Pandas for Data Analysis - Learning Path Part 1

December 13, 2024

1 TASK #1. DEFINE A PANDAS SERIES

[10]:
```
# Pandas is a data manipulation and analysis tool that is built on Numpy.
# Pandas uses a data structure known as DataFrame (think of it as Microsoft␣
 ↪excel in Python).
# DataFrames empower programmers to store and manipulate data in a tabular␣
 ↪fashion (rows and columns).
# Series Vs. DataFrame? Series is considered a single column of a DataFrame.

import pandas as pd
```

[11]:
```
# Let's define a Python list that contains 5 crypto currencies
crypto_list =["BTC", "XRP", "ADA", "ETH"]
crypto_list
```

[11]:
```
['BTC', 'XRP', 'ADA', 'ETH']
```

[12]:
```
# Let's confirm the Datatype
type(crypto_list)
```

[12]:
```
list
```

[13]:
```
# Let's create a one dimensional Pandas "series"
# Let's use Pandas Constructor Method to create a series from a Python list
# Note that series is formed of data and associated index (numeric index has␣
 ↪been automatically generated)
# Check Pandas Documentation for More information: https://pandas.pydata.org/
 ↪pandas-docs/stable/reference/api/pandas.Series.html#pandas.Series
# Object datatype is used for text data (String)
crypto_series = pd.Series(data = crypto_list)
crypto_series
```

[13]:
```
0    BTC
1    XRP
2    ADA
3    ETH
dtype: object
```

[14]:
```
# Let's confirm the Pandas Series Datatype
type(crypto_series)
```

[14]:
```
pandas.core.series.Series
```

[15]:
```
# Let's define another Pandas Series that contains numeric values (crypto␣
 ↪prices) instead of text data
# Note that we have int64 datatype which means it's integer stored in 64 bits␣
 ↪in memory
crypto_prices_series = pd.Series(data = [2000, 500, 2000, 20, 50])
crypto_prices_series
```

[15]:
```
0    2000
1     500
2    2000
3      20
4      50
dtype: int64
```

MINI CHALLENGE #1: - Define a Pandas Series named "my_series" that contains your top 3 favourite stocks. Confirm the datatype of "my_series"

[16]:
```
my_series = ["NVIDIA", "TESLA", "JIO"]
my_series = pd.Series(data = my_series)
my_series
```

[16]:
```
0    NVIDIA
1     TESLA
2       JIO
dtype: object
```

[17]:
```
type(my_series)
```

[17]:
```
pandas.core.series.Series
```

2 TASK #2. DEFINE A PANDAS SERIES WITH CUSTOM INDEX

[18]:
```
# Let's define a Python list that contains 5 Crypto currencies
crypto_list =['BTC', 'XRP', 'LTC', 'ADA', 'ETH']
crypto_list
```

[18]:
```
['BTC', 'XRP', 'LTC', 'ADA', 'ETH']
```

[19]:
```
# Let's define a python list as shown below. This python list will be used for␣
 ↪the Series index:
```

```
crypto_labels = ['crypto#1', 'crypto#2', 'crypto#3', 'crypto#4', 'crypto#5']
crypto_labels
```

[19]:
```
['crypto#1', 'crypto#2', 'crypto#3', 'crypto#4', 'crypto#5']
```

[20]:
```
# Let's create a one dimensional Pandas "series"
# Let's use Pandas Constructor Method to create a series from a Python list
# Note that this series is formed of data and associated labels
crypto_series = pd.Series(data = crypto_list, index = crypto_labels)
```

[21]:
```
# Let's view the series
crypto_series
```

[21]:
```
crypto#1    BTC
crypto#2    XRP
crypto#3    LTC
crypto#4    ADA
crypto#5    ETH
dtype: object
```

[22]:
```
# Let's obtain the datatype
type(crypto_series)
```

[22]:
```
pandas.core.series.Series
```

MINI CHALLENGE #2: - Define a Pandas Series named "my_series" that contains your top 3 favourite stocks. Instead of using default numeric indexes (similar to mini challenge #1), use the following indexes "stock #1", "stock #2", and "stock #3"

[23]:
```
my_series = ["NVIDIA", "TESLA", "JIO"]
my_labels = ['stock#1', 'stock#2', 'stock#3']
my_series = pd.Series(data = my_series, index = my_labels)
my_series
```

[23]:
```
stock#1    NVIDIA
stock#2     TESLA
stock#3       JIO
dtype: object
```

3 TASK #3. DEFINE A PANDAS SERIES FROM A DICTIONARY

[24]:
```
# A Dictionary consists of a collection of key-value pairs. Each key-value pair
↪maps the key to its corresponding value.
# Keys are unique within a dictionary while values may not be.
```

```
# List elements are accessed by their position in the list, via indexing while␣
 ↪Dictionary elements are accessed via keys
# Define a dictionary named "my_dict" using key-value pairs
my_dict = {'Employee ID': 1,
 'Employee Name': 'Steve',
 'Salary [$]': 2000,
 'Years with company': 10}
```

[25]:
```
# Show the dictionary
my_dict
```

[25]:
```
{'Employee ID': 1,
 'Employee Name': 'Steve',
 'Salary [$]': 2000,
 'Years with company': 10}
```

[26]:
```
# Confirm the dictionary datatype
type(my_dict)
```

[26]:
```
dict
```

[27]:
```
# Let's define a Pandas Series Using the dictionary
employee_series = pd.Series( data = my_dict )
employee_series
```

[27]:
```
Employee ID              1
Employee Name        Steve
Salary [$]            2000
Years with company      10
dtype: object
```

MINI CHALLENGE #3: - Create a Pandas Series from a dictionary with 3 of your favourite stocks and their corresponding prices

[28]:
```
stocks = {"NVIDIA" : 1000,
               "TESLA" : 800,
               "JIO" : 500}
print(stocks)
```

```
{'NVIDIA': 1000, 'TESLA': 800, 'JIO': 500}
```

[29]:
```
my_series = pd.Series(stocks)
my_series
```

[29]:
```
NVIDIA    1000
TESLA      800
JIO        500
```

```
dtype: int64
```

4 TASK #4. PANDAS ATTRIBUTES

[30]:
```
# Attributes/Properties: do not use parantheses "()" and are used to get Pandas Series Properties. Ex: my_series.values, my_series.shape
# Methods: use parantheses "()" and might include arguments and they actually alter/change the Pandas Series. Ex: my_series.tail(), my_series.head(), my_series.drop_duplicates()
# Indexers: use square brackets "[]" and are used to access specific elements in a Pandas Series or DataFrame. Ex: my_series.loc[], my_series.iloc[]

# Let's redefine a Pandas Series containing our favourite 5 cryptos
crypto_list = ['BTC', 'XRP', 'LTC', 'ADA', 'ETH']
crypto_series = pd.Series(data = crypto_list)
crypto_series
```

[30]:
```
0    BTC
1    XRP
2    LTC
3    ADA
4    ETH
dtype: object
```

[31]:
```
# ".Values" attribute is used to return Series as ndarray depending on its dtype
# Check this for more information: https://pandas.pydata.org/docs/reference/api/pandas.Series.values.html#pandas.Series.values
crypto_series.values
```

[31]:
```
array(['BTC', 'XRP', 'LTC', 'ADA', 'ETH'], dtype=object)
```

[32]:
```
# index is used to return the index (axis labels) of the Series
crypto_series.index
```

[32]:
```
RangeIndex(start=0, stop=5, step=1)
```

[33]:
```
# dtype is used to return the datatype of the Series ('O' stands for 'object' datatype)
crypto_series.dtype
```

[33]:
```
dtype('O')
```

[34]:
```
# Check if all elements are unique or not
crypto_series.is_unique
```

[34]:
```
True
```

[35]:
```
# Check the shape of the Series
# note that a Series is one dimensional
crypto_series.shape
```

[35]:
```
(5,)
```

MINI CHALLENGE #4: - What is the size of the Pandas Series? (External Research for the proper attribute is Required)

[36]:
```
crypto_series.size
```

[36]:
```
5
```

5 TASK #5. PANDAS METHODS

[37]:
```
# Methods have parentheses and they actually alter/change the Pandas Series
# Methods: use parantheses "()" and might include arguments. Ex: my_series.
 ↪tail(), my_series.head(), my_series.drop_duplicates()

# Let's define another Pandas Series that contains numeric values (crypto␣
 ↪prices) instead of text data
# Note that we have int64 datatype which means it contains integer values␣
 ↪stored in 64 bits in memory
crypto_prices = pd.Series(data = [400, 500, 1500, 20, 70])
crypto_prices
```

[37]:
```
0     400
1     500
2    1500
3      20
4      70
dtype: int64
```

[38]:
```
# Let's obtain the sum of all elements in the Pandas Series
crypto_prices.sum()
```

[38]:
```
2490
```

[39]:
```
# Let's obtain the multiplication of all elements in the Pandas Series
crypto_prices.product()
```

[39]:
```
420000000000
```

[42]:
```
# Let's obtain the average
crypto_prices.mean()
```

[42]:
```
498.0
```

[43]:
```
# Let's show the first couple of elements in the Pandas Series
crypto_prices.head(2)
```

[43]:
```
0    400
1    500
dtype: int64
```

[44]:
```
# Note that head creates a new dataframe
new_crypto_prices = crypto_prices.head(3)
new_crypto_prices
```

[44]:
```
0     400
1     500
2    1500
dtype: int64
```

MINI CHALLENGE #5: - Show the last 2 rows in the Pandas Series (External Research is Required) - How many bytes does this Pandas Series consume in memory? (External Research is Required)

[45]:
```
crypto_prices.tail(2)
```

[45]:
```
3    20
4    70
dtype: int64
```

[46]:
```
crypto_prices.memory_usage()
```

[46]:
```
168
```

6 TASK #6. IMPORT CSV DATA (1-D) USING PANDAS

[47]:
```
# Pandas read_csv is used to read a csv file and store data in a DataFrame by␣
 ↪default (DataFrames will be covered shortly!)
# Use Squeeze to convert it into a Pandas Series (One-dimensional)
# Notice that no foramtting exists when a Series is plotted
BTC_price_series = pd.read_csv("crypto.csv", squeeze = True)
```

[48]:
```
BTC_price_series
```

[48]:
```
0         457.334015
1         424.440002
2         394.795990
3         408.903992
```

```
4          398.821014
              …
2380     55950.746090
2381     57750.199220
2382     58917.691410
2383     58918.832030
2384     59095.808590
Name: BTC-USD Price, Length: 2385, dtype: float64
```

[49]:
```
type(BTC_price_series)
```

[49]:
```
pandas.core.series.Series
```

MINI CHALLENGE #6: - Set Squeeze = False and rerun the cell, what do you notice? Use Type to compare both outputs

[50]:
```
BTC_price_series = pd.read_csv("crypto.csv", squeeze = False)
BTC_price_series
```

[50]:
```
       BTC-USD Price
0         457.334015
1         424.440002
2         394.795990
3         408.903992
4         398.821014
…                …
2380    55950.746090
2381    57750.199220
2382    58917.691410
2383    58918.832030
2384    59095.808590

[2385 rows x 1 columns]
```

[51]:
```
type(BTC_price_series)
```

[51]:
```
pandas.core.frame.DataFrame
```

7 TASK #7. PANDAS BUILT-IN FUNCTIONS

[52]:
```
# Pandas works great with pre-existing python functions
# You don't have to play with pandas methods and directly leverage Python␣
 ↪functions
# Check Python built-in functions here: https://docs.python.org/3/library/
 ↪functions.html
BTC_price_series = pd.read_csv("crypto.csv", squeeze = True)
```

```
BTC_price_series
```

[52]:
```
0          457.334015
1          424.440002
2          394.795990
3          408.903992
4          398.821014
             …
2380     55950.746090
2381     57750.199220
2382     58917.691410
2383     58918.832030
2384     59095.808590
Name: BTC-USD Price, Length: 2385, dtype: float64
```

[53]:
```
# Obtain the Data Type of the Pandas Series
type(BTC_price_series)
```

[53]:
```
pandas.core.series.Series
```

[55]:
```
# Obtain the length of the Pandas Series
len(BTC_price_series)
```

[55]:
```
2385
```

[56]:
```
# Obtain the maximum value of the Pandas Series
max(BTC_price_series)
```

[56]:
```
61243.08594
```

[57]:
```
# Obtain the minimum value of the Pandas Series
min(BTC_price_series)
```

[57]:
```
178.1029968
```

MINI CHALLENGE #7: - Given the following Pandas Series, convert all positive values to negative using python built-in functions - Obtain only unique values (ie: Remove duplicates) using python built-in functions - my_series = pd.Series(data = [-10, 100, -30, 50, 100])

[58]:
```
my_series = pd.Series(data = [-10, 100, -30, 50, 100])
my_series
```

[58]:
```
0    -10
1    100
2    -30
3     50
```

```
4    100
dtype: int64
```

[62]:
```
abs(my_series)
```

[62]:
```
0     10
1    100
2     30
3     50
4    100
dtype: int64
```

[63]:
```
set(my_series)
```

[63]:
```
{-30, -10, 50, 100}
```

8 TASK #8. SORTING PANDAS SERIES

[64]:
```
# Let's import CSV data as follows:
BTC_price_series = pd.read_csv("crypto.csv", squeeze = True)
BTC_price_series
```

[64]:
```
0         457.334015
1         424.440002
2         394.795990
3         408.903992
4         398.821014
             …
2380    55950.746090
2381    57750.199220
2382    58917.691410
2383    58918.832030
2384    59095.808590
Name: BTC-USD Price, Length: 2385, dtype: float64
```

[65]:
```
# You can sort the values in the dataframe as follows
BTC_price_series.sort_values()
```

[65]:
```
119       178.102997
122       199.259995
121       208.097000
120       209.843994
123       210.339004
             …
2382    58917.691410
2383    58918.832030
```

```
2384    59095.808590
2366    59302.316410
2365    61243.085940
Name: BTC-USD Price, Length: 2385, dtype: float64
```

[66]:
```
# Let's view Pandas Series again after sorting, Note that nothing changed in␣
 ↪memory! you have to make sure that inplace is set to True
BTC_price_series
```

[66]:
```
0         457.334015
1         424.440002
2         394.795990
3         408.903992
4         398.821014
             …
2380    55950.746090
2381    57750.199220
2382    58917.691410
2383    58918.832030
2384    59095.808590
Name: BTC-USD Price, Length: 2385, dtype: float64
```

[69]:
```
# Set inplace = True to ensure that change has taken place in memory
BTC_price_series.sort_values(inplace = True)
```

[70]:
```
# Note that now the change (ordering) took place
BTC_price_series
```

[70]:
```
119       178.102997
122       199.259995
121       208.097000
120       209.843994
123       210.339004
             …
2382    58917.691410
2383    58918.832030
2384    59095.808590
2366    59302.316410
2365    61243.085940
Name: BTC-USD Price, Length: 2385, dtype: float64
```

[73]:
```
# Notice that the indexes are now changed
# You can also sort by index (revert back to the original Pandas Series) as␣
 ↪follows:
BTC_price_series.sort_index(inplace = True)
BTC_price_series
```

[73]:
```
0          457.334015
1          424.440002
2          394.795990
3          408.903992
4          398.821014
             …
2380     55950.746090
2381     57750.199220
2382     58917.691410
2383     58918.832030
2384     59095.808590
Name: BTC-USD Price, Length: 2385, dtype: float64
```

MINI CHALLENGE #8: - Sort the BTC_price_series values in a decending order instead. Make sure to update values in-memory.

[74]:
```
BTC_price_series.sort_values(ascending = False, inplace = True)
BTC_price_series
```

[74]:
```
2365     61243.085940
2366     59302.316410
2384     59095.808590
2383     58918.832030
2382     58917.691410
             …
123        210.339004
120        209.843994
121        208.097000
122        199.259995
119        178.102997
Name: BTC-USD Price, Length: 2385, dtype: float64
```

9 TASK #9. PERFORM MATH OPERATIONS ON PANDAS SERIES

[75]:
```
# Let's import CSV data as follows:
BTC_price_series = pd.read_csv("crypto.csv", squeeze = True)
BTC_price_series
```

[75]:
```
0          457.334015
1          424.440002
2          394.795990
3          408.903992
4          398.821014
             …
2380     55950.746090
```

```
2381    57750.199220
2382    58917.691410
2383    58918.832030
2384    59095.808590
Name: BTC-USD Price, Length: 2385, dtype: float64
```

[77]:
```
# Apply Sum Method on Pandas Series
BTC_price_series.sum()
```

[77]:
```
15435379.738852698
```

[78]:
```
# Apply count Method on Pandas Series
BTC_price_series.count()
```

[78]:
```
2385
```

[79]:
```
# Obtain the maximum value
BTC_price_series.max()
```

[79]:
```
61243.08594
```

[80]:
```
# Obtain the minimum value
BTC_price_series.min()
```

[80]:
```
178.1029968
```

[81]:
```
# My favourite: Describe!
# Describe is used to obtain all statistical information in one place
BTC_price_series.describe()
```

[81]:
```
count     2385.000000
mean      6471.857333
std       9289.022505
min        178.102997
25%        454.618988
50%       4076.632568
75%       8864.766602
max      61243.085940
Name: BTC-USD Price, dtype: float64
```

MINI CHALLENGE #9: - Obtain the average price of the BTC_price_series using two different methods

[82]:
```
BTC_price_series.mean()
```

[82]:
```
6471.857332852284
```

[83]:
```
BTC_price_series.sum()/BTC_price_series.count()
```

[83]:
```
6471.857332852284
```

10 TASK #10. CHECK IF A GIVEN ELEMENT EXISTS IN A PANDAS SERIES

[84]:
```
# Let's import CSV data as follows:
BTC_price_series = pd.read_csv("crypto.csv", squeeze = True)
BTC_price_series
```

[84]:
```
0         457.334015
1         424.440002
2         394.795990
3         408.903992
4         398.821014
            …
2380    55950.746090
2381    57750.199220
2382    58917.691410
2383    58918.832030
2384    59095.808590
Name: BTC-USD Price, Length: 2385, dtype: float64
```

[87]:
```
# Check if a given number exists in a Pandas Series values
# Returns a boolean "True" or "False"
408.8 in BTC_price_series.values
```

[87]:
```
False
```

[88]:
```
# Check if a given number exists in a Pandas Series index
1295 in BTC_price_series.index
```

[88]:
```
True
```

[89]:
```
# Note that by default 'in' will search in Pandas index and not values
1295 in BTC_price_series
```

[89]:
```
True
```

MINI CHALLENGE #10: - Check if the stock price 399 exists in the BTC_price_series Pandas Series or not - Round stock prices to the nearest integer and check again

[90]:
```
399 in BTC_price_series.values
```

[90]:
```
False
```

[91]:
```
prices_series = round(BTC_price_series)
prices_series
```

[91]:
```
0         457.0
1         424.0
2         395.0
3         409.0
4         399.0
          ...
2380    55951.0
2381    57750.0
2382    58918.0
2383    58919.0
2384    59096.0
Name: BTC-USD Price, Length: 2385, dtype: float64
```

[92]:
```
399 in prices_series.values
```

[92]:
```
True
```

11 EXCELLENT JOB!

12 MINI CHALLENGE SOLUTIONS

MINI CHALLENGE #1 SOLUTION: - Define a Pandas Series named "my_series" that contains your top 3 favourite stocks. Confirm the datatype of "my_series"

[51]:
```
# Let's define a Python list that contains 3 top stocks
my_list = ['Facebook','Apple','Nvidia']
my_series = pd.Series(data = my_list)
my_series
```

[51]:
```
0    Facebook
1       Apple
2      Nvidia
dtype: object
```

[52]:
```
type(my_series)
```

[52]:
```
pandas.core.series.Series
```

MINI CHALLENGE #2 SOLUTION: - Define a Pandas Series named "my_series" that contains your top 3 favourite stocks. Instead of using default numeric indexes (similar to mini challenge #1), use the following indexes "stock #1", "stock #2", and "stock #3"

[53]:
```
# Let's define a Python list that contains 3 stocks as follows
my_list = ['Facebook','Apple','Nvidia']

# Let's define a python list as shown below. This python list will be used for␣
 ↪the Series index:
my_labels = ['stock #1', 'stock #2', 'stock #3']
```

[54]:
```
# Let's create a one dimensional Pandas "series"
# Let's use Pandas Constructor Method to create a series from a Python list
# Note that this series is formed of data and associated labels
my_series = pd.Series(data = my_list, index = my_labels)
my_series
```

[54]:
```
stock #1    Facebook
stock #2       Apple
stock #3      Nvidia
dtype: object
```

MINI CHALLENGE #3 SOLUTION: - Create a Pandas Series from a dictionary with 3 of your favourite stocks and their corresponding prices

[55]:
```
stocks = {'Facebook': 3000,
          'Apple'   : 400,
          'Nvidia'  : 2200}
print(stocks)
```

```
{'Facebook': 3000, 'Apple': 400, 'Nvidia': 2200}
```

[56]:
```
# Let's define a Pandas Series Using the dictionary
my_series = pd.Series(stocks)
my_series
```

[56]:
```
Facebook    3000
Apple        400
Nvidia      2200
dtype: int64
```

MINI CHALLENGE #4 SOLUTION: - What is the size of the Pandas Series? (External Research is Required)

[57]:
```
# size is used to return the size of the series
crypto_series.size
```

[57]:
```
5
```

MINI CHALLENGE #5 SOLUTION: - Show the last 2 rows in the Pandas Series (External Research is Required) - How many bytes does this Pandas Series consume in memory? (External Research is Required)

[58]:
```
crypto_prices.tail(2)
```

[58]:
```
3    20
4    70
dtype: int64
```

[59]:
```
crypto_prices.memory_usage()
```

[59]:
```
168
```

[ ]:
```

```

MINI CHALLENGE #6 SOLUTION: - Set Squeeze = False and rerun the cell, what do you notice? Use Type to compare both outputs

[60]:
```
BTC_price_series = pd.read_csv('crypto.csv', squeeze = False)
# Note that when you set Squeeze = False, the data is stored in a DataFrame by␣
 ↪default.
# DataFrame is simply used to store multi dimensional data as compares to␣
 ↪Pandas Series that only holds 1-D dataset
# Note that DataFrames has proper formatting when you attempt to view them as␣
 ↪shown below
# Note that Pandas Series has no formatting
```

[61]:
```
BTC_price_series
```

[61]:
```
      BTC-USD Price
0         457.334015
1         424.440002
2         394.795990
3         408.903992
4         398.821014
…                …
2380    55950.746090
2381    57750.199220
2382    58917.691410
2383    58918.832030
2384    59095.808590

[2385 rows x 1 columns]
```

[62]:
```
type(BTC_price_series)
```

[62]:
```
pandas.core.frame.DataFrame
```

[63]:
```
BTC_price_series = pd.read_csv('crypto.csv', squeeze = True)
type(BTC_price_series)
```

[63]:
```
pandas.core.series.Series
```

MINI CHALLENGE #7 SOLUTION: - Given the following Pandas Series, convert all positive values to negative using python built-in functions - Obtain only unique values (ie: Remove duplicates) using python built-in functions - my_series = pd.Series(data = [-10, 100, -30, 50, 100])

[64]:
```
my_series = pd.Series(data = [-10, 100, -30, 50, 100])
my_series
```

[64]:
```
0    -10
1    100
2    -30
3     50
4    100
dtype: int64
```

[65]:
```
abs(my_series)
```

[65]:
```
0     10
1    100
2     30
3     50
4    100
dtype: int64
```

[66]:
```
set(my_series)
```

[66]:
```
{-30, -10, 50, 100}
```

MINI CHALLENGE #8 SOLUTION: - Sort the BTC_price_series values in a decending order instead. Make sure to update values in-memory.

[67]:
```
BTC_price_series.sort_values(ascending = False, inplace = True)
BTC_price_series
```

[67]:
```
2365    61243.085940
2366    59302.316410
2384    59095.808590
2383    58918.832030
2382    58917.691410
           ...
123       210.339004
120       209.843994
121       208.097000
122       199.259995
119       178.102997
Name: BTC-USD Price, Length: 2385, dtype: float64
```

MINI CHALLENGE #9 SOLUTION: - Obtain the average price using two different methods

[68]:
```
# Obtain the average - Solution #1
BTC_price_series.sum()/BTC_price_series.count()
```

[68]:
```
6471.857332852285
```

[69]:
```
# Obtain the average - Solution #s
BTC_price_series.mean()
```

[69]:
```
6471.857332852285
```

[ ]:
```

```

MINI CHALLENGE #10 SOLUTION: - Check if the stock price 399 exists in the BTC_price_series Pandas Series or not - Round stock prices to the nearest integer and check again

[70]:
```
399 in BTC_price_series.values
```

[70]:
```
False
```

[71]:
```
prices_series = round(BTC_price_series)
prices_series
```

[71]:
```
2365    61243.0
2366    59302.0
2384    59096.0
2383    58919.0
2382    58918.0
          …
123       210.0
120       210.0
121       208.0
122       199.0
119       178.0
Name: BTC-USD Price, Length: 2385, dtype: float64
```

[72]:
```
399 in prices_series.values
```

[72]:
```
True
```

[ ]:
```

```

[ ]:
```

```