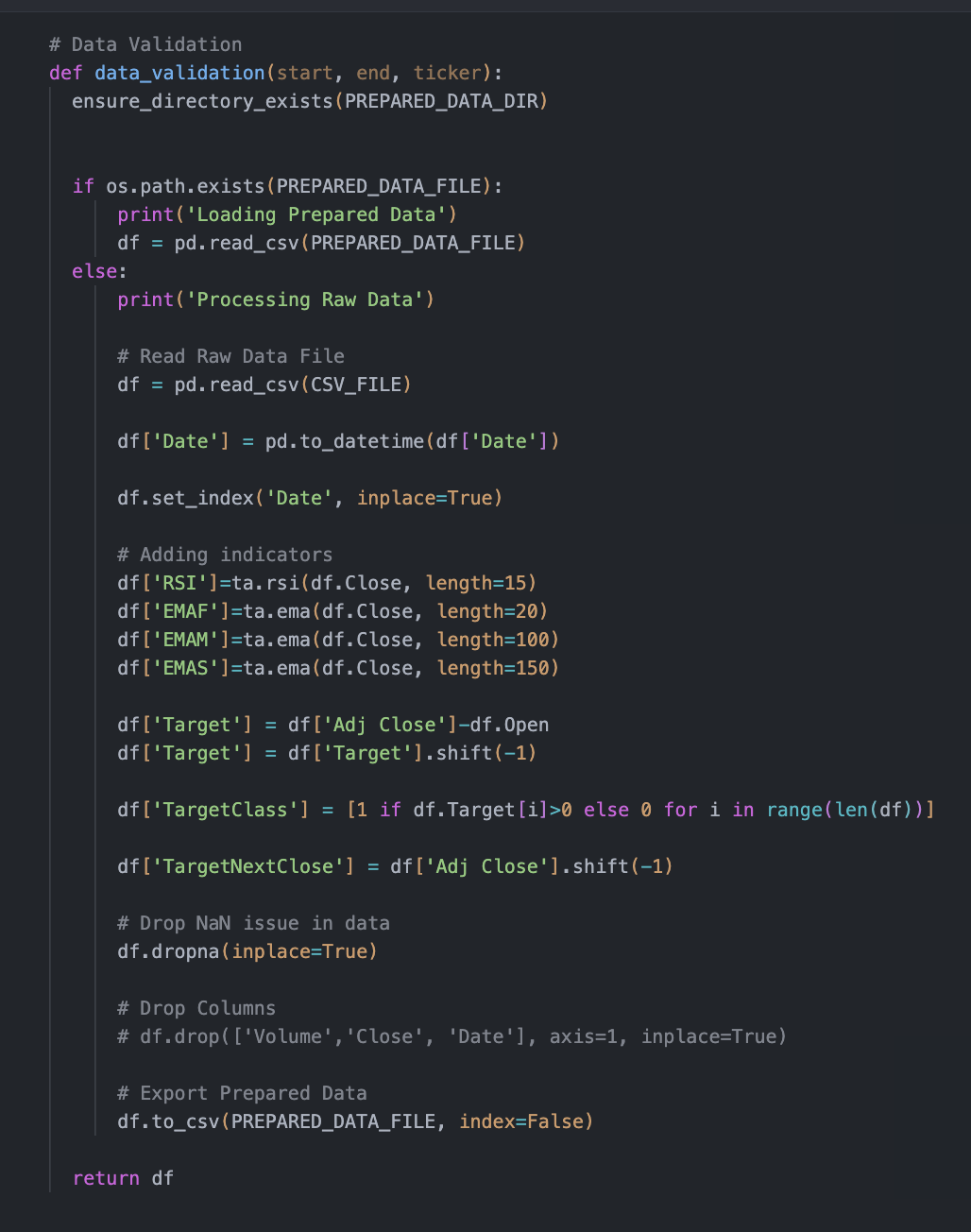
* `start` and `end`: These are date values in the format 'YYYY-MM-DD' that define the time range for the financial data you want to load.
* `ticker`: This parameter is a string representing the stock ticker symbol (e.g., 'AAPL' for Apple Inc.) for which you want to fetch the financial data.
* `source`: This parameter is optional and specifies the data source. The default value is 'yahoo', which refers to Yahoo Finance.

This function has the following features:

* Creating a Directory: The function first ensures that a directory exists to hold the financial data. If the directory doesn't exist, it creates one using the path defined in the DATA\_DIR variable.
* Checking for Existing Data: The function then checks if the financial data already exists by looking for a CSV file at the path specified in the CSV\_FILE variable. If the file is found, the function assumes the data has already been downloaded or loaded and reads it from the CSV using the Pandas library.
* Downloading and Saving Data: If the CSV file containing the financial data doesn't exist, the function assumes that the data needs to be fetched. It uses the yf.download function from Yahoo Finance (based on the specified source) to download the financial data for the given stock ticker and time range. The progress=False argument suppresses progress messages during the download. The downloaded data is then saved to a new CSV file using the to\_csv method.

## 

## data\_validation



The function `data\_validation` takes several parameters:

* `start` and `end`: These are date values in the format 'YYYY-MM-DD' that define the time range for the financial data you want to validate and process.
* `ticker`: This parameter is a string representing the stock ticker symbol (e.g., 'AAPL' for Apple Inc.) for which you intend to validate and preprocess the financial data.

This function has the following features:

* Creating a Directory: The function first ensures that a directory exists to hold the prepared data. If the directory doesn't exist, it creates one using the path defined in the `PREPARED\_DATA\_DIR` variable.
* Checking for Existing Data: The function then checks if prepared data already exists by looking for a CSV file at the path specified in the `PREPARED\_DATA\_FILE` variable. If the file is found, the function assumes the data has already been processed and loads it from the CSV using the Pandas library.
* Processing Raw Data: If the prepared data CSV file doesn't exist, the function assumes that the raw data needs to be processed. It reads the raw financial data from a CSV file located at the path specified in the `CSV\_FILE` variable. Then, the function applies several preprocessing steps to this raw data:
  + Adding Indicators: The function adds indicators to the data, such as the Relative Strength Index (RSI) and various Exponential Moving Averages (EMAF, EMAM, EMAS), calculated using the `ta` library.
  + Calculating Targets: The function calculates the 'Target' column, which represents the difference between the adjusted closing price and the opening price. It also shifts this target one step back to represent the future movement.
  + Creating Target Class: The function generates a binary 'TargetClass' column based on whether the 'Target' is greater than zero, indicating a positive change.
  + TargetNextClose: The function creates a 'TargetNextClose' column by shifting the 'Adj Close' column one step back.
  + Handling Missing Data: The function removes any rows that contain NaN (missing) values.
  + Dropping Columns: Optionally, there are commented-out lines to drop certain columns like 'Volume', 'Close', and 'Date'. You can uncomment these lines if you want to remove these columns from the final processed data.
  + Exporting Prepared Data: Once all the preprocessing steps are complete, the function saves the processed data to a new CSV file using the `to\_csv` method. This ensures that the next time the function is called, it can load the already processed data directly from the CSV file without repeating the preprocessing steps.

## 

## split\_data



The function `split\_data` takes the following parameters:

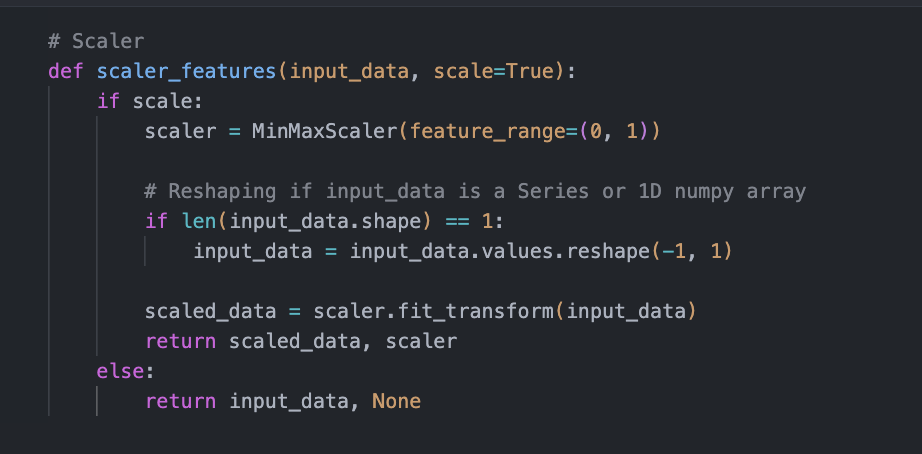
* `df`: This parameter is a DataFrame containing the financial data that you want to split.
* `split\_ratio`: This is the ratio of data to be used for training. The rest will be used for testing. For example, a `split\_ratio` of 0.8 would mean 80% of the data is used for training and 20% for testing.
* `split\_by\_date`: This is an optional boolean parameter (defaulting to `True`) that indicates whether you want to split the data by date or randomly.

This function has the following features:

* Splitting Data by Date: If `split\_by\_date` is set to `True`, the function calculates the index at which the split should occur based on the ratio of data for training. It then splits the DataFrame into two parts: the first part for training (`train\_data`) and the remaining part for testing (`test\_data`).
* Splitting Data Randomly: If `split\_by\_date` is set to `False`, the function uses the `train\_test\_split` function from the scikit-learn library to randomly split the DataFrame into training and testing sets according to the specified `split\_ratio`. The `random\_state=42` ensures reproducibility of the random split.
* Printing Shapes: After splitting, the function prints the shapes of the training and testing data, indicating how many rows and columns each set contains.

## 

## scaler\_features



The function `scaler\_features` takes the following parameters:

* `input\_data`: This parameter represents the data that you want to scale. It could be a pandas Series, a 1D numpy array, or a 2D numpy array.
* `scale`: This is an optional boolean parameter (defaulting to `True`) that indicates whether you want to scale the data.

This function has the following features:

* Scaling Data: If the `scale` parameter is set to `True`, the function creates an instance of the `MinMaxScaler` from scikit-learn. The `feature\_range` parameter sets the range to which the data will be scaled (between 0 and 1).
* Reshaping Data: Before scaling, the function checks if the input data has a shape of 1 dimension (i.e., it's a Series or 1D numpy array). If so, it reshapes the data into a 2D array with one column using the `.reshape(-1, 1)` method. This is necessary because scikit-learn's scaler expects a 2D input.
* Scaling and Transforming Data: The function then uses the scaler to fit and transform the input data, resulting in scaled data. This scaled data is returned along with the scaler instance.
* Not Scaling Data: If the `scale` parameter is set to `False`, the function simply returns the original input data as-is, without any scaling. In this case, the scaler instance returned is `None`.

## 

## create\_datasets



The function `create\_datasets` takes the following parameters:

* `start` and `end`: These are date strings in the format 'YYYY-MM-DD'. They define the time range for downloading the financial data.
* `ticker`: A string that represents the stock ticker symbol for which the datasets are to be created. For example, 'AAPL' for Apple Inc.

This function has the following features:

* Downloading or Loading Raw Data: The function initially calls the `load\_data` function to either download or load existing raw financial data based on the provided `start`, `end`, and `ticker` parameters.
* Data Validation: Once the raw data is loaded, the `data\_validation` function is called to preprocess and validate the data. The cleaned and processed data is stored in a DataFrame (`df`).
* Splitting Data: The `split\_data` function is invoked to partition the cleaned DataFrame (`df`) into training (`train\_data`) and testing (`test\_data`) datasets. The split is determined by a predefined `split\_ratio`.
* Defining Features and Target: The function specifies which columns in the DataFrame will be treated as features (`feature\_columns`) and which one will be used as the target (`target\_column`).
* Preparing Train Datasets: The steps are carried out:
  + Feature scaling is performed using `train\_feature\_scaler`, which scales the feature columns of the training data.
  + Target scaling is done using `train\_target\_scaler`, which scales the target values in the training data.
  + Sequences of scaled features and corresponding target values (`x\_train` and `y\_train`) are generated. These sequences are designed for training time-series models like LSTMs.
* Preparing Test Datasets:
  + Applies the same feature scaler (`train\_feature\_scaler`) that was used on the training data to scale the feature columns.
  + Uses the same target scaler (`train\_target\_scaler`) that was used on the training data to scale the target values.
  + Creates sequences of scaled features and corresponding target values (`x\_test` and `y\_test`) for testing.
* Saving Prepared Train Data: The prepared training sequences (`x\_train` and `y\_train`) and testing sequences (`x\_test` and `y\_test`) are saved to `.npz` files for future use. This is done using the `np.savez` function.
* Returning Prepared Data and Information: The function returns various data and objects:
  + Raw data (`data`)
  + Processed DataFrame (`df`)
  + Training and testing datasets (`train\_data` and `test\_data`)
  + Feature and target scalers (`train\_feature\_scaler` and `train\_target\_scaler`)
  + Prepared training and testing sequences (`x\_train`, `y\_train`, `x\_test`, `y\_test`)

# 

# Data Processing 1 - Output

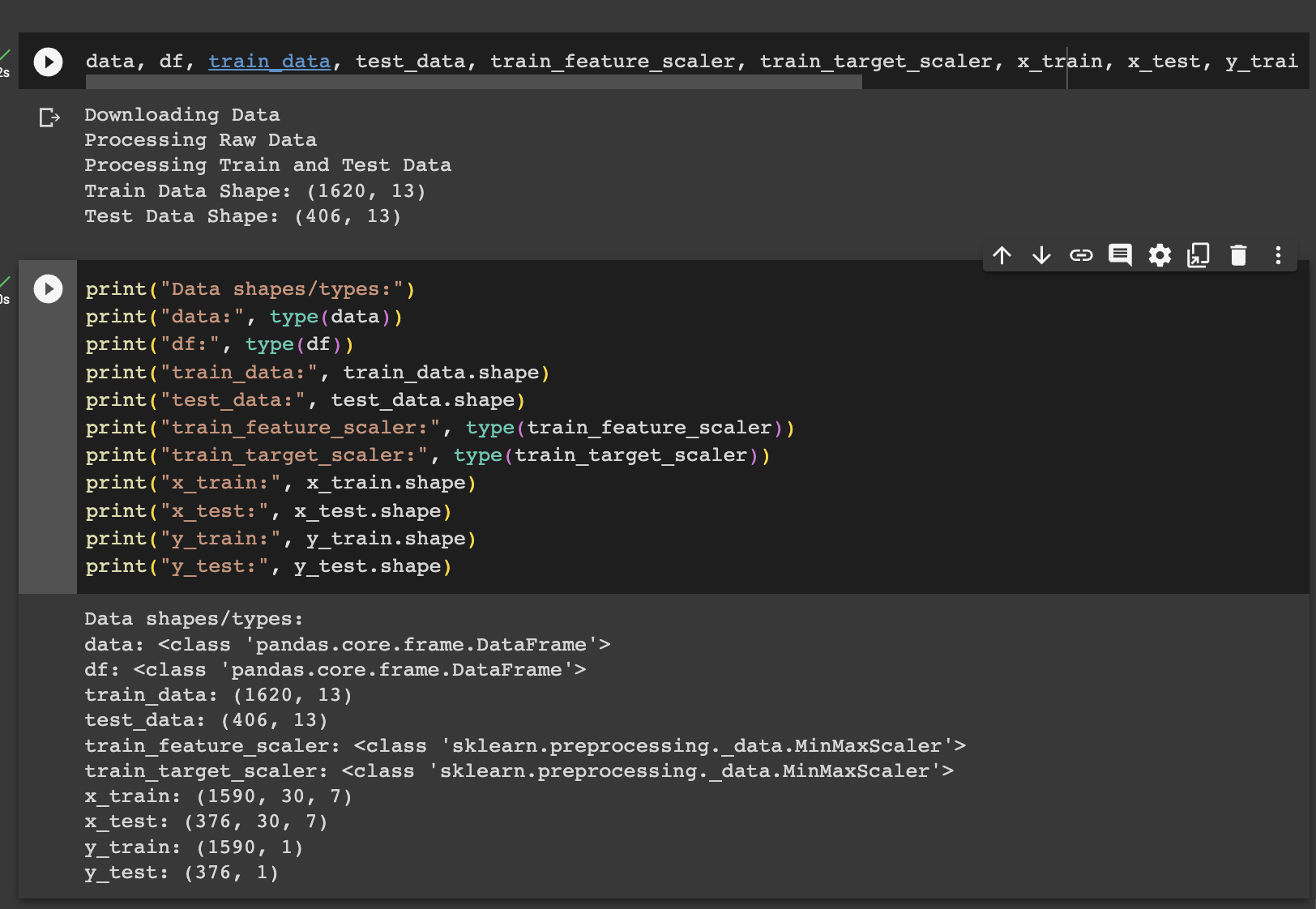
## Set Up:

## 

## 

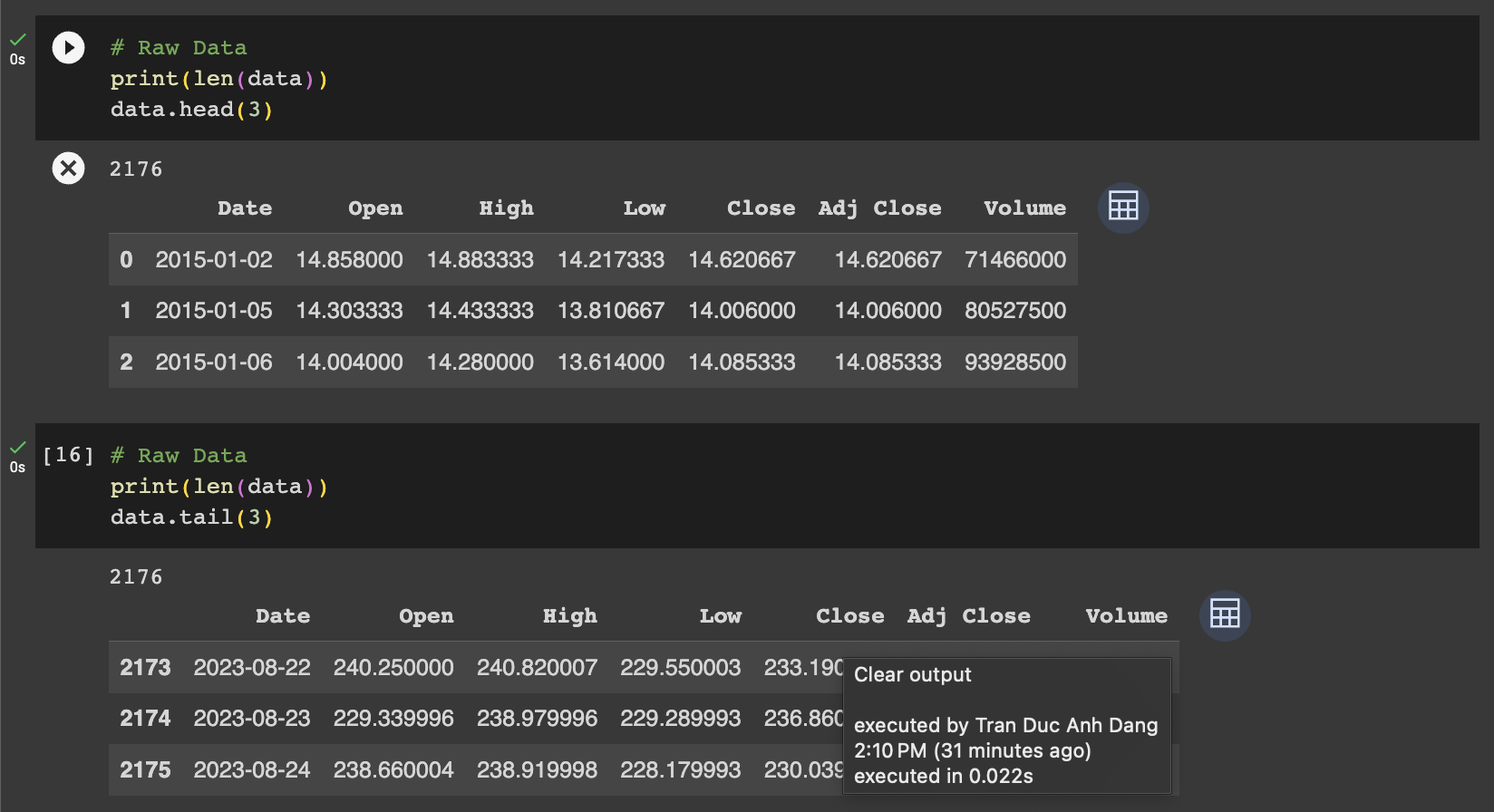
## Scenario 1 - First Time Running Output, split\_by\_date=True

### Checking Data Type



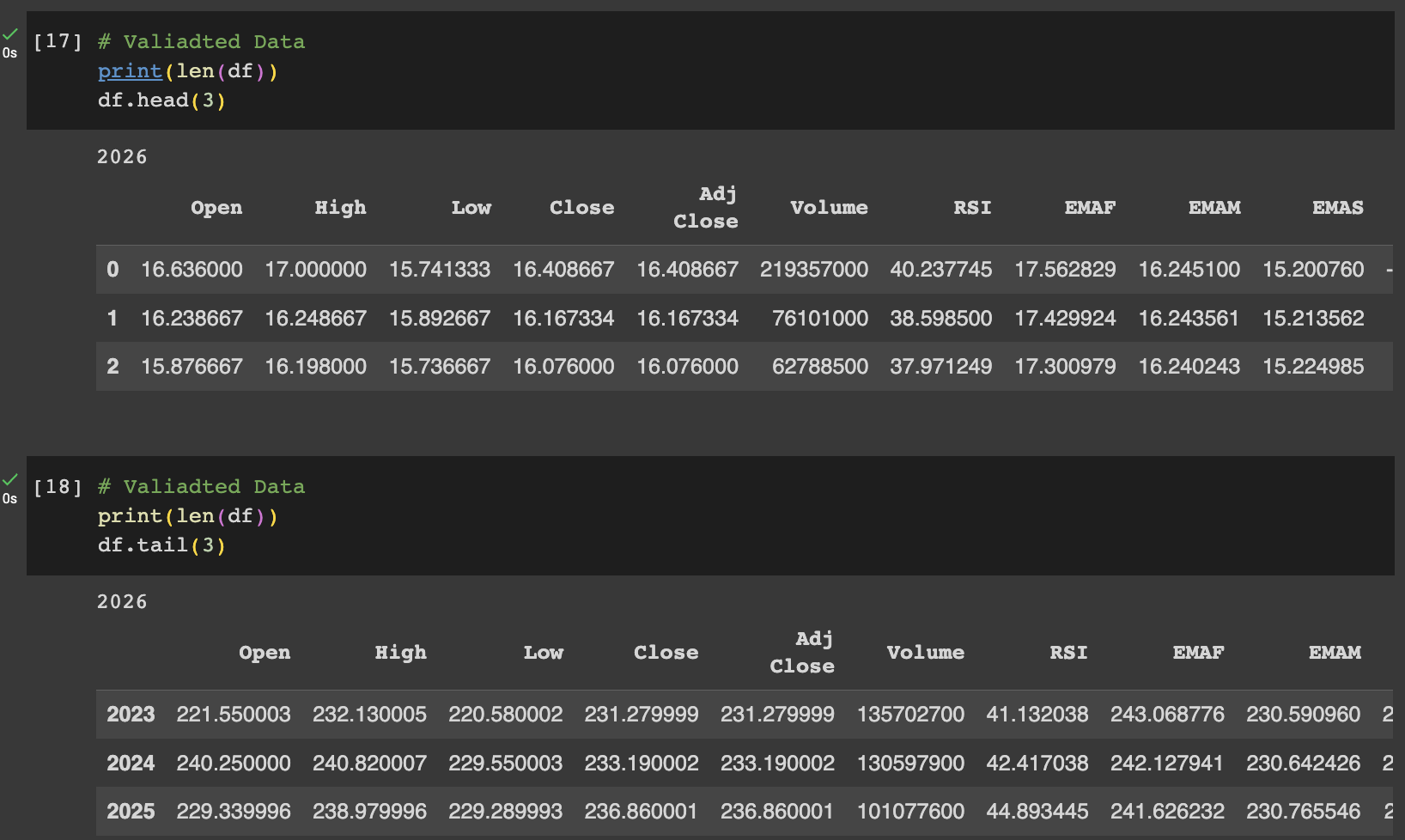
### 

### Raw Data



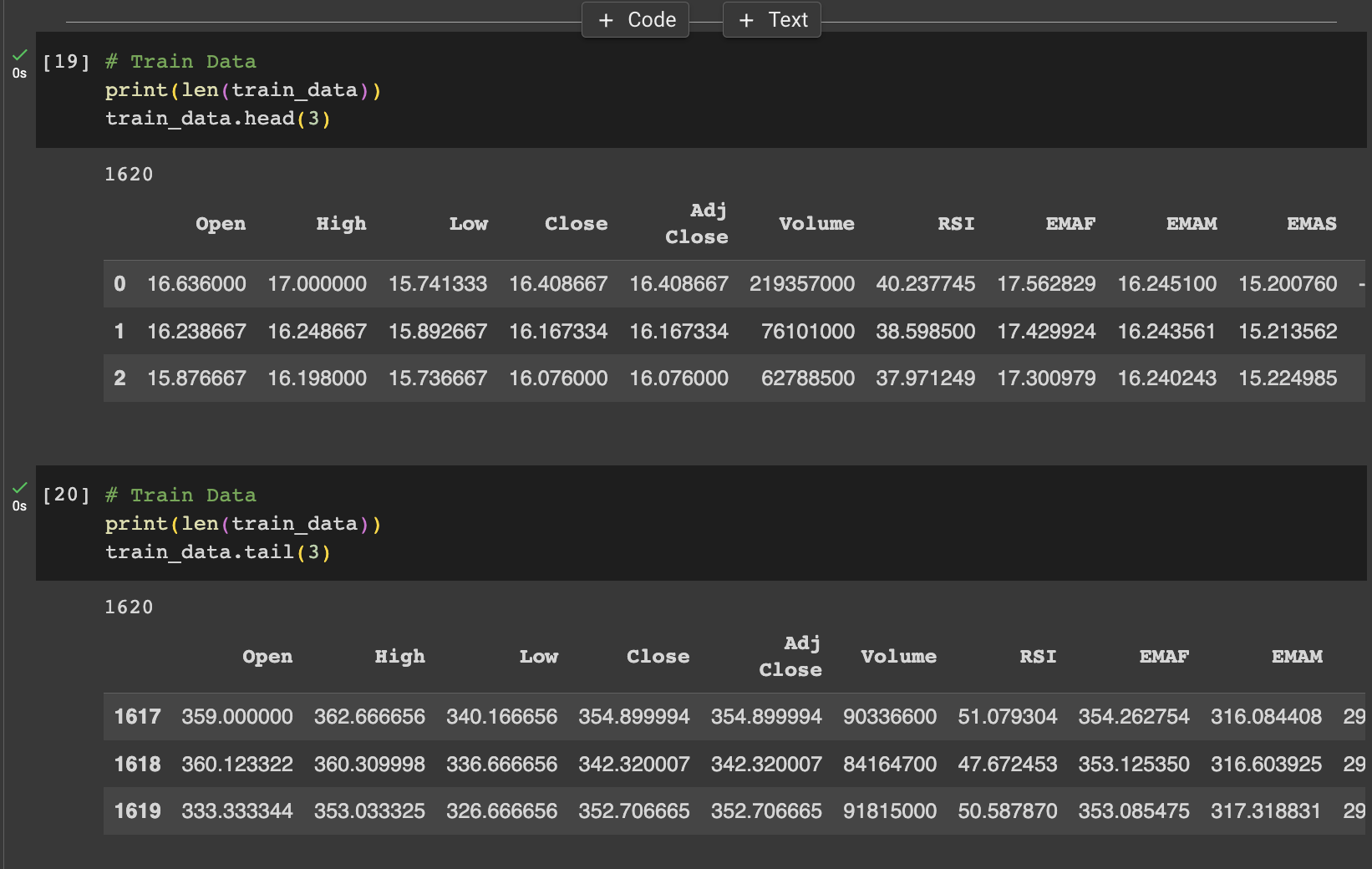
### 

### Validated Data



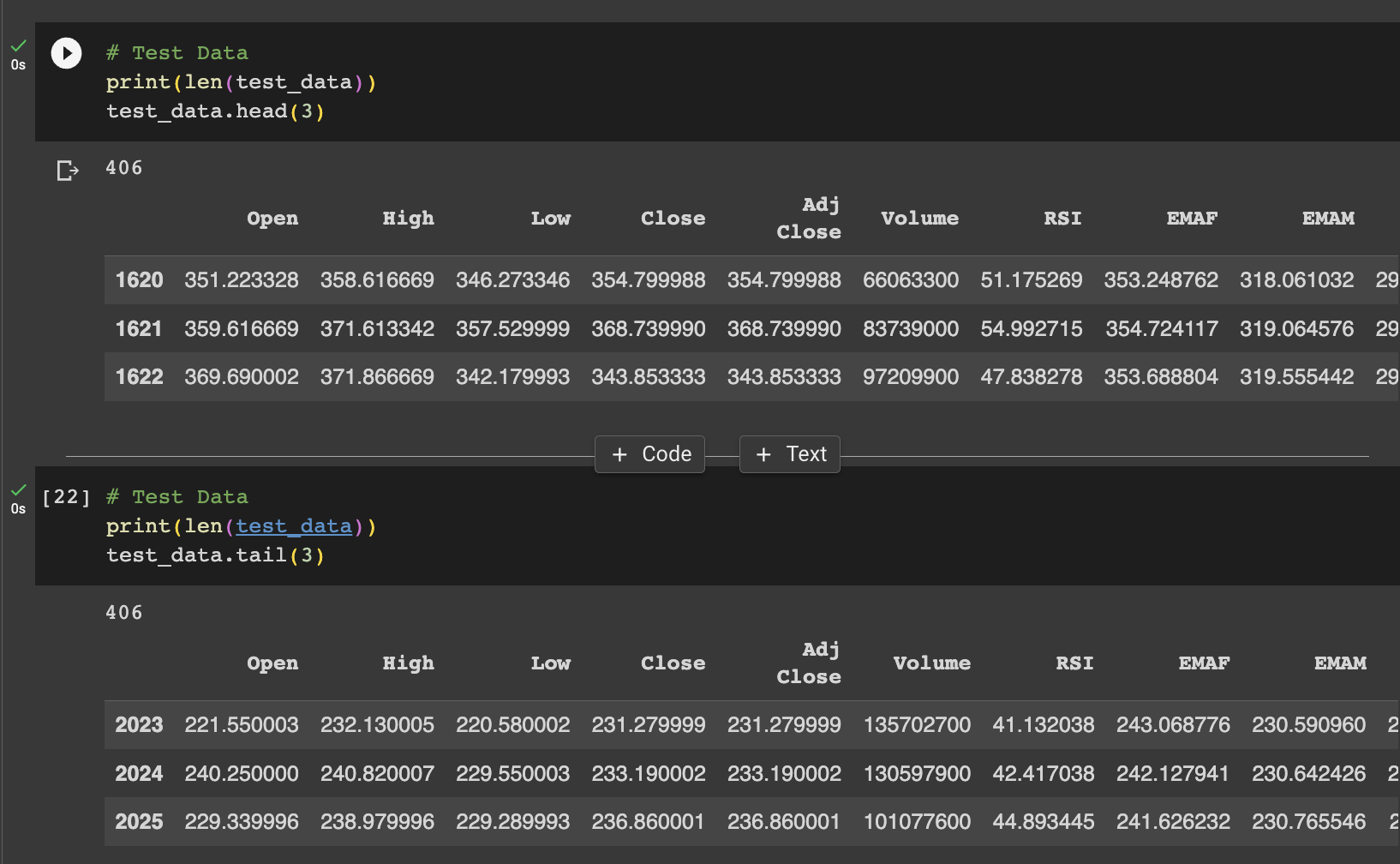
### 

### Train Data



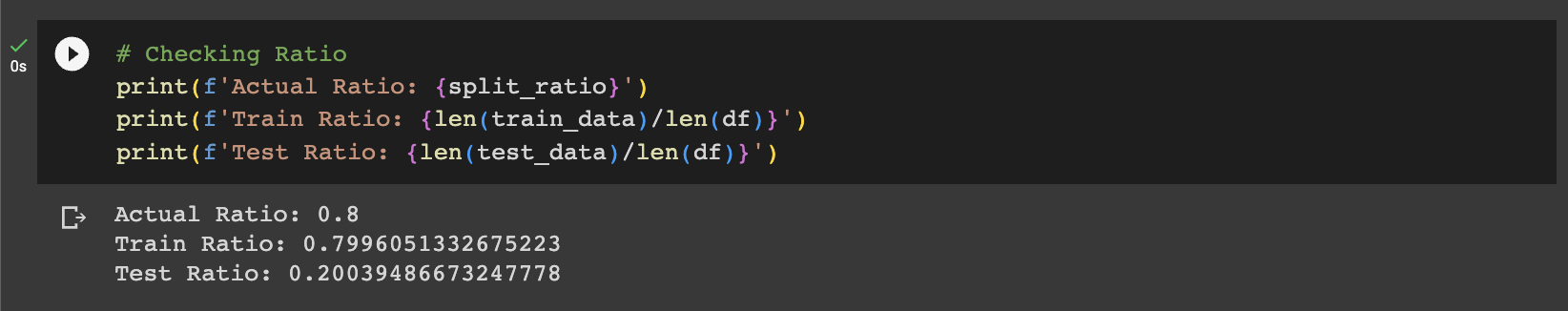
### 

### Test Data



### 

### Checking Ratio



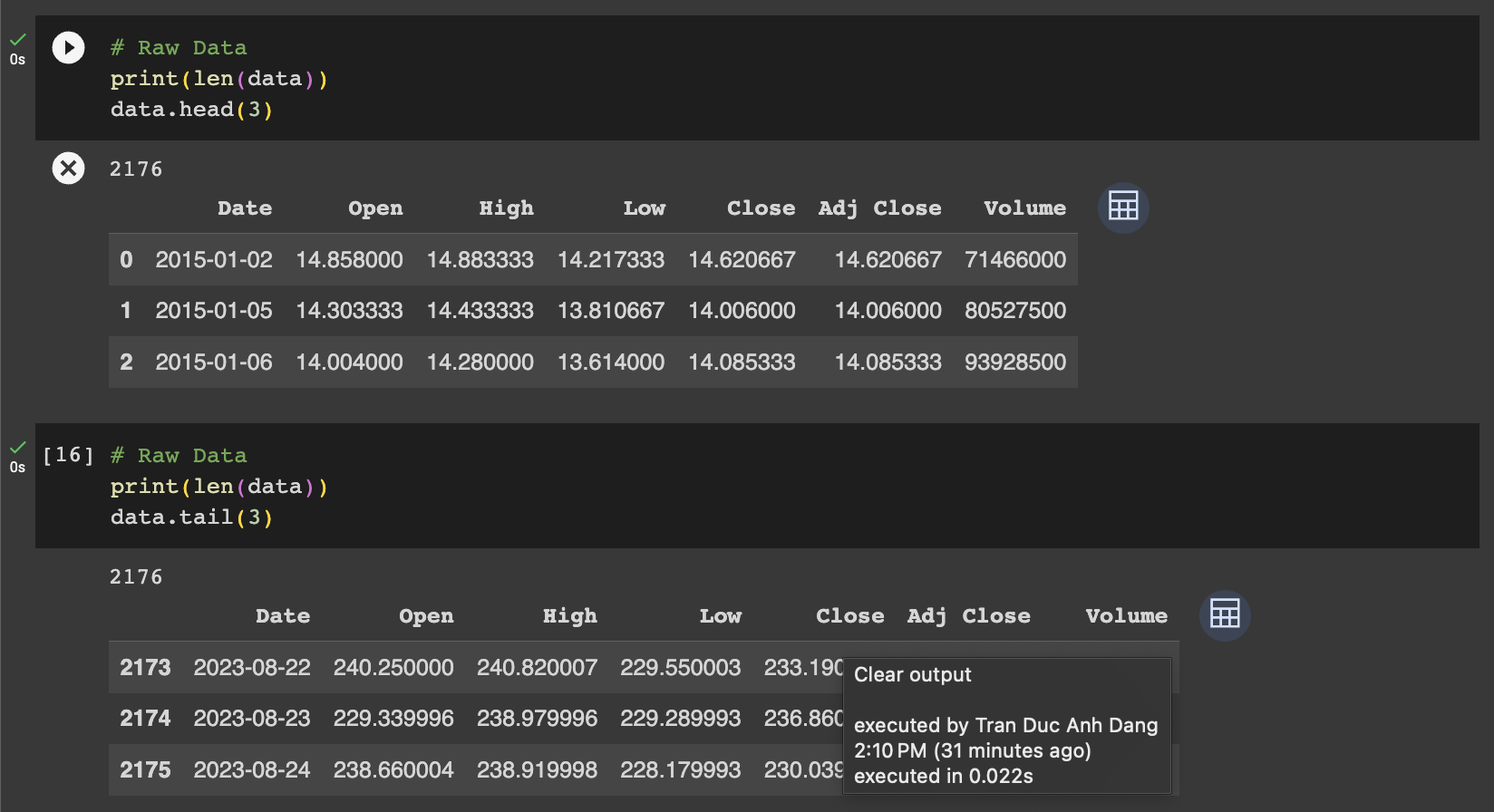
## 

## Scenario 2 - Loaded Data, split\_by\_date=True

### Checking Data Type

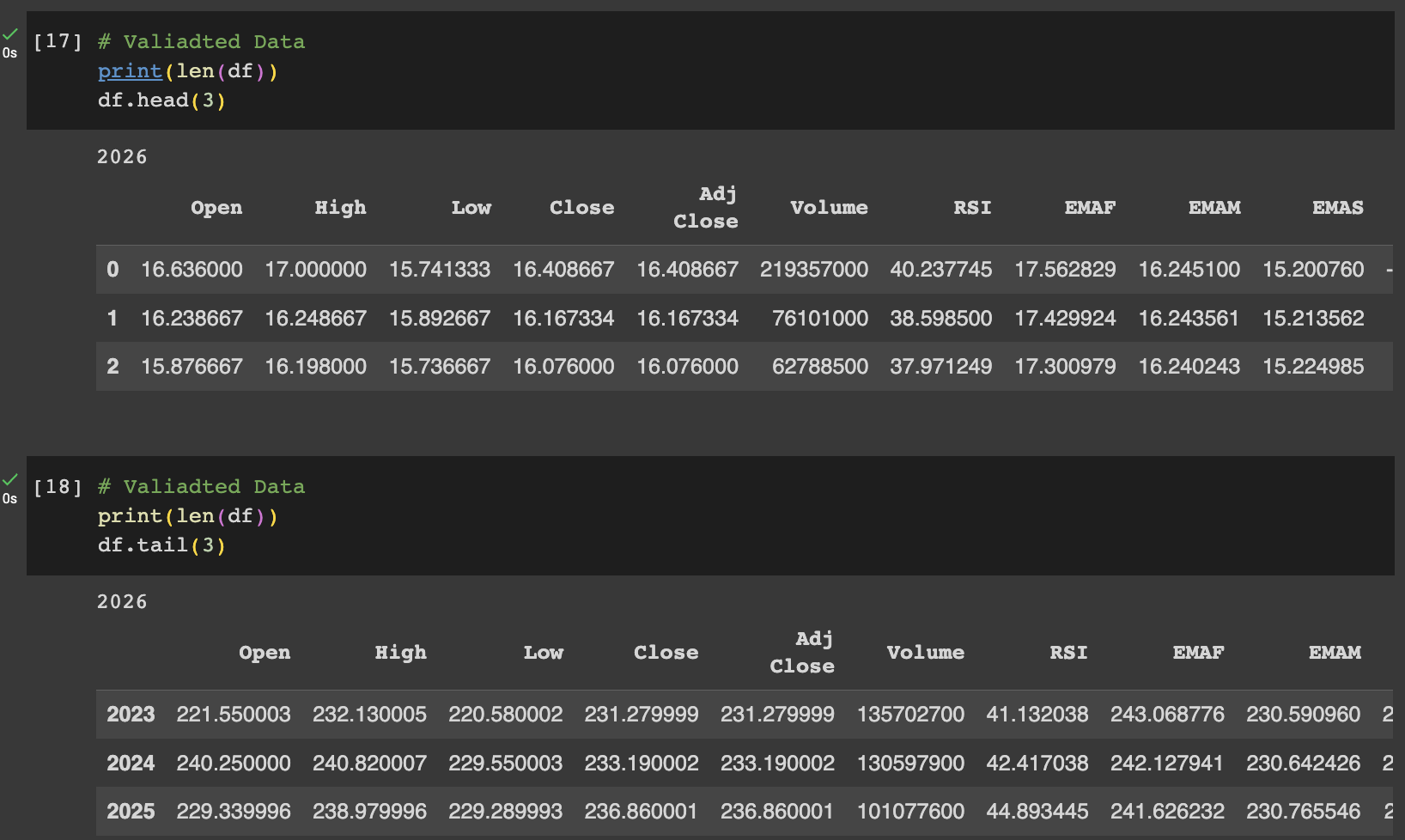
### 

### Raw Data



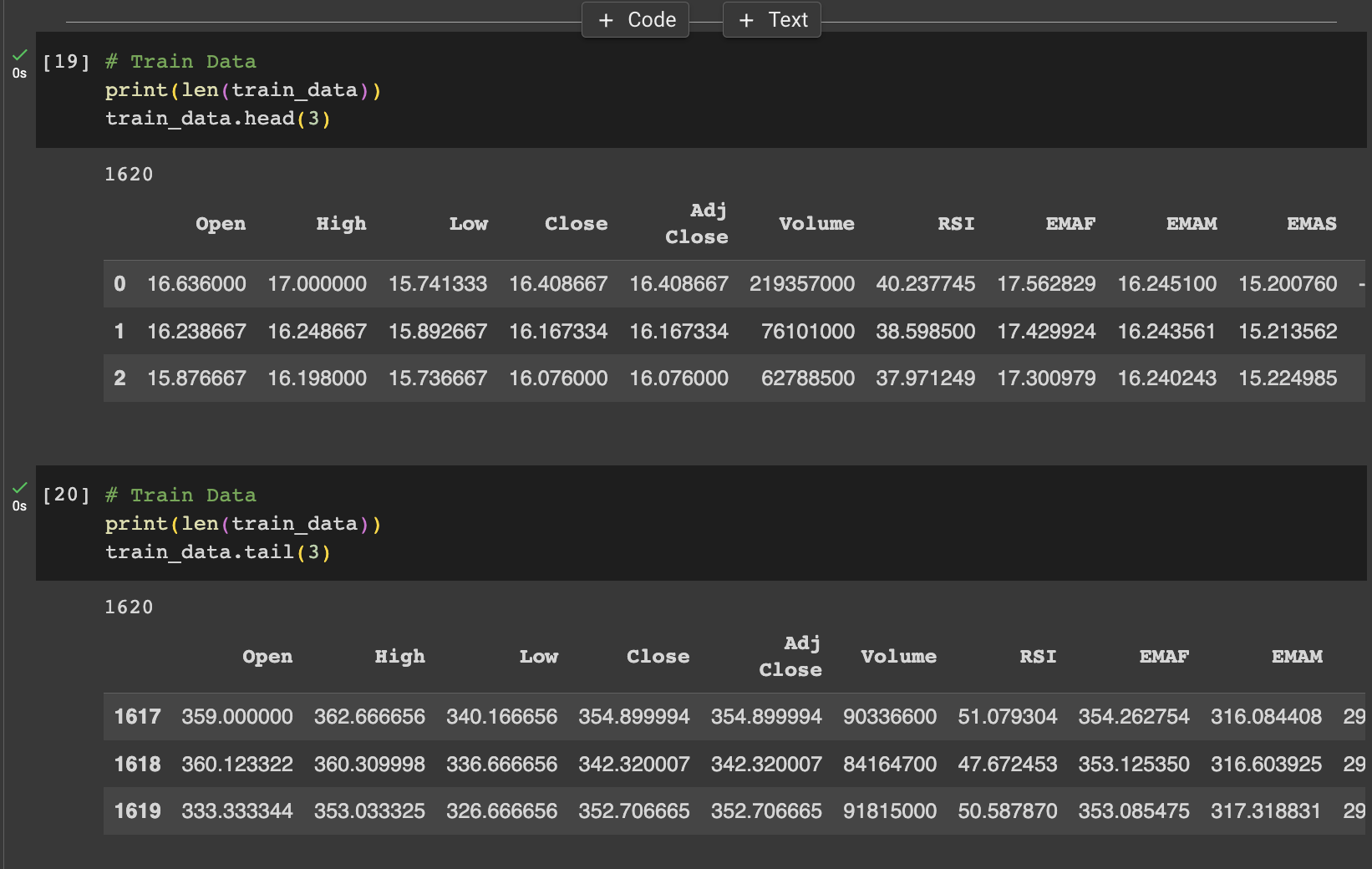
### 

### Validated Data



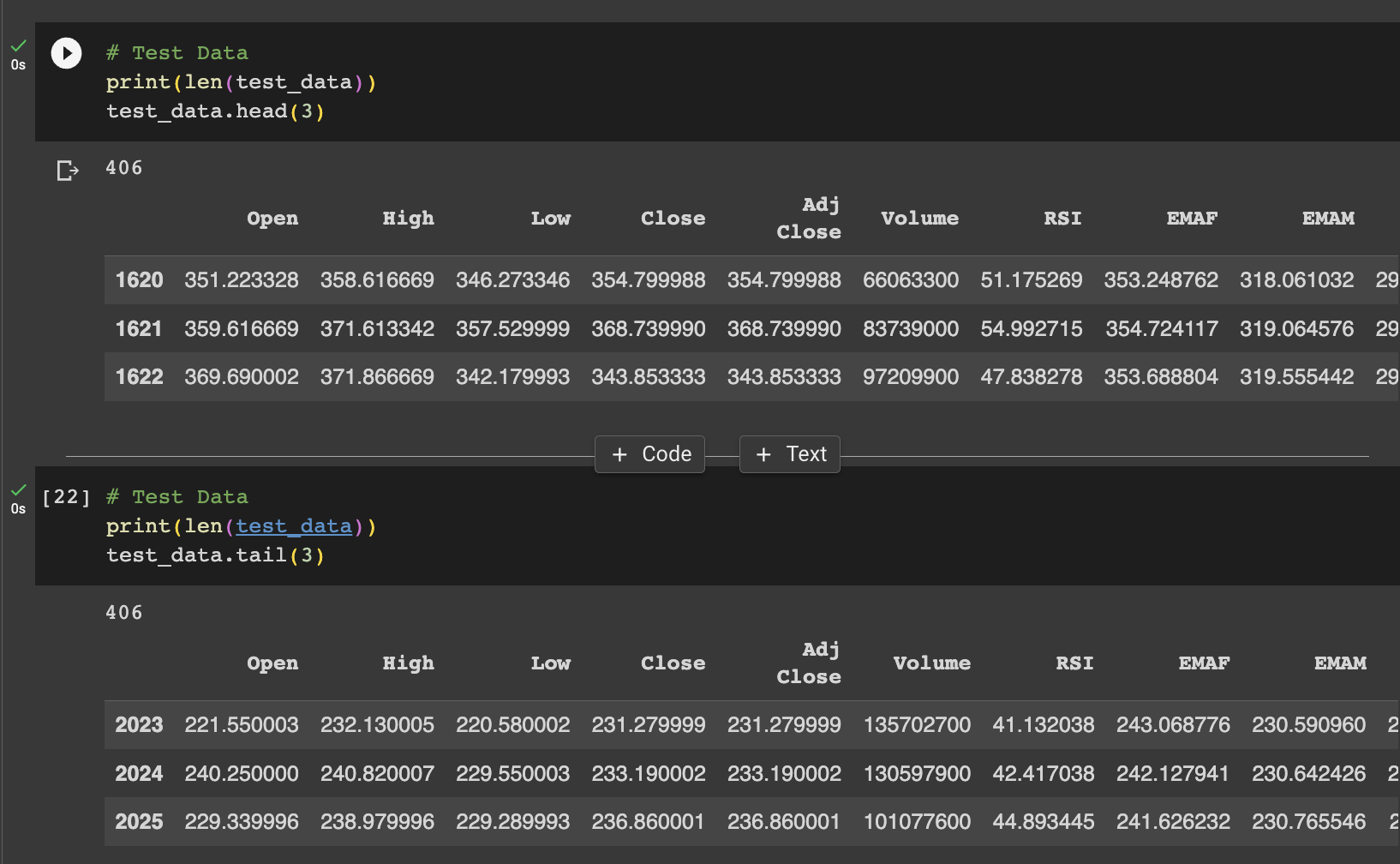
### 

### Train Data



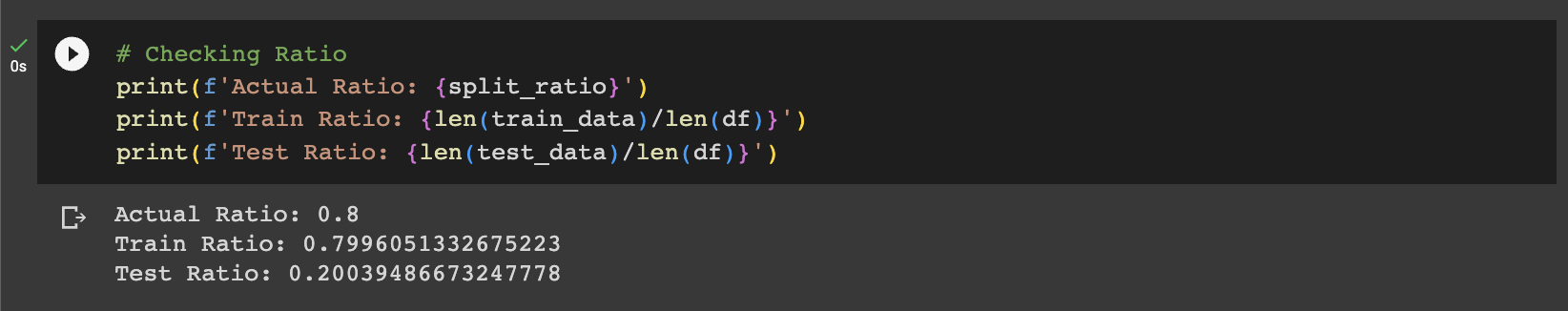
### 

### Test Data



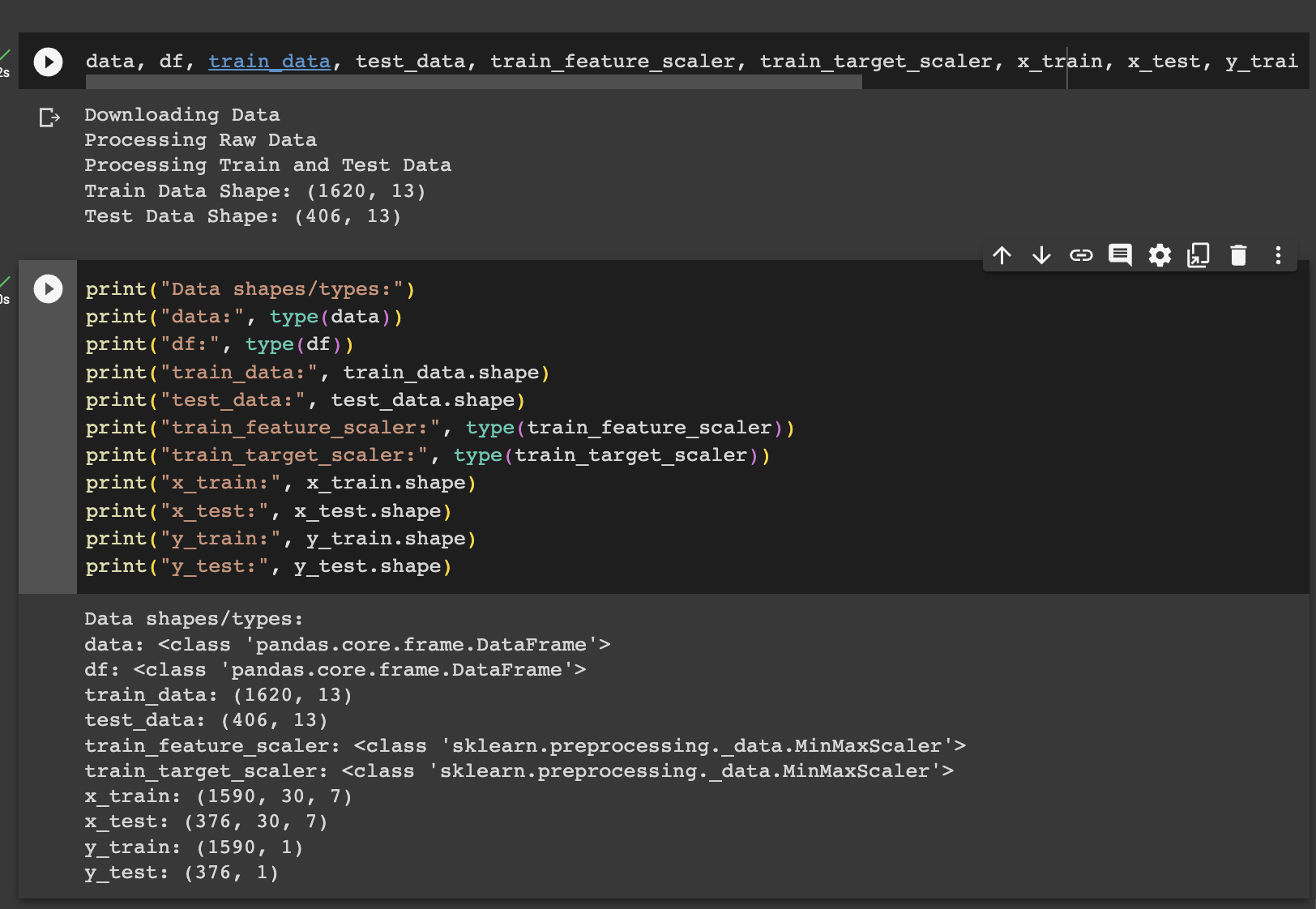
### 

### Checking Ratio



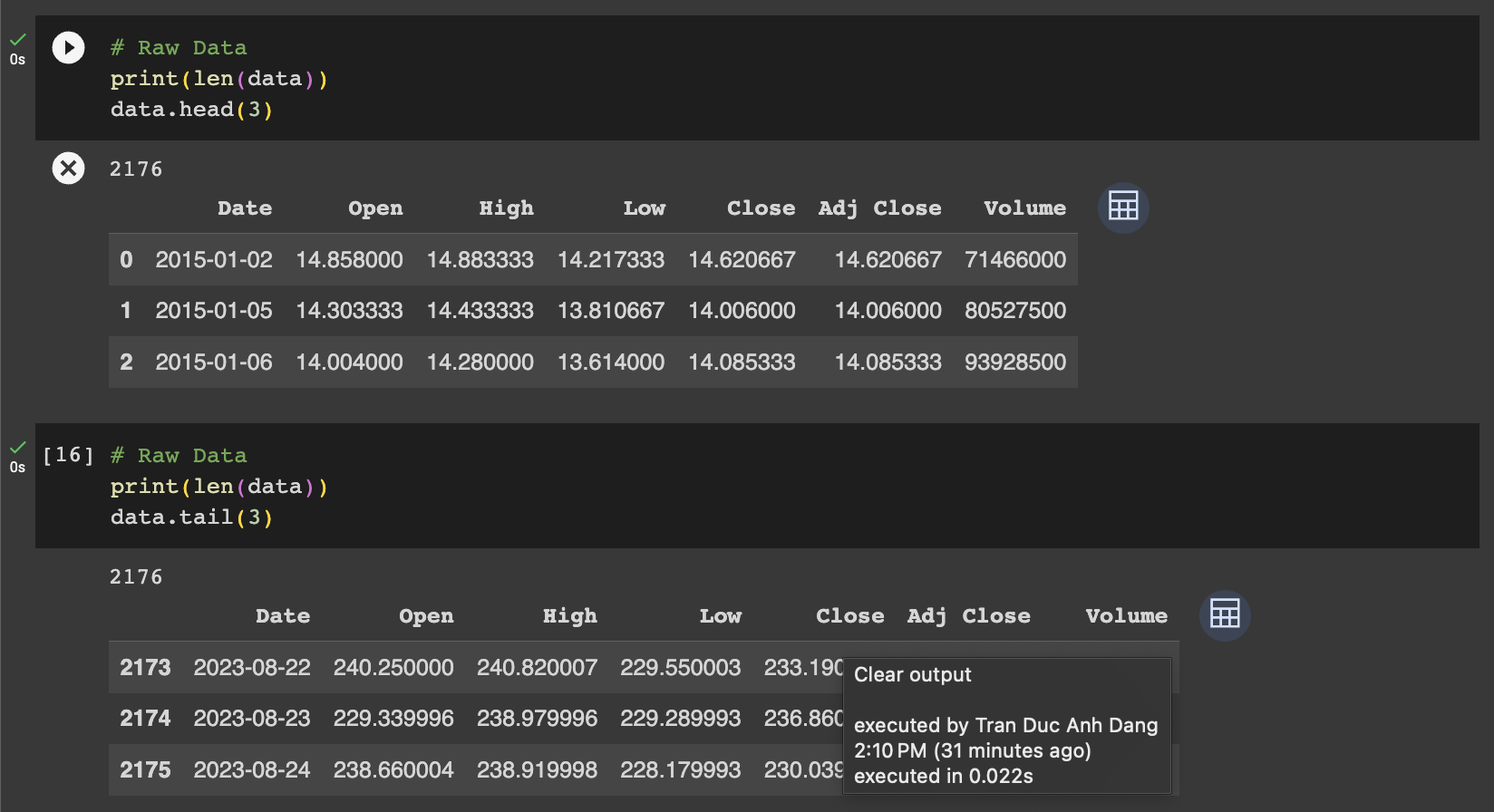
## Scenario 3 - First Time Running Output, split\_by\_date=False

### Checking Data Type



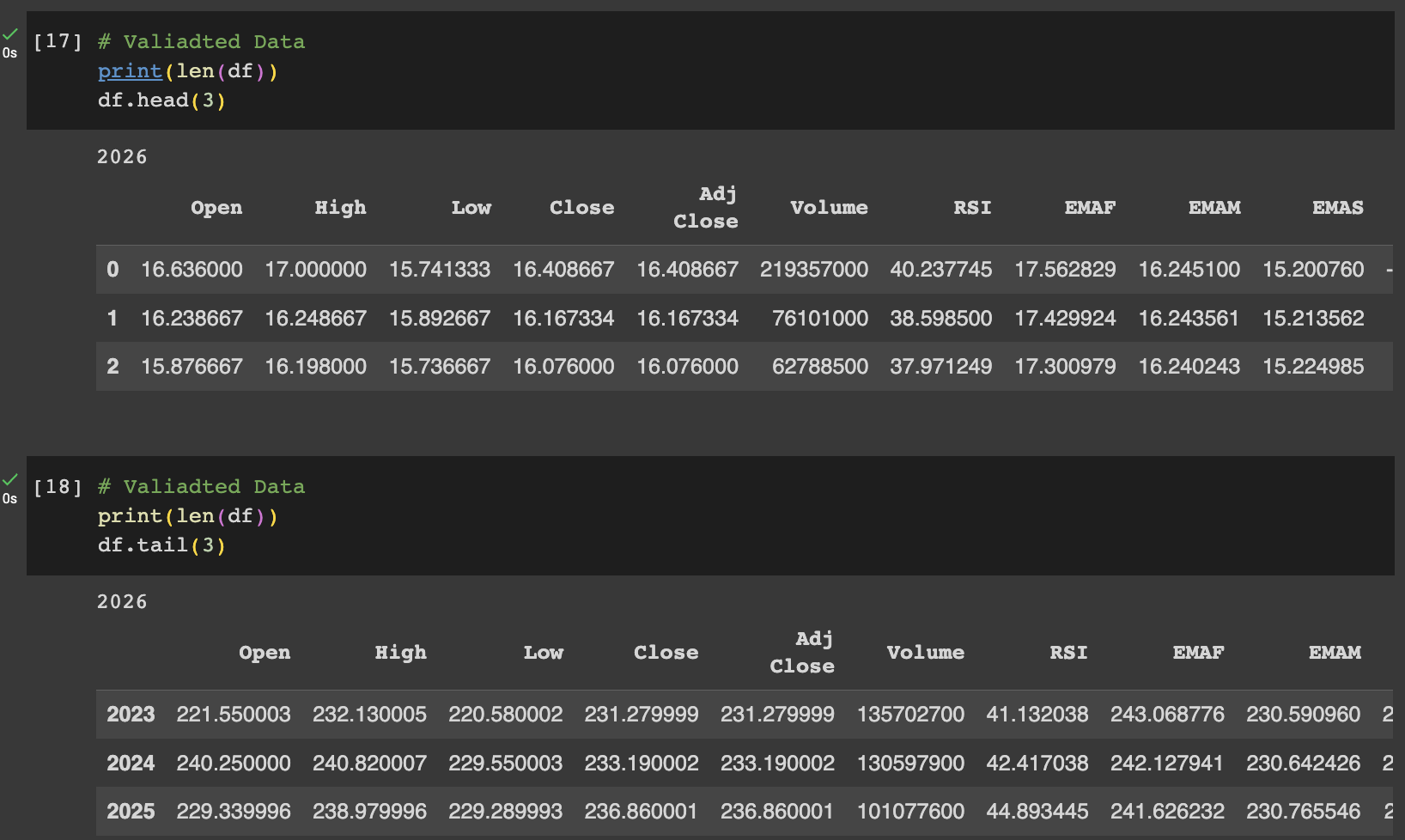
### 

### Raw Data



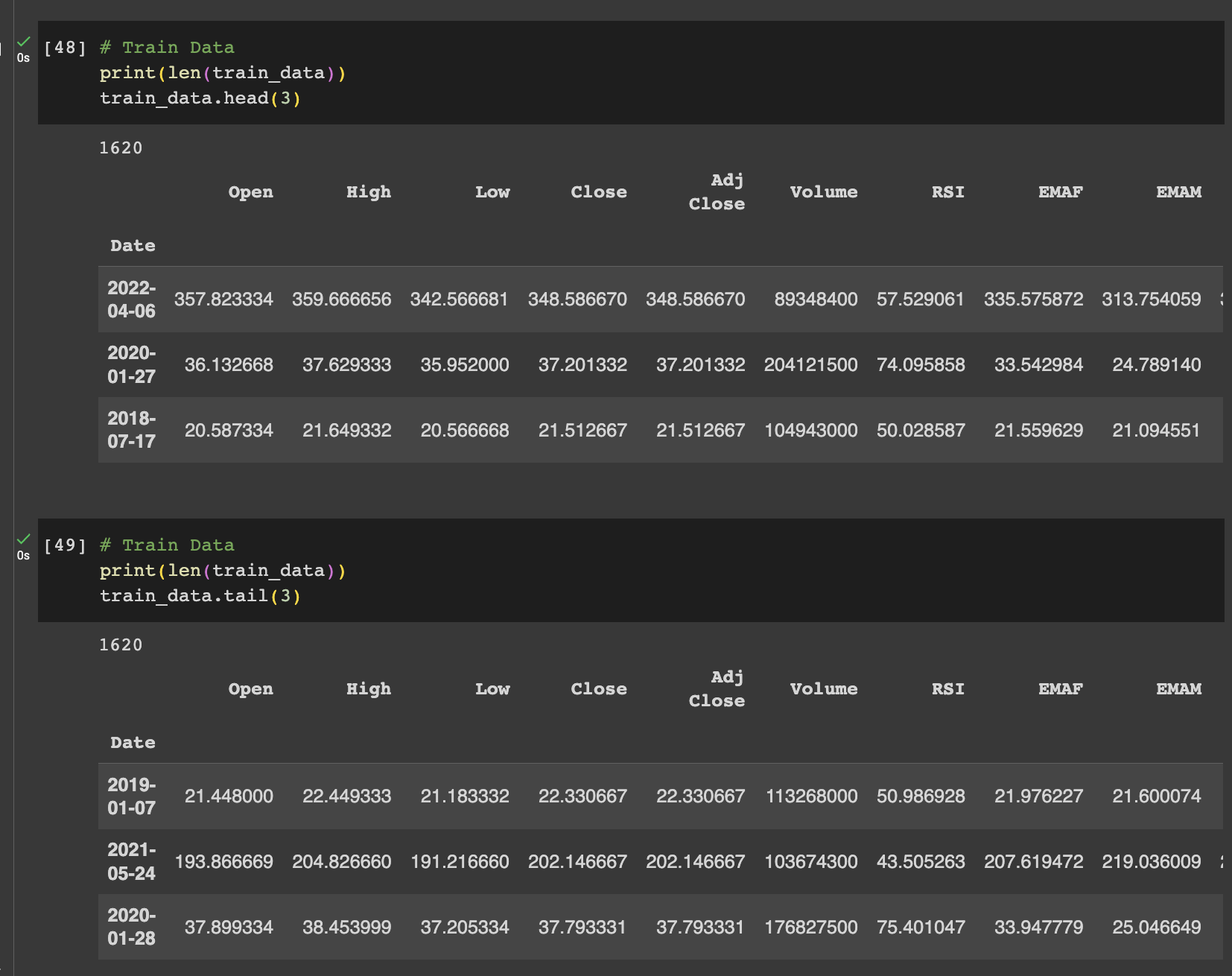
### 

### Validated Data



### 

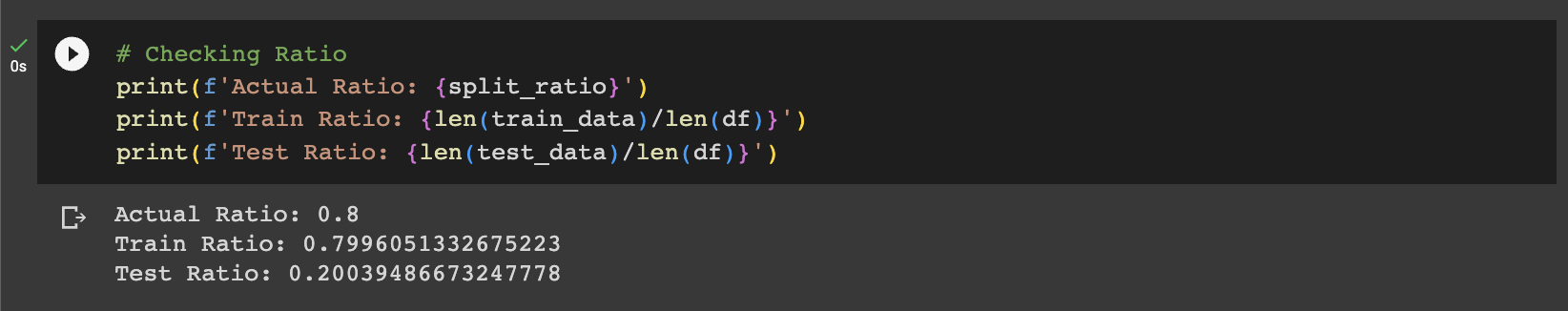
### Train Data



### 

### Test Data

### Checking Ratio



## 

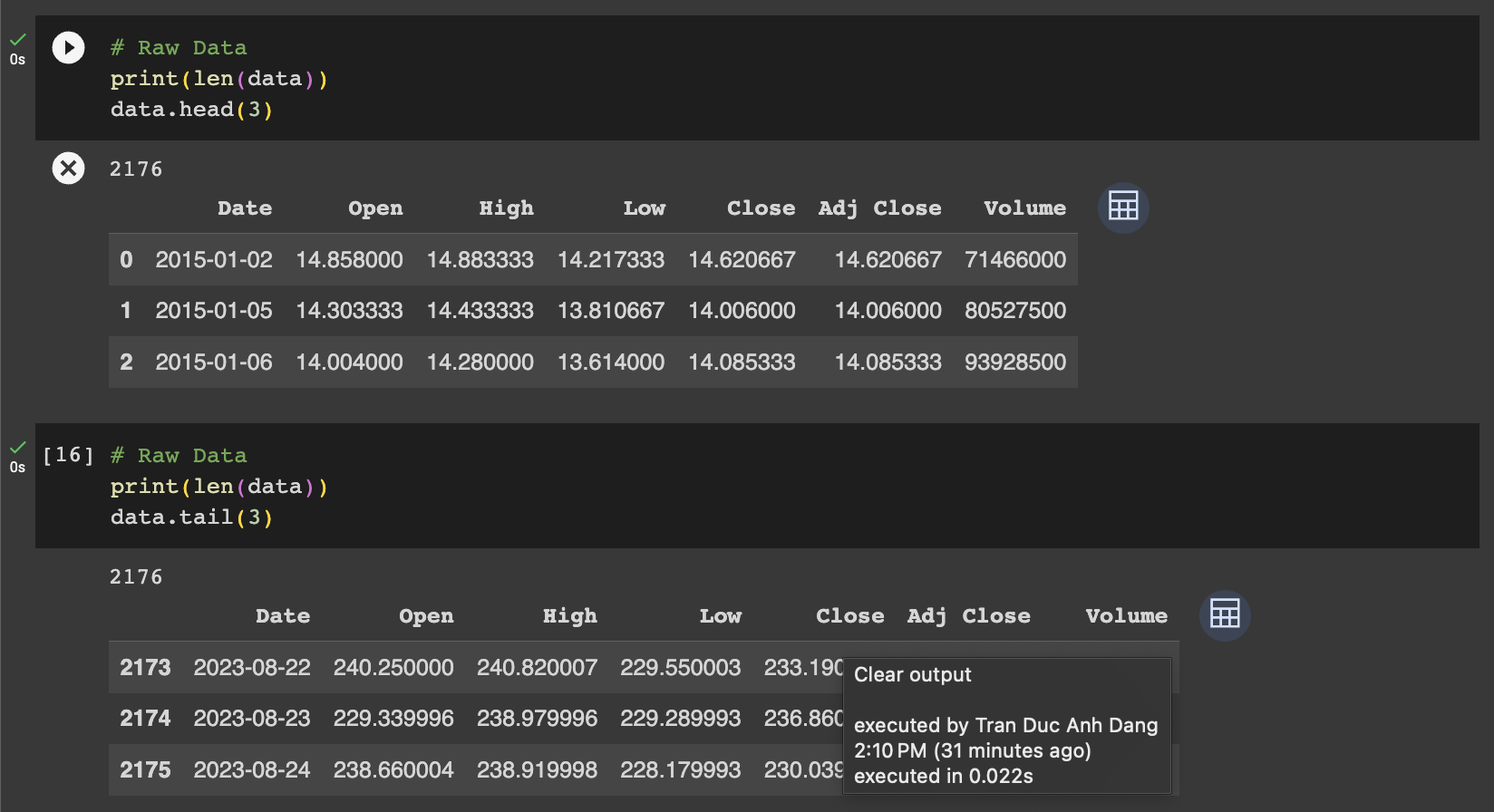
## 

## Scenario 4 - Loaded Data, split\_by\_date=False

### Checking Data Type

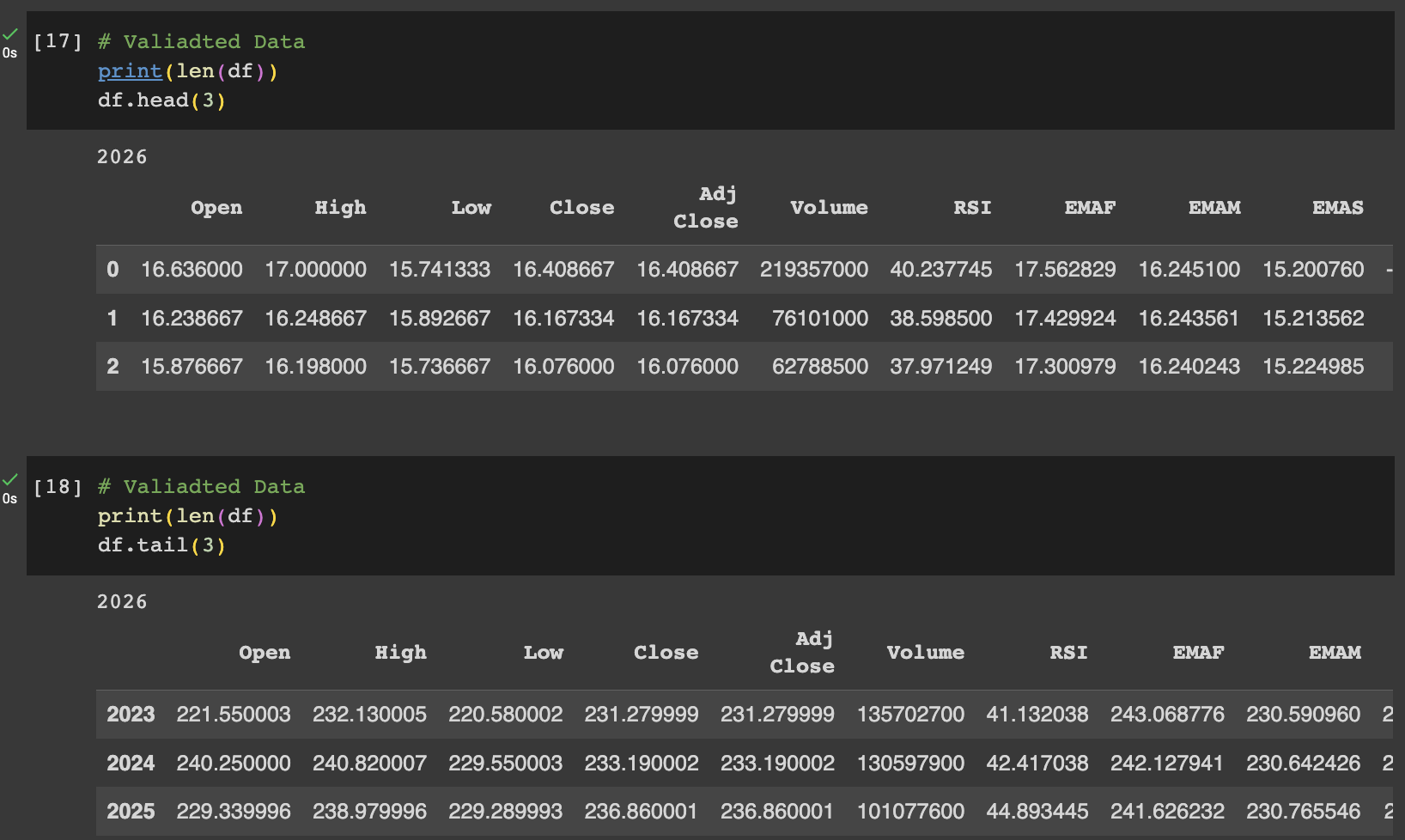
### 

### Raw Data



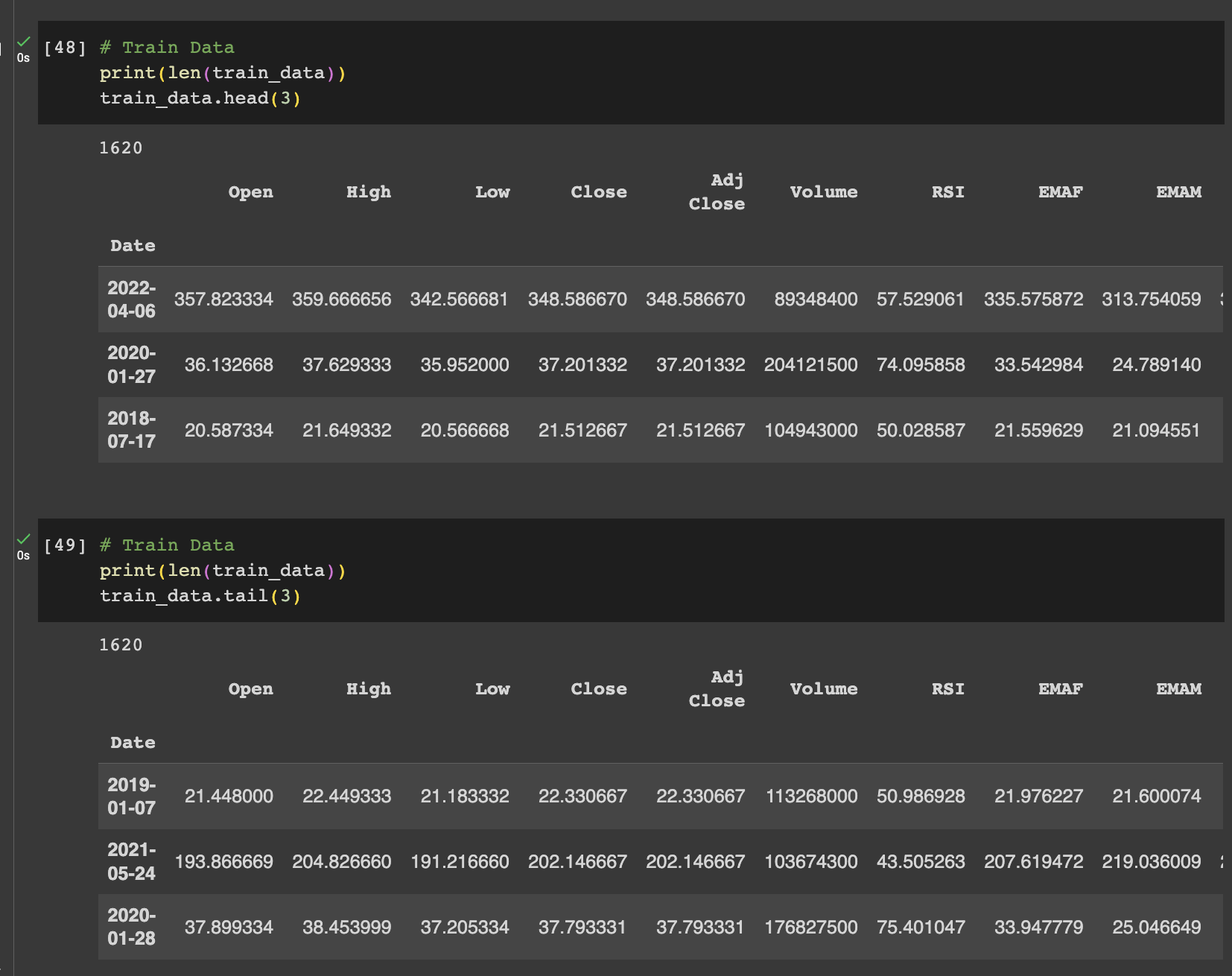
### 

### Validated Data



### 

### Train Data

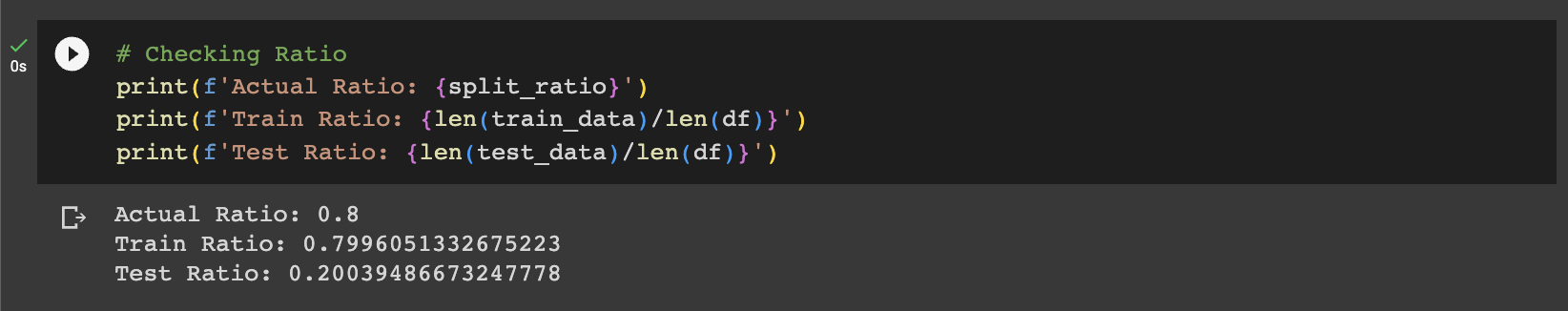


### 

### Test Data

### 

### Checking Ratio



## Testing Summary:

1. Training and Test Data: Prepared datasets were either loaded from existing files or freshly processed and then saved for future use.
2. Split by Date:
   1. Train Data Shape: (Shape when split by date)
   2. Test Data Shape: (Shape when split by date)
3. Random Split:
   1. Train Data Shape: (Shape when split randomly)
   2. Test Data Shape: (Shape when split randomly)
4. Feature and Target Scaling: Scalers were used to normalize the features and target columns. These scalers were saved for future use.
5. Prepared Sequences: For both training and test datasets, sequences of scaled features and target values were prepared and saved in .npz files.

Testing Conclusion:

The create\_datasets function successfully prepared the data for machine learning applications, with options for both chronological and random data splitting. The prepared data and associated utilities like scalers are saved for easy retrieval, making the pipeline efficient and robust.