

# API Mini Project

This exercise will require you to pull some data from <https://data.nasdaq.com/> (<https://data.nasdaq.com/>) (formerly Quandl API).

As a first step, you will need to register a free account on the <https://data.nasdaq.com/> (<https://data.nasdaq.com/>) website.

After you register, you will be provided with a unique API key, that you should store:

*Note:* Use a `.env` file and put your key in there and `python-dotenv` to access it in this notebook.

The code below uses a key that was used when generating this project but has since been deleted. Never submit your keys to source control. There is a `.env-example` file in this repository to illustrate what you need. Copy that to a file called `.env` and use your own api key in that `.env` file. Make sure you also have a `.gitignore` file with a line for `.env` added to it.

The standard Python gitignore is [here](https://github.com/github/gitignore/blob/master/Python.gitignore) (<https://github.com/github/gitignore/blob/master/Python.gitignore>) you can just copy that.

```
In [1]: ▶ # get api key from your .env file
import os
from dotenv import load_dotenv

load_dotenv()
API_KEY = os.getenv('NASDAQ_API_KEY')
```

Nasdaq Data has a large number of data sources, but, unfortunately, most of them require a Premium subscription. Still, there are also a good number of free datasets.

For this mini project, we will focus on equities data from the Frankfurt Stock Exchange (FSE), which is available for free. We'll try and analyze the stock prices of a company called Carl Zeiss Meditec, which manufactures tools for eye examinations, as well as medical lasers for laser eye surgery: <https://www.zeiss.com/meditec/int/home.html> (<https://www.zeiss.com/meditec/int/home.html>). The company is listed under the stock ticker AFX\_X.

You can find the detailed Nasdaq Data API instructions here: <https://docs.data.nasdaq.com/docs/in-depth-usage> (<https://docs.data.nasdaq.com/docs/in-depth-usage>)

While there is a dedicated Python package for connecting to the Nasdaq API, we would prefer that you use the *requests* package, which can be easily downloaded using *pip* or *conda*. You can find the documentation for the package here: <http://docs.python-requests.org/en/master/> (<http://docs.python-requests.org/en/master/>).

Finally, apart from the *requests* package, you are encouraged to not use any third party Python packages, such as *pandas*, and instead focus on what's available in the Python Standard Library (the *collections* module might come in handy: <https://pymotw.com/3/collections/> (<https://pymotw.com/3/collections/>)). Also, since you won't have access to DataFrames, you are encouraged to use Python's native data structures - preferably dictionaries, though some questions can also be answered using lists. You can read more on these data structures here: <https://docs.python.org/3/tutorial/datastructures.html> (<https://docs.python.org/3/tutorial/datastructures.html>).

Keep in mind that the JSON responses you will be getting from the API map almost one-to-one to Python's dictionaries. Unfortunately, they can be very nested, so make sure you read up on indexing dictionaries in the documentation provided above.

```
In [2]:  # First, import the relevant modules
import requests
import json
import collections
import pandas as pd
```

Note: API's can change a bit with each version, for this exercise it is recommended to use the nasdaq api at <https://data.nasdaq.com/api/v3/> . This is the same api as what used to be quandl so <https://www.quandl.com/api/v3/> should work too.

Hint: We are looking for the AFX\_X data on the datasets/FSE/ dataset.

```
In [3]:  # Now, call the Nasdaq API and pull out a small sample of the data (only one day) to get a glimpse
# into the JSON structure that will be returned
url = 'https://data.nasdaq.com/api/v3/datasets/FSE/AFX_X?start_date=2017-01-01&end_date=2017-01-01&api_key='+API_KEY
r = requests.get(url)
r.close()
```

```
In [4]: ▶ # Inspect the JSON structure of the object you created, and take note of how nested it is,
# as well as the overall structure
json_data = r.json()
print(json_data)
```

```
{'dataset': {'id': 10095370, 'dataset_code': 'AFX_X', 'database_code': 'FSE', 'name': 'Carl Zeiss Meditec (AFX_X)', 'description': 'Stock Prices for Carl Zeiss Meditec (2020-11-02) from the Frankfurt Stock Exchange.<br><br>Trading System: Xetra<br><br>ISIN: DE0005313704', 'refreshed_at': '2020-12-01T14:48:09.907Z', 'newest_available_date': '2020-12-01', 'oldest_available_date': '2000-06-07', 'column_names': ['Date', 'Open', 'High', 'Low', 'Close', 'Change', 'Traded Volume', 'Turnover', 'Last Price of the Day', 'Daily Traded Units', 'Daily Turnover'], 'frequency': 'daily', 'type': 'Time Series', 'premium': False, 'limit': None, 'transform': None, 'column_index': None, 'start_date': '2017-01-01', 'end_date': '2017-01-01', 'data': [], 'collapse': None, 'order': None, 'database_id': 6129}}
```

```
In [5]: ▶ for k, value in json_data['dataset'].items():
print(k + ': ', json_data['dataset'][k], '---', type(json_data['dataset'][k]))
```

```
id: 10095370 --- <class 'int'>
dataset_code: AFX_X --- <class 'str'>
database_code: FSE --- <class 'str'>
name: Carl Zeiss Meditec (AFX_X) --- <class 'str'>
description: Stock Prices for Carl Zeiss Meditec (2020-11-02) from the Frankfurt Stock Exchange.<br><br>Trading System: Xetra<br><br>ISIN: DE0005313704 --- <class 'str'>
refreshed_at: 2020-12-01T14:48:09.907Z --- <class 'str'>
newest_available_date: 2020-12-01 --- <class 'str'>
oldest_available_date: 2000-06-07 --- <class 'str'>
column_names: ['Date', 'Open', 'High', 'Low', 'Close', 'Change', 'Traded Volume', 'Turnover', 'Last Price of the Day', 'Daily Traded Units', 'Daily Turnover'] --- <class 'list'>
frequency: daily --- <class 'str'>
type: Time Series --- <class 'str'>
premium: False --- <class 'bool'>
limit: None --- <class 'NoneType'>
transform: None --- <class 'NoneType'>
column_index: None --- <class 'NoneType'>
start_date: 2017-01-01 --- <class 'str'>
end_date: 2017-01-01 --- <class 'str'>
data: [] --- <class 'list'>
collapse: None --- <class 'NoneType'>
order: None --- <class 'NoneType'>
database_id: 6129 --- <class 'int'>
```

## The following tasks are completed below:

These are your tasks for this mini project:

1. Collect data from the Frankfurt Stock Exchange, for the ticker AFX\_X, for the whole year 2017 (keep in mind that the date format is YYYY-MM-DD).
2. Convert the returned JSON object into a Python dictionary.
3. Calculate what the highest and lowest opening prices were for the stock in this period.
4. What was the largest change in any one day (based on High and Low price)?
5. What was the largest change between any two days (based on Closing Price)?
6. What was the average daily trading volume during this year?
7. (Optional) What was the median trading volume during this year. (Note: you may need to implement your own function for calculating the median.)

```
In [6]: ▶ #data from the Frankfurt Stock Exchange, for the ticker AFX_X, for the whole year 2017
url2 = "https://data.nasdaq.com/api/v3/datasets/FSE/AFX_X?start_date=2017-01-01&end_date=2017-12-31&api_key="+API_K
r2 = requests.get(url2)
print(r2)
r2.close()
```

<Response [200]>

```
In [7]: ▶ #Convert the returned JSON object into a Python dictionary
json_data_2017 = r2.json()
for key, value in json_data_2017.items():
    afx_x_2017 = json_data_2017['dataset']
```

```
In [8]: ▶ # Convert json_data_2017 into a DataFrame
afx_x_2017 = json_data_2017.get('dataset', {})

# Convert the 'data' field inside 'dataset' to a DataFrame
df = pd.DataFrame(afx_x_2017.get('data', []), columns=afx_x_2017.get('column_names', []))
```

```
In [10]: # Calculate what the highest and lowest opening prices were for the stock in this period.  
high_open_price = df['Open'].max()  
low_open_price = df['Open'].min()  
  
print(f"\nHighest opening price for 2017 was: ${high_open_price}")  
print(f"Lowest opening price for 2017 was: ${low_open_price}")
```

Highest opening price for 2017 was: \$53.11  
Lowest opening price for 2017 was: \$34.0

```
In [11]: # What was the largest change in any one day (based on High and Low price)?  
#Let's create a new column for daily_change  
df['Daily_Change'] = df['Low'] - df['High']  
  
#Find the greatest change by taking the absolute values  
largest_change = df['Daily_Change'].abs().max()  
  
print(f"Largest change in any one day based on High and Low price was: {largest_change}")
```

Largest change in any one day based on High and Low price was: 2.8100000000000023

In [12]:  *# What was the Largest change between any two days (based on Closing Price)?*

```
# Extract the values fro the 'Close' column
closing_prices_list = df['Close']
n = len(closing_prices_list)

#Initialize to store the largest change
closing_price_change = []

#Iterate through the list to find the Largest change
for i in range(0, n-1):
    change = abs(closing_prices_list[i] - closing_prices_list[i+1])
    closing_price_change.append(change)

# Find the Largest change between any two days
twod_largest_change = max(closing_price_change)

print(f"Largest change between any two days based on the closing price was: {twod_largest_change}")
```


Largest change between any two days based on the closing price was: 2.5599999999999995

In [13]:  *# What was the average daily trading volume during this year?*

```
avg_daily_traded_vol = df['Traded Volume'].mean()

print(f"The average daily trading volume during the 2017 year was: {avg_daily_traded_vol}")
```

The average daily trading volume during the 2017 year was: 89124.33725490196

In [14]:  *# (Optional) What was the median trading volume during this year.*  
*# (Note: you may need to implement your own function for calculating the median.)*

```
median_traded_vol_2017 = df['Traded Volume'].median()

print(f"The median trading volume during the 2017 year was: {median_traded_vol_2017}")
```

The median trading volume during the 2017 year was: 76286.0

