3_Data_Wrangling_and_Visualization

December 25, 2023

Coursebook: Data Wrangling and Visualization

- Bagian 3 Audit Analytics untuk Bank Rakyat Indonesia
- Durasi: 7 Jam
- *Last Updated*: December 2023

- *Coursebook* ini disusun dan dikurasi oleh tim produk dan instruktur dari Algoritma Data Science School

1 Latar Belakang

Coursebook ini merupakan bagian dari **BRI Audit Analytics** yang disiapkan oleh Algoritma. *Coursebook* ini ditujukan hanya untuk khalayak terbatas, yaitu individu dan organisasi yang menerima *coursebook* ini langsung dari organisasi pelatihan. Tidak boleh direproduksi, didistribusikan, diterjemahkan, atau diadaptasi dalam bentuk apapun di luar individu dan organisasi ini tanpa izin.

Algoritma adalah pusat pendidikan *data science* yang berbasis di Jakarta. Kami menyelenggarakan *workshop* dan program pelatihan untuk membantu para profesional dan mahasiswa dalam menguasai berbagai sub-bidang *data science* yaitu: *data visualization, machine learning,* statistik, dan lain sebagainya.

2 Tujuan Pelatihan

Coursebook ini berfokus pada:

- Stacking and Unstacking
- Reproducible Environment
- Working with MultiIndex DataFrames
- Reshaping your DataFrame with Melt
- Using Group By Effectively
- Visual Data Exploratory

3 Reproducible Environment

Ada beberapa paket baru yang akan digunakan dalam materi ini. Biasanya, kita dapat menggunakan `pip install/conda install` untuk menginstal library baru ke environment kita. Namun untuk saat ini, mari kita coba pendekatan lain dalam mempersiapkan library yang diperlukan untuk proyek tertentu.

Bayangkan Anda sedang bekerja dengan tim Anda dalam sebuah proyek kolaboratif. Anda menginisialisasi proyek dengan dependensi dan versi tertentu di komputer Anda dan semuanya berjalan dengan baik. Nantinya, Anda perlu 'mengirimkan' proyek itu ke tim Anda yang mengharuskan mereka menyiapkan environment yang sama dengan Anda. Lalu apa yang akan Anda lakukan untuk memastikan program itu juga berjalan lancar di mesin mereka?

Di sinilah Anda perlu membuat environment Anda dapat direproduksi dengan membuat file `requirements.txt`.

Lihat pada folder material utama, Anda akan menemukan file `requirements.txt` yang isinya seperti ini:

```
matplotlib==3.8.1
numpy==1.26.1
pandas==2.0.0
yfinance==0.2.31
```

Perhatikan kita memiliki baris untuk setiap library, lalu nomor versi. Hal ini penting karena saat Anda mulai mengembangkan aplikasi python, Anda akan mengembangkan aplikasi dengan mempertimbangkan versi library tertentu. Sederhananya, `requirements.txt` membantu melacak versi setiap library yang Anda gunakan untuk mencegah perubahan yang tidak terduga.

3.1 Importing Requirements

Kita sudah membahas untuk apa file persyaratan itu, tetapi bagaimana cara menggunakannya? Karena kita tidak ingin menginstal dan melacak secara manual setiap library yang diperlukan untuk proyek tertentu, mari kita coba mengimpor persyaratan dengan langkah-langkah berikut:

1. Aktifkan environment yang ingin digunakan:

   `conda activate <ENV_NAME>`

   Apabila belum ada, maka perlu membuat environment baru:

   conda create -n [ENV_NAME] python=[PYTHON_VERSION]

   Jangan lupa instalasi kernel di dalam environment tersebut apabila ingin dapat diakses menggunakan jupyter notebook:

   > pip install ipykernel

   > python -m ipykernel install –user –name=[ENV_NAME]

2. Navigasikan path ke folder di mana file `requirements.txt` berada

   `cd <PATH_TO_REQUIREMENTS>`

3. Instalasi packages dari file tersebut

   "' pip install -r requirements.txt

3.2 Exporting Requirements

Perintah `pip install` selalu menginstal versi terbaru dari sebuah library, namun terkadang, Anda mungkin ingin menginstal versi tertentu yang Anda tahu berfungsi pada proyek Anda.

File persyaratan memungkinkan Anda menentukan dengan tepat library dan versi mana yang harus diinstal. Anda dapat mengikuti langkah-langkah berikut untuk membuat file kebutuhan Anda:

1. Aktifkan environment

```
conda activate <ENV_NAME>
```

2. Navigasikan path ke folder tempat di mana file `requirements.txt` ingin disimpan

```
cd <PATH_TO_REQUIREMENTS_FOLDER>
```

3. Export environment: membuat daftar packages beserta versinya.

```
pip list --format=freeze > requirements.txt
```

Notes!

Anda dapat menyimpan file dengan nama lain, namun sebagai konvensi biasa digunakan penamaan requirements.txt

---

4 Data Wrangling and Reshaping

Dalam dua course sebelumnya, kita telah mempelajari beberapa teknik umum dan mempelajari cara mengeksplorasi data menggunakan metode bawaan `pandas`. Secara khusus, di bagian pertama dan kedua dari seri ini kita telah membahas cara menggunakan tools inspeksi, diagnostik, dan eksplorasi berikut:

Data Inspection

- `.head()` and `.tail()`
- `.describe()`
- `.shape` and `.size`
- `.axes`
- `.dtypes`
- Subsetting using `.loc`, `.iloc` and conditionals

Diagnostic and Exploratory

- Tables
- Cross-Tables and Aggregates
- Using `aggfunc` for aggregate functions
- Pivot Tables
- Working with DateTime
- Working with Categorical Data
- Duplicates and Missing Value Treatment

Paruh pertama course ini berfungsi sebagai perpanjangan dari materi terakhir. Kami akan mengenalkan beberapa teknik baru untuk melengkapi tools EDA. Mari kita mulai dengan teknik `reshaping` kembali.

[ ]:
```
import pandas as pd
import yfinance as data
pd.set_option('display.float_format', '{:,.3f}'.format) #display setting␣
 ↪purpose only
```

[ ]:
```
symbol = ['BRIS.JK' ,'BBRI.JK', 'BMRI.JK']
start_date = '2020-01-01' # 1 januari 2020
end_date = '2023-12-31' # 31 desember 2023
stock = data.download(tickers = symbol, start = start_date, end = end_date)
stock.columns.names = ['Attributes', 'Symbols']
stock.head()
```

```
[**********100%%**********]  3 of 3 completed
```

[ ]:
```
Attributes Adj Close                          Close                          High
Symbols      BBRI.JK   BMRI.JK BRIS.JK   BBRI.JK   BMRI.JK BRIS.JK   BBRI.JK  \
Date
2020-01-02 3,772.726 3,239.328 326.177 4,410.000 3,875.000 332.000 4,410.000
2020-01-03 3,781.281 3,228.878 322.247 4,420.000 3,862.500 328.000 4,440.000
2020-01-06 3,738.507 3,176.631 318.318 4,370.000 3,800.000 324.000 4,390.000
2020-01-07 3,764.171 3,176.631 312.423 4,400.000 3,800.000 318.000 4,410.000
2020-01-08 3,747.061 3,134.833 306.528 4,380.000 3,750.000 312.000 4,400.000

Attributes                           Low                           Open
Symbols      BMRI.JK BRIS.JK   BBRI.JK   BMRI.JK BRIS.JK   BBRI.JK   BMRI.JK  \
Date
2020-01-02 3,887.500 336.000 4,360.000 3,825.000 330.000 4,400.000 3,837.500
2020-01-03 3,912.500 336.000 4,390.000 3,812.500 326.000 4,420.000 3,875.000
2020-01-06 3,837.500 334.000 4,320.000 3,762.500 320.000 4,360.000 3,825.000
2020-01-07 3,862.500 324.000 4,380.000 3,787.500 316.000 4,410.000 3,862.500
2020-01-08 3,775.000 318.000 4,340.000 3,687.500 312.000 4,380.000 3,775.000

Attributes              Volume
Symbols    BRIS.JK    BBRI.JK    BMRI.JK  BRIS.JK
Date
2020-01-02 330.000   41714100   37379800  1456400
2020-01-03 334.000   82898300   70294600  4989600
2020-01-06 328.000   44225100   61892000  6937900
2020-01-07 324.000  103948100   70895600  6319400
2020-01-08 318.000  171751200  105080600  4058800
```

Jika Anda belum menginstal modul `pandas_datareader`, atau jika Anda mengikuti buku pelajaran ini tanpa koneksi internet aktif, Anda dapat memuatnya dari objek serial yang kami simpan di folder `data_cache` Anda.

Membuat objek DataFrame dengan membaca dari `pickle`:

- `stock = pd.read_pickle('data_cache/stock')`

Membuat serial objek DataFrame menjadi aliran byte menggunakan `pickle`:

- `stock.to_pickle('data_cache/stock')`

[ ]:
```
# buat dataframe kedalam pickle
# stock.to_pickle('data_cache/stock')
```

[ ]:
```
# stock = pd.read_pickle('data_cache/stock')
# stock.head()
```

4.1 Slicing Multi-Index DataFrame

Multi-Index Dataframe adalah bentuk dataframe yang memiliki level indexing lebih dari 1 baik pada baris, kolom, ataupun keduanya. Hal yang perlu diperhatikan dalam MultiIndex Dataframe adalah bentuk dataframe ini terkadang tidak bisa langsung kita gunakan untuk menganalisis data, sehingga akan ada beberapa perlakuan untuk kita mengiris atau mengubah bentuknya ke dataframe yang lebih sederhana.

Perhatikan bahwa data `stock` adalah Multi-Index DataFrame, di mana level dari columnnya terdiri dari:

- `Attributes` = level 0
- `Symbols` = level 1

Perhatikan bagaimana Dataframe kita merupakan data multi-indeks. Jika Anda memperhatikan dengan seksama, Anda dapat melihat 2 tingkat sumbu kolom: `Attributes` dan `Symbols`. Jika Anda mengelompokkan data menggunakan tanda kurung siku [ ] , Anda akan mengakses indeks tingkat tertinggi:

[ ]:
```
# Ambil kolom High dengan semua attributnya
stock['High']

# Jika tidak, kode ini akan menimbulkan kesalahan
# stock['AAPL']
```

[ ]:
```
Symbols      BBRI.JK   BMRI.JK   BRIS.JK
Date
2020-01-02 4,410.000 3,887.500   336.000
2020-01-03 4,440.000 3,912.500   336.000
2020-01-06 4,390.000 3,837.500   334.000
2020-01-07 4,410.000 3,862.500   324.000
2020-01-08 4,400.000 3,775.000   318.000
…                …         …         …
2023-12-15 5,600.000 6,000.000 1,715.000
2023-12-18 5,575.000 5,950.000 1,705.000
2023-12-19 5,550.000 5,975.000 1,710.000
```

```
2023-12-20 5,700.000 5,975.000 1,765.000
2023-12-21 5,600.000 5,975.000 1,765.000

[970 rows x 3 columns]
```

4.1.1 Cross Section

Menggunakan method `.xs()` (cross section) untuk mengambil kolom (`axis = 1`) pada level dalam. Parameter:

- `key` : nama kolom/baris yang kita ingin ambil
- `level` : kolom/baris tersebut ada di level apa?
- `axis` : levelnya terdapat pada index kolom/baris
  - `0` untuk baris
  - `1` untuk kolom

[ ]:
```
# mengambil seluruh nilai BBRI
stock.xs(key='BBRI.JK',
         level= 1,
         axis = 1)
```

[ ]:
```
Attributes  Adj Close     Close      High       Low      Open     Volume
Date
2020-01-02  3,772.726 4,410.000 4,410.000 4,360.000 4,400.000   41714100
2020-01-03  3,781.281 4,420.000 4,440.000 4,390.000 4,420.000   82898300
2020-01-06  3,738.507 4,370.000 4,390.000 4,320.000 4,360.000   44225100
2020-01-07  3,764.171 4,400.000 4,410.000 4,380.000 4,410.000  103948100
2020-01-08  3,747.061 4,380.000 4,400.000 4,340.000 4,380.000  171751200
…                …         …         …         …         …         …
2023-12-15  5,550.000 5,550.000 5,600.000 5,550.000 5,575.000  252448800
2023-12-18  5,500.000 5,500.000 5,575.000 5,500.000 5,575.000  102780900
2023-12-19  5,550.000 5,550.000 5,550.000 5,450.000 5,450.000  135207300
2023-12-20  5,550.000 5,550.000 5,700.000 5,550.000 5,700.000  138470900
2023-12-21  5,575.000 5,575.000 5,600.000 5,525.000 5,550.000   99049600

[970 rows x 6 columns]
```

4.2 `stack()` and `unstack()`

`stack()` menumpuk level yang ditentukan dari kolom ke indeks dan sangat berguna pada DataFrames yang memiliki kolom multi-level. Ia melakukannya dengan "menggeser" kolom untuk membuat level baru pada indeksnya.

Hal ini lebih mudah dipahami bila kita hanya melihat contohnya. Perhatikan bahwa `stock` memiliki kolom 2 tingkat (Atribut dan Simbol) dan indeks 1 tingkat (Tanggal):

[ ]:
```
stock.head(10)
```

[ ]:
```
Attributes Adj Close                        Close                        High
Symbols      BBRI.JK   BMRI.JK BRIS.JK   BBRI.JK   BMRI.JK BRIS.JK   BBRI.JK
Date
2020-01-02 3,772.726 3,239.328 326.177 4,410.000 3,875.000 332.000 4,410.000  \
2020-01-03 3,781.281 3,228.878 322.247 4,420.000 3,862.500 328.000 4,440.000
2020-01-06 3,738.507 3,176.631 318.318 4,370.000 3,800.000 324.000 4,390.000
2020-01-07 3,764.171 3,176.631 312.423 4,400.000 3,800.000 318.000 4,410.000
2020-01-08 3,747.061 3,134.833 306.528 4,380.000 3,750.000 312.000 4,400.000
2020-01-09 3,764.171 3,218.429 312.423 4,400.000 3,850.000 318.000 4,420.000
2020-01-10 3,772.726 3,228.878 308.493 4,410.000 3,862.500 314.000 4,430.000
2020-01-13 3,858.275 3,228.878 314.388 4,510.000 3,862.500 320.000 4,510.000
2020-01-14 3,909.605 3,239.328 314.388 4,570.000 3,875.000 320.000 4,600.000
2020-01-15 3,918.160 3,197.530 314.388 4,580.000 3,825.000 320.000 4,580.000

Attributes                         Low                         Open
Symbols      BMRI.JK BRIS.JK   BBRI.JK   BMRI.JK BRIS.JK   BBRI.JK   BMRI.JK
Date
2020-01-02 3,887.500 336.000 4,360.000 3,825.000 330.000 4,400.000 3,837.500  \
2020-01-03 3,912.500 336.000 4,390.000 3,812.500 326.000 4,420.000 3,875.000
2020-01-06 3,837.500 334.000 4,320.000 3,762.500 320.000 4,360.000 3,825.000
2020-01-07 3,862.500 324.000 4,380.000 3,787.500 316.000 4,410.000 3,862.500
2020-01-08 3,775.000 318.000 4,340.000 3,687.500 312.000 4,380.000 3,775.000
2020-01-09 3,862.500 322.000 4,370.000 3,762.500 310.000 4,400.000 3,775.000
2020-01-10 3,862.500 324.000 4,390.000 3,825.000 314.000 4,430.000 3,850.000
2020-01-13 3,875.000 324.000 4,420.000 3,825.000 314.000 4,430.000 3,862.500
2020-01-14 3,900.000 326.000 4,520.000 3,850.000 318.000 4,540.000 3,900.000
2020-01-15 3,900.000 322.000 4,530.000 3,800.000 316.000 4,570.000 3,875.000

Attributes              Volume
Symbols    BRIS.JK    BBRI.JK    BMRI.JK   BRIS.JK
Date
2020-01-02 330.000   41714100   37379800   1456400
2020-01-03 334.000   82898300   70294600   4989600
2020-01-06 328.000   44225100   61892000   6937900
2020-01-07 324.000  103948100   70895600   6319400
2020-01-08 318.000  171751200  105080600   4058800
2020-01-09 312.000   72072000   75587000   4928400
2020-01-10 318.000  124809200   64231800   6175000
2020-01-13 316.000  111351400   75399600  10833200
2020-01-14 322.000  158414200  104134400   7388400
2020-01-15 320.000  102566700   89795200  18046800
```

Saat kita menumpuk `stock` DataFrame, kita mengecilkan jumlah level pada kolomnya sebanyak satu: `stock` sekarang memiliki 1 kolom level bernama `Attributes`:

[ ]:
```
stock.stack()
```

[ ]:
```
Attributes           Adj Close     Close      High       Low      Open
Date       Symbols
2020-01-02 BBRI.JK   3,772.726 4,410.000 4,410.000 4,360.000 4,400.000  \
           BMRI.JK   3,239.328 3,875.000 3,887.500 3,825.000 3,837.500
           BRIS.JK     326.177   332.000   336.000   330.000   330.000
2020-01-03 BBRI.JK   3,781.281 4,420.000 4,440.000 4,390.000 4,420.000
           BMRI.JK   3,228.878 3,862.500 3,912.500 3,812.500 3,875.000
…                        …         …         …         …         …
2023-12-20 BMRI.JK   5,925.000 5,925.000 5,975.000 5,900.000 5,900.000
           BRIS.JK   1,755.000 1,755.000 1,765.000 1,710.000 1,710.000
2023-12-21 BBRI.JK   5,575.000 5,575.000 5,600.000 5,525.000 5,550.000
           BMRI.JK   5,975.000 5,975.000 5,975.000 5,925.000 5,950.000
           BRIS.JK   1,690.000 1,690.000 1,765.000 1,680.000 1,765.000

Attributes             Volume
Date       Symbols
2020-01-02 BBRI.JK   41714100
           BMRI.JK   37379800
           BRIS.JK    1456400
2020-01-03 BBRI.JK   82898300
           BMRI.JK   70294600
…                        …
2023-12-20 BMRI.JK   65361000
           BRIS.JK   39163500
2023-12-21 BBRI.JK   99049600
           BMRI.JK   50363900
           BRIS.JK   25310000

[2910 rows x 6 columns]
```

`unstack()` melakukan yang sebaliknya: ia "menggeser" level dari sumbu indeks ke sumbu kolom. **Coba dan buat tumpukan DataFrame, lalu terapkan `unstack` pada DataFrame baru untuk melihatnya kembali ke bentuk aslinya:**

[ ]:
```
## Write your code to try out .unstack() method here
```

Dive Deeper

Jawablah pertanyaan-pertanyaan berikut ini untuk memastikan Anda dapat melanjutkan sesi berikutnya:

1. Bagaimana cara menukar posisi (level) Symbols dan Attributes?
2. Berdasarkan pengetahuan Anda, (simbol) perusahaan apa yang layak untuk diinvestasikan? (Anda dapat melihat fluktuasinya, artinya, dll)

[ ]:
```
# Write your solution code here
```

Knowledge Check: Stack and Unstack

Which of the following statement is correct? Manakah dibawah ini yang merupakan pernyataan

yang benar?

- ☐ `stack()` mengubah DataFrame dari lebar ke panjang
- ☐ `unstack()` mengubah DataFrame dari panjang ke lebar
- ☐ `unstack()` mengubah DataFrame dari lebar ke panjang

4.3 Melt

Berbicara tentang membentuk kembali DataFrame dari format lebar ke format panjang, metode lain yang harus ada di perangkat Anda adalah `melt()`. Pertimbangkan DataFrame berikut, yang dibuat dari metode Pemotong MultiIndex `pandas`, `.xs()` (Singkatan dari 'Cross Section'):

[ ]:
```
bbri = stock.xs(key = 'BBRI.JK', level='Symbols', axis=1)
bbri.head()
```

[ ]:
```
Attributes  Adj Close     Close      High       Low      Open     Volume
Date
2020-01-02  3,772.726 4,410.000 4,410.000 4,360.000 4,400.000   41714100
2020-01-03  3,781.281 4,420.000 4,440.000 4,390.000 4,420.000   82898300
2020-01-06  3,738.507 4,370.000 4,390.000 4,320.000 4,360.000   44225100
2020-01-07  3,764.171 4,400.000 4,410.000 4,380.000 4,410.000  103948100
2020-01-08  3,747.061 4,380.000 4,400.000 4,340.000 4,380.000  171751200
```

[ ]:
```
bbri.shape
```

[ ]:
```
(970, 6)
```

DataFrame di atas lebar: memiliki 968 baris dan 6 kolom. Fungsi `melt()` mengumpulkan semua kolom menjadi satu dan menyimpan nilai yang sesuai dengan setiap kolom sehingga DataFrame yang dihasilkan memiliki 968 * 6 = 5.808 baris, bersama dengan kolom pengidentifikasi dan nilai:

[ ]:
```
bbri_melted = bbri.melt()
bbri_melted
```

[ ]:
```
     Attributes           value
0     Adj Close       3,772.726
1     Adj Close       3,781.281
2     Adj Close       3,738.507
3     Adj Close       3,764.171
4     Adj Close       3,747.061
…          …               …
5815     Volume 252,448,800.000
5816     Volume 102,780,900.000
5817     Volume 135,207,300.000
5818     Volume 138,470,900.000
5819     Volume  99,049,600.000

[5820 rows x 2 columns]
```

[ ]:
```
bbri_melted.shape
```

[ ]:
```
(5820, 2)
```

Knowledge Check : Apa yang membedakan antara melt dengan stack?

Secara opsional, kita dapat menentukan satu atau lebih kolom untuk menjadi variabel pengenal (`id_vars`), yang memperlakukan semua kolom lainnya sebagai variabel nilai (`value_vars`):

[ ]:
```
bbri.reset_index().melt(id_vars=['Date'])
```

[ ]:
```
           Date Attributes           value
0    2020-01-02  Adj Close       3,772.726
1    2020-01-03  Adj Close       3,781.281
2    2020-01-06  Adj Close       3,738.507
3    2020-01-07  Adj Close       3,764.171
4    2020-01-08  Adj Close       3,747.061
…           …          …               …
5815 2023-12-15     Volume 252,448,800.000
5816 2023-12-18     Volume 102,780,900.000
5817 2023-12-19     Volume 135,207,300.000
5818 2023-12-20     Volume 138,470,900.000
5819 2023-12-21     Volume  99,049,600.000

[5820 rows x 3 columns]
```

[ ]:
```
bbri.reset_index().melt(value_vars=['High', 'Low'])
```

[ ]:
```
     Attributes     value
0          High 4,410.000
1          High 4,440.000
2          High 4,390.000
3          High 4,410.000
4          High 4,400.000
…           …         …
1935        Low 5,550.000
1936        Low 5,500.000
1937        Low 5,450.000
1938        Low 5,550.000
1939        Low 5,525.000

[1940 rows x 2 columns]
```

5 Pandas dan Matplotlib

Tentunya ini adalah titik di mana seorang analis data menyiapkan beberapa grafik yang mencolok menggunakan perpustakaan `matplotlib` yang populer?

Baiklah. Bahkan lebih baik lagi, kita akan menggunakan metode `DataFrame.plot()`, yang ada

di dalam `pandas` yang kemudian memanggil fungsi plotting `matplotlib` di baliknya. Perhatikan bahwa kita menambahkan `matplotlib.pyplot` sebagai impor, meskipun kode kita tidak akan memanggil `matplotlib` secara eksplisit tetapi bergantung pada implementasi `pandas`.

Sekarang mari kita lihat kerangka data saham BRI:

[ ]:
```
bbri.head()
```

[ ]:
```
Attributes  Adj Close     Close      High       Low      Open     Volume
Date
2020-01-02  3,772.726 4,410.000 4,410.000 4,360.000 4,400.000   41714100
2020-01-03  3,781.281 4,420.000 4,440.000 4,390.000 4,420.000   82898300
2020-01-06  3,738.507 4,370.000 4,390.000 4,320.000 4,360.000   44225100
2020-01-07  3,764.171 4,400.000 4,410.000 4,380.000 4,410.000  103948100
2020-01-08  3,747.061 4,380.000 4,400.000 4,340.000 4,380.000  171751200
```

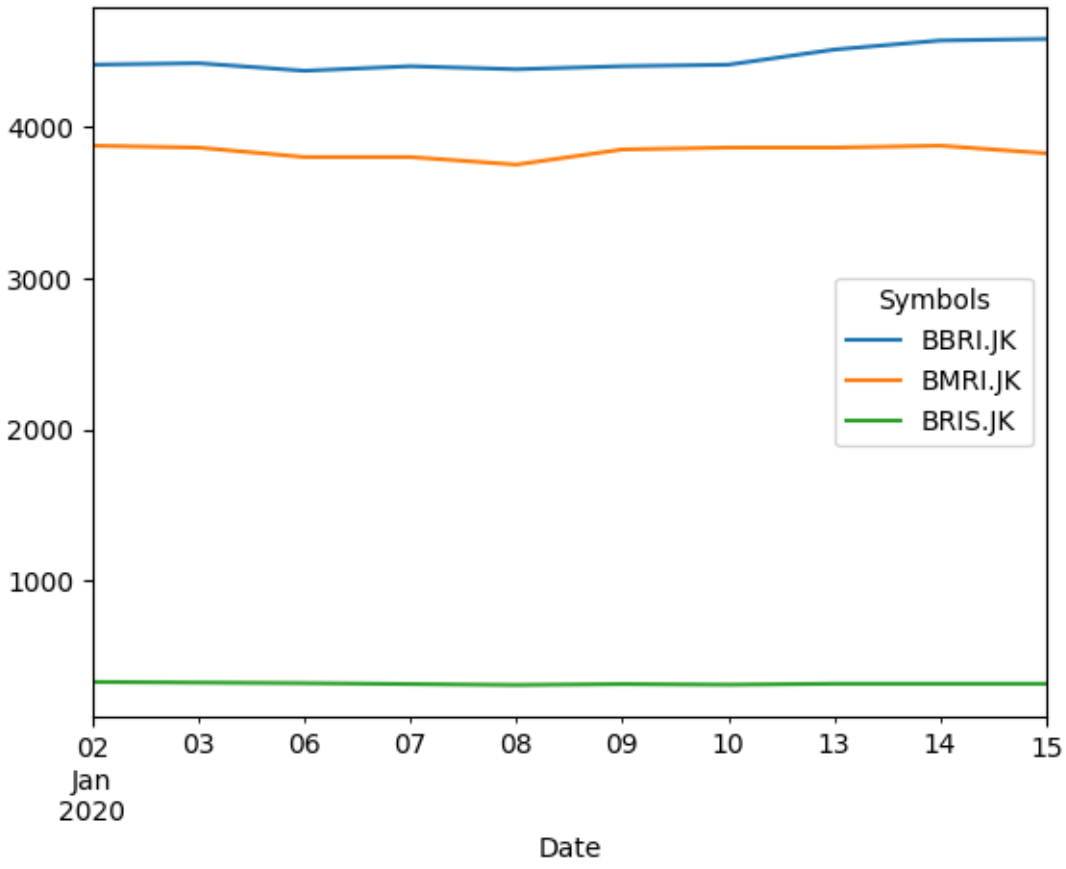
Cara terbaik untuk mendemonstrasikan peningkatan efisiensi `DataFrame.plot()` adalah dengan melihatnya bekerja. Kami akan memanggil `.plot()` langsung di `DataFrame` - `pandas` kami akan menangani kode matplotlib yang, menurut pengakuan matplotlib sendiri, *dapat menjadi hal yang menakutkan bagi banyak pengguna baru.*

[ ]:
```
stock
```

[ ]:
```
Attributes Adj Close                          Close
Symbols      BBRI.JK   BMRI.JK   BRIS.JK   BBRI.JK   BMRI.JK   BRIS.JK
Date
2020-01-02 3,772.726 3,239.328   326.177 4,410.000 3,875.000   332.000  \
2020-01-03 3,781.281 3,228.878   322.247 4,420.000 3,862.500   328.000
2020-01-06 3,738.507 3,176.631   318.318 4,370.000 3,800.000   324.000
2020-01-07 3,764.171 3,176.631   312.423 4,400.000 3,800.000   318.000
2020-01-08 3,747.061 3,134.833   306.528 4,380.000 3,750.000   312.000
…              …       …       …       …       …       …
2023-12-15 5,550.000 5,900.000 1,700.000 5,550.000 5,900.000 1,700.000
2023-12-18 5,500.000 5,925.000 1,695.000 5,500.000 5,925.000 1,695.000
2023-12-19 5,550.000 5,975.000 1,705.000 5,550.000 5,975.000 1,705.000
2023-12-20 5,550.000 5,925.000 1,755.000 5,550.000 5,925.000 1,755.000
2023-12-21 5,575.000 5,975.000 1,690.000 5,575.000 5,975.000 1,690.000

Attributes      High                          Low
Symbols      BBRI.JK   BMRI.JK   BRIS.JK   BBRI.JK   BMRI.JK   BRIS.JK
Date
2020-01-02 4,410.000 3,887.500   336.000 4,360.000 3,825.000   330.000  \
2020-01-03 4,440.000 3,912.500   336.000 4,390.000 3,812.500   326.000
2020-01-06 4,390.000 3,837.500   334.000 4,320.000 3,762.500   320.000
2020-01-07 4,410.000 3,862.500   324.000 4,380.000 3,787.500   316.000
2020-01-08 4,400.000 3,775.000   318.000 4,340.000 3,687.500   312.000
…              …       …       …       …       …       …
2023-12-15 5,600.000 6,000.000 1,715.000 5,550.000 5,900.000 1,700.000
2023-12-18 5,575.000 5,950.000 1,705.000 5,500.000 5,850.000 1,670.000
```

```
2023-12-19 5,550.000 5,975.000 1,710.000 5,450.000 5,925.000 1,690.000
2023-12-20 5,700.000 5,975.000 1,765.000 5,550.000 5,900.000 1,710.000
2023-12-21 5,600.000 5,975.000 1,765.000 5,525.000 5,925.000 1,680.000

Attributes      Open                          Volume
Symbols      BBRI.JK   BMRI.JK   BRIS.JK     BBRI.JK     BMRI.JK   BRIS.JK
Date
2020-01-02 4,400.000 3,837.500   330.000   41714100   37379800   1456400
2020-01-03 4,420.000 3,875.000   334.000   82898300   70294600   4989600
2020-01-06 4,360.000 3,825.000   328.000   44225100   61892000   6937900
2020-01-07 4,410.000 3,862.500   324.000  103948100   70895600   6319400
2020-01-08 4,380.000 3,775.000   318.000  171751200  105080600   4058800
…               …         …         …          …          …         …
2023-12-15 5,575.000 5,950.000 1,710.000  252448800  142382200  12317700
2023-12-18 5,575.000 5,850.000 1,690.000  102780900  101458600  12521500
2023-12-19 5,450.000 5,950.000 1,700.000  135207300   40763200   5861900
2023-12-20 5,700.000 5,900.000 1,710.000  138470900   65361000  39163500
2023-12-21 5,550.000 5,950.000 1,765.000   99049600   50363900  25310000

[970 rows x 18 columns]
```

[ ]:
```
stock['Close'].head(10).plot();
```



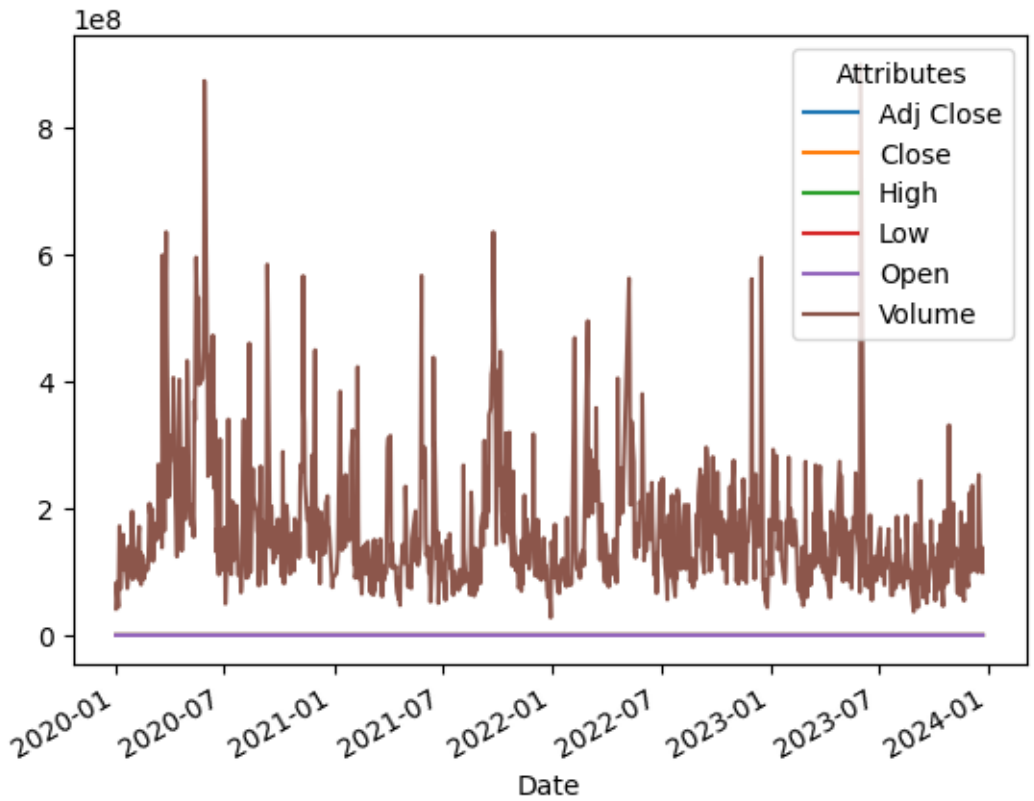
Kita dapat menyesuaikan plot kami dengan style sheet tetapi referensi praktis dapat dijangkau. Anda dapat mengganti 'default' dengan salah satu gaya yang tersedia dan menjalankan kembali kode plot untuk melihat gaya yang diterapkan.

[ ]:
```
import matplotlib.pyplot as plt
print(plt.style.available)
plt.style.use('default')
```

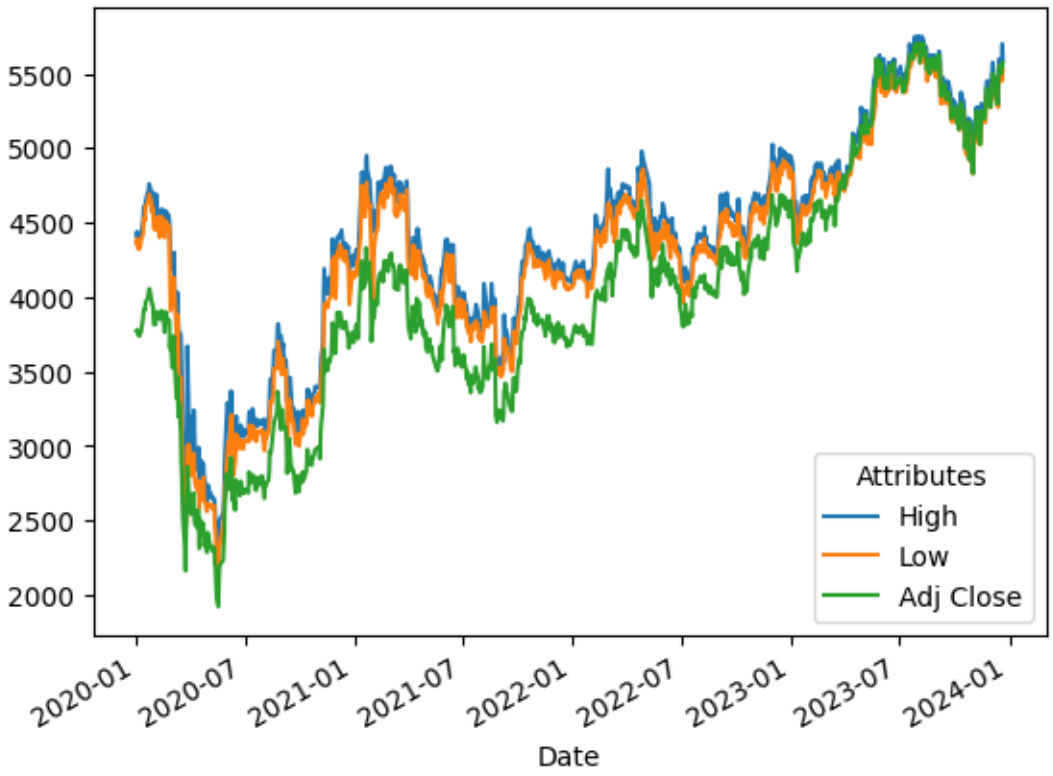
```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-
nogrid', 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight',
'ggplot', 'grayscale', 'seaborn-v0_8', 'seaborn-v0_8-bright',
'seaborn-v0_8-colorblind', 'seaborn-v0_8-dark', 'seaborn-v0_8-dark-palette',
'seaborn-v0_8-darkgrid', 'seaborn-v0_8-deep', 'seaborn-v0_8-muted',
'seaborn-v0_8-notebook', 'seaborn-v0_8-paper', 'seaborn-v0_8-pastel',
'seaborn-v0_8-poster', 'seaborn-v0_8-talk', 'seaborn-v0_8-ticks',
'seaborn-v0_8-white', 'seaborn-v0_8-whitegrid', 'tableau-colorblind10']
```

Karena metode `.plot()` dipanggil pada objek DataFrame, kita dapat memiliki DataFrame yang diindeks dengan beberapa kolom dan `plot` akan menanganinya menggunakan opsi defaultnya:

[ ]:
```
bbri.plot();
```



```
[ ]: bbri.loc[:, ['High', 'Low', 'Adj Close']].plot();
```



5.1 Other Visualization

Visualisasi berikut hanya perlu menggunakan **satu** kolom:

- Data kategorik:
  - **`.plot(kind='bar')` untuk barplot (diagram batang)**
  - **`.plot(kind='barh')` untuk horizontal barplot**
  - **`.plot(kind='box')` untuk boxplot (five number summary)**
  - `.plot(kind='pie')` untuk pie chart
- Data numerik:
  - **`.plot(kind='hist')` untuk histogram**
  - `.plot(kind='density')` untuk density plot
  - `.plot(kind='area')` untuk area plot

Visualisasi berikut perlu menggunakan **dua** kolom:

- `.plot(kind='scatter')` untuk scatter plot
- `.plot(kind='hexbin')` untuk hexagonal bin plot

Panduan untuk menentukan tipe visualisasi yang tepat: https://www.data-to-viz.com/

Silahkan mengacu referensi lengkapnya di official documentation untuk method `plot` apabila ingin eksplor visualisasi yang ada di luar lingkup course ini

6 Group By

Membentuk ulang data adalah komponen penting dari setiap tahapan di data wrangling karena memungkinkan analis untuk "mengirimkan pesan" data ke dalam bentuk yang diinginkan untuk diproses lebih lanjut.

Teknik lain yang sama pentingnya adalah pengelompokan berdasarkan operasi. Analis yang memiliki pengalaman dengan SQL atau perangkat analisis data lainnya (misalnya `tidyverse` R) akan menganggap operasi grup sebagai strategi yang familier dalam banyak alur kerja yang banyak menganalisis.

Pertimbangkan DataFrame berikut:

[ ]:
```
stock_adj = stock.stack()
stock_adj['Volume USD'] = stock_adj['Volume'] * stock_adj['Adj Close']
stock_adj = stock_adj.unstack()
stock_adj
```

[ ]:
```
Attributes Adj Close                          Close
Symbols      BBRI.JK   BMRI.JK   BRIS.JK   BBRI.JK   BMRI.JK   BRIS.JK
Date
2020-01-02 3,772.726 3,239.328   326.177 4,410.000 3,875.000   332.000  \
2020-01-03 3,781.281 3,228.878   322.247 4,420.000 3,862.500   328.000
2020-01-06 3,738.507 3,176.631   318.318 4,370.000 3,800.000   324.000
2020-01-07 3,764.171 3,176.631   312.423 4,400.000 3,800.000   318.000
2020-01-08 3,747.061 3,134.833   306.528 4,380.000 3,750.000   312.000
…               …       …       …       …       …       …
2023-12-15 5,550.000 5,900.000 1,700.000 5,550.000 5,900.000 1,700.000
2023-12-18 5,500.000 5,925.000 1,695.000 5,500.000 5,925.000 1,695.000
2023-12-19 5,550.000 5,975.000 1,705.000 5,550.000 5,975.000 1,705.000
2023-12-20 5,550.000 5,925.000 1,755.000 5,550.000 5,925.000 1,755.000
2023-12-21 5,575.000 5,975.000 1,690.000 5,575.000 5,975.000 1,690.000

Attributes      High                          Low  …               Open
Symbols      BBRI.JK   BMRI.JK   BRIS.JK   BBRI.JK  …   BRIS.JK   BBRI.JK
Date                                               …
2020-01-02 4,410.000 3,887.500   336.000 4,360.000  …   330.000 4,400.000  \
2020-01-03 4,440.000 3,912.500   336.000 4,390.000  …   326.000 4,420.000
2020-01-06 4,390.000 3,837.500   334.000 4,320.000  …   320.000 4,360.000
2020-01-07 4,410.000 3,862.500   324.000 4,380.000  …   316.000 4,410.000
2020-01-08 4,400.000 3,775.000   318.000 4,340.000  …   312.000 4,380.000
…               …       …       …       …  …       …       …
2023-12-15 5,600.000 6,000.000 1,715.000 5,550.000  … 1,700.000 5,575.000
2023-12-18 5,575.000 5,950.000 1,705.000 5,500.000  … 1,670.000 5,575.000
2023-12-19 5,550.000 5,975.000 1,710.000 5,450.000  … 1,690.000 5,450.000
```

```
2023-12-20 5,700.000 5,975.000 1,765.000 5,550.000  … 1,710.000 5,700.000
2023-12-21 5,600.000 5,975.000 1,765.000 5,525.000  … 1,680.000 5,550.000

Attributes                          Volume
Symbols      BMRI.JK   BRIS.JK    BBRI.JK    BMRI.JK   BRIS.JK
Date
2020-01-02 3,837.500   330.000   41714100   37379800   1456400  \
2020-01-03 3,875.000   334.000   82898300   70294600   4989600
2020-01-06 3,825.000   328.000   44225100   61892000   6937900
2020-01-07 3,862.500   324.000  103948100   70895600   6319400
2020-01-08 3,775.000   318.000  171751200  105080600   4058800
…              …       …         …         …          …
2023-12-15 5,950.000 1,710.000  252448800  142382200  12317700
2023-12-18 5,850.000 1,690.000  102780900  101458600  12521500
2023-12-19 5,950.000 1,700.000  135207300   40763200   5861900
2023-12-20 5,900.000 1,710.000  138470900   65361000  39163500
2023-12-21 5,950.000 1,765.000   99049600   50363900  25310000

Attributes            Volume USD
Symbols                  BBRI.JK             BMRI.JK            BRIS.JK
Date
2020-01-02   157,375,882,916.675 121,085,428,320.947    475,044,585.657
2020-01-03   313,461,787,446.875 226,972,716,839.746  1,607,885,657.007
2020-01-06   165,335,827,872.876 196,608,067,369.141  2,208,455,440.582
2020-01-07   391,278,438,345.825 225,209,185,370.898  1,974,324,582.458
2020-01-08   643,562,271,192.773 329,410,184,669.629  1,244,136,031.702
…                           …                  …              …
2023-12-15 1,401,090,840,000.000 840,054,980,000.000 20,940,090,000.000
2023-12-18   565,294,950,000.000 601,142,205,000.000 21,223,942,500.000
2023-12-19   750,400,515,000.000 243,560,120,000.000  9,994,539,500.000
2023-12-20   768,513,495,000.000 387,263,925,000.000 68,731,942,500.000
2023-12-21   552,201,520,000.000 300,924,302,500.000 42,773,900,000.000

[970 rows x 21 columns]
```

[ ]:
```
volumeusd = stock_adj.xs(key='Volume USD',
                         level='Attributes',
                         axis=1)
volumeusd = volumeusd.round(2)
volumeusd
```

[ ]:
```
Symbols                  BBRI.JK             BMRI.JK            BRIS.JK
Date
2020-01-02   157,375,882,916.670 121,085,428,320.950    475,044,585.660
2020-01-03   313,461,787,446.880 226,972,716,839.750  1,607,885,657.010
2020-01-06   165,335,827,872.880 196,608,067,369.140  2,208,455,440.580
2020-01-07   391,278,438,345.830 225,209,185,370.900  1,974,324,582.460
```

```
2020-01-08   643,562,271,192.770 329,410,184,669.630  1,244,136,031.700
…                            …                   …                …
2023-12-15 1,401,090,840,000.000 840,054,980,000.000 20,940,090,000.000
2023-12-18   565,294,950,000.000 601,142,205,000.000 21,223,942,500.000
2023-12-19   750,400,515,000.000 243,560,120,000.000  9,994,539,500.000
2023-12-20   768,513,495,000.000 387,263,925,000.000 68,731,942,500.000
2023-12-21   552,201,520,000.000 300,924,302,500.000 42,773,900,000.000

[970 rows x 3 columns]
```

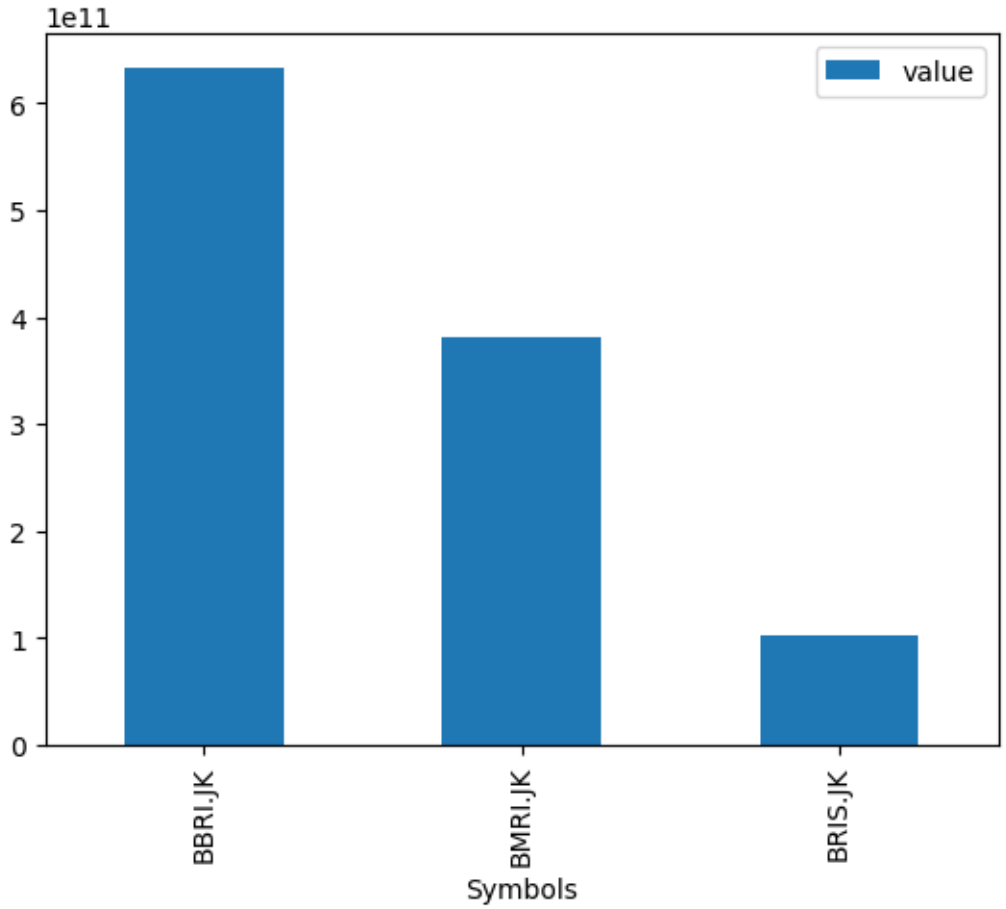
Perhatikan bagaimana bingkai data menunjukkan jumlah volume transaksi harian, misalnya kita ingin membandingkan rata-rata transaksi harian untuk BBRI.JK, BMRI.JK, dan BRIS.JK. Mari kita lakukan fungsi peleburan:

[ ]:
```
volume_melted = volumeusd.melt()
volume_melted
```

[ ]:
```
      Symbols               value
0     BBRI.JK 157,375,882,916.670
1     BBRI.JK 313,461,787,446.880
2     BBRI.JK 165,335,827,872.880
3     BBRI.JK 391,278,438,345.830
4     BBRI.JK 643,562,271,192.770
…       …                   …
2905  BRIS.JK  20,940,090,000.000
2906  BRIS.JK  21,223,942,500.000
2907  BRIS.JK   9,994,539,500.000
2908  BRIS.JK  68,731,942,500.000
2909  BRIS.JK  42,773,900,000.000

[2910 rows x 2 columns]
```

Misalkan kita ingin membandingkan rata-rata volume transaksi antara masing-masing harga saham. Rata-rata, manakah dari 3 saham tersebut yang memiliki rata-rata volume transaksi harian tertinggi?

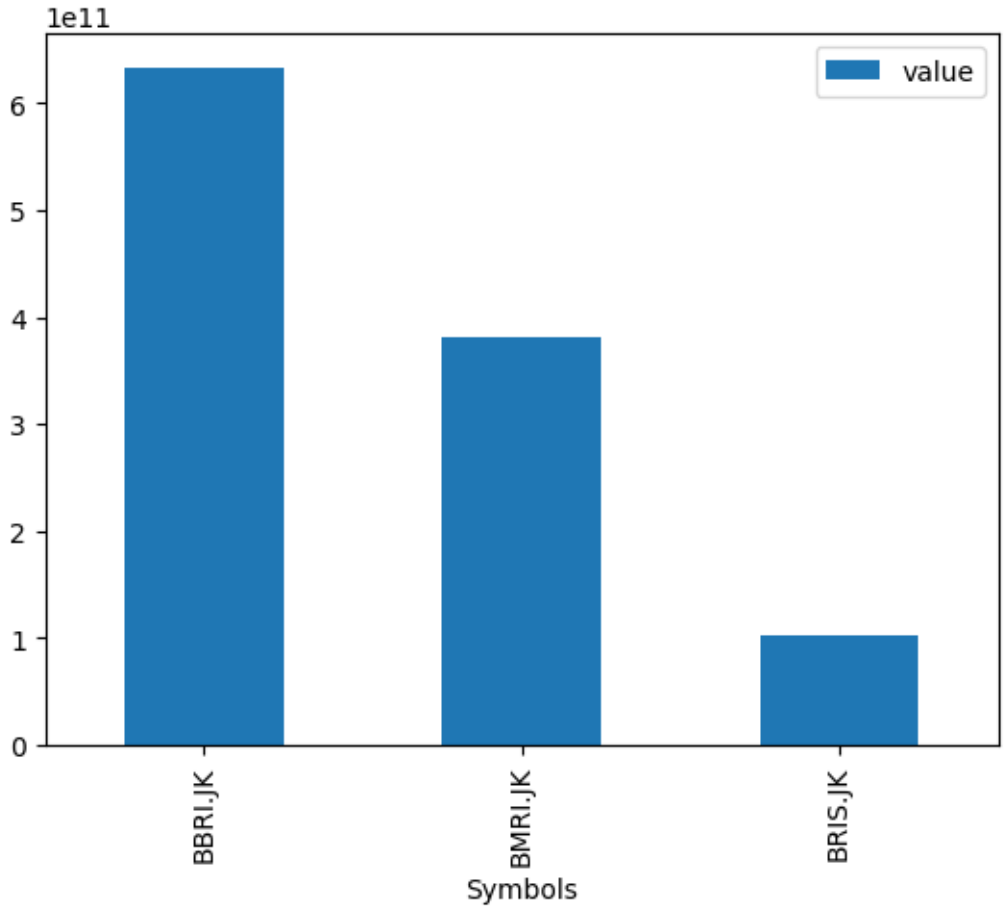
[ ]:
```
volume_melted.groupby(['Symbols']).mean().plot.bar();
```



6.1 Visualisasi Barchart untuk Perbandingan

Katakanlah kita ingin membandingkan rata-rata volume penjualan harian dari perusahaan-perusahaan tersebut. Untuk melakukan itu, kita perlu mengekstrak atribut volume dari kerangka data kita, dan menjalankan fungsi leleh:

[ ]:
```
volume_melted.groupby('Symbols').mean().plot.bar();
```



Jika kita membandingkan visualisasi dengan angka numerik, akan jauh lebih mudah untuk membandingkan volume rata-rata setiap saham. Sekarang mari kita pertimbangkan kerangka data berikut ini:

[ ]:
```
bbri = stock.xs('BBRI.JK', level='Symbols', axis=1)
bbri = bbri.round(2)
bbri['Close_Diff'] = bbri['Close'].diff()
bbri['Weekday'] = bbri.index.day_name()
bbri
```

[ ]:
```
Attributes  Adj Close     Close      High       Low      Open     Volume
Date
2020-01-02  3,772.730 4,410.000 4,410.000 4,360.000 4,400.000   41714100  \
2020-01-03  3,781.280 4,420.000 4,440.000 4,390.000 4,420.000   82898300
2020-01-06  3,738.510 4,370.000 4,390.000 4,320.000 4,360.000   44225100
2020-01-07  3,764.170 4,400.000 4,410.000 4,380.000 4,410.000  103948100
2020-01-08  3,747.060 4,380.000 4,400.000 4,340.000 4,380.000  171751200
…                 …         …         …         …         …         …
```

```
2023-12-15  5,550.000 5,550.000 5,600.000 5,550.000 5,575.000  252448800
2023-12-18  5,500.000 5,500.000 5,575.000 5,500.000 5,575.000  102780900
2023-12-19  5,550.000 5,550.000 5,550.000 5,450.000 5,450.000  135207300
2023-12-20  5,550.000 5,550.000 5,700.000 5,550.000 5,700.000  138470900
2023-12-21  5,575.000 5,575.000 5,600.000 5,525.000 5,550.000   99049600

Attributes  Close_Diff    Weekday
Date
2020-01-02         NaN   Thursday
2020-01-03      10.000     Friday
2020-01-06     -50.000     Monday
2020-01-07      30.000    Tuesday
2020-01-08     -20.000  Wednesday
…                   …          …
2023-12-15       0.000     Friday
2023-12-18     -50.000     Monday
2023-12-19      50.000    Tuesday
2023-12-20       0.000  Wednesday
2023-12-21      25.000   Thursday

[970 rows x 8 columns]
```

Berikan perhatian khusus pada bagaimana kolom Close_Diff dibuat. Ini adalah selisih antara nilai Penutupan harga saham pada hari tertentu dan hari berikutnya.

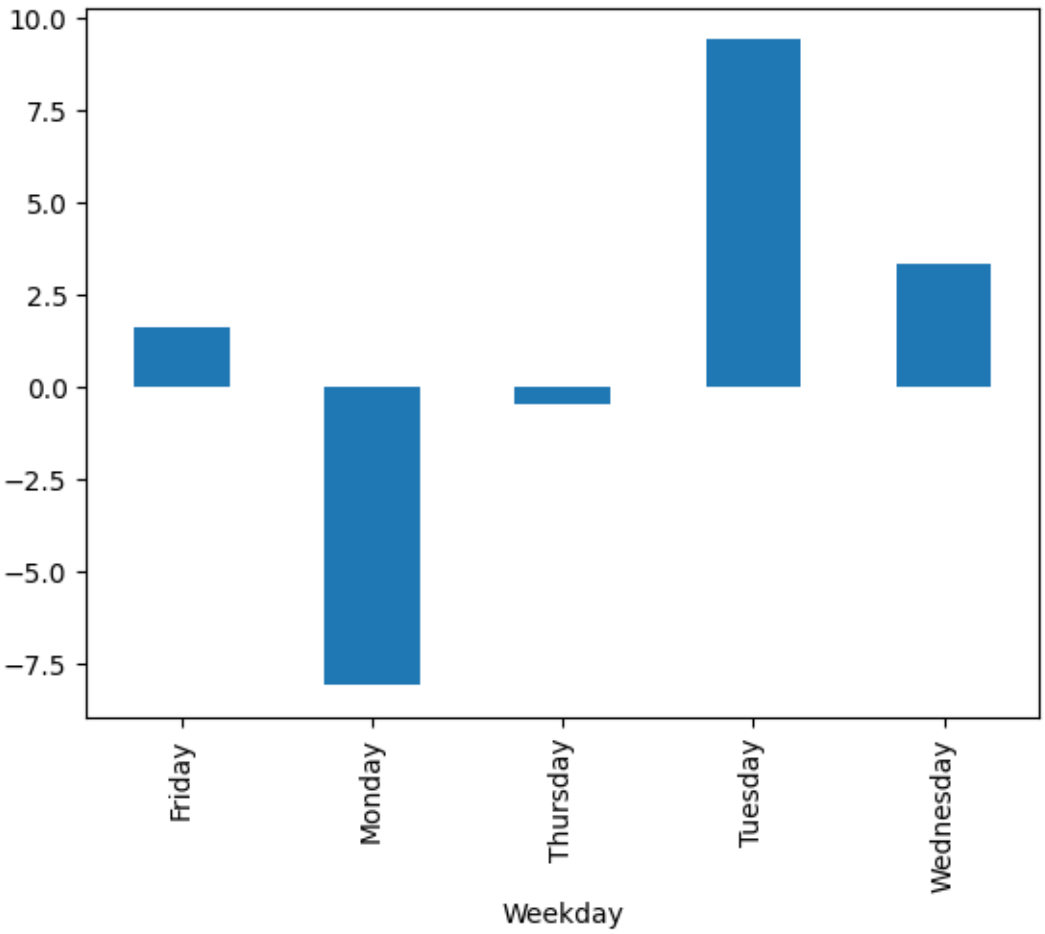
Misalkan kita ingin membandingkan `Close_Diff` antara setiap Hari Kerja; Rata-rata, apakah hari Selasa mencatat perbedaan yang lebih tinggi antara harga Penutupan saham BRI dibandingkan hari Kamis?

[ ]:
```
bbri.groupby('Weekday').mean(numeric_only=True)
```

[ ]:
```
Attributes  Adj Close     Close      High       Low      Open          Volume
Weekday
Friday      4,049.947 4,348.636 4,400.580 4,298.007 4,351.735 170,257,775.441  \
Monday      4,029.579 4,330.059 4,382.903 4,283.034 4,341.923 147,363,182.086
Thursday    4,021.308 4,325.322 4,379.851 4,278.062 4,332.870 166,182,807.247
Tuesday     4,033.065 4,330.981 4,386.888 4,284.111 4,332.204 163,620,687.776
Wednesday   4,030.649 4,331.024 4,384.109 4,282.291 4,338.537 169,151,324.725

Attributes  Close_Diff
Weekday
Friday          1.634
Monday         -8.090
Thursday       -0.457
Tuesday         9.430
Wednesday       3.370
```

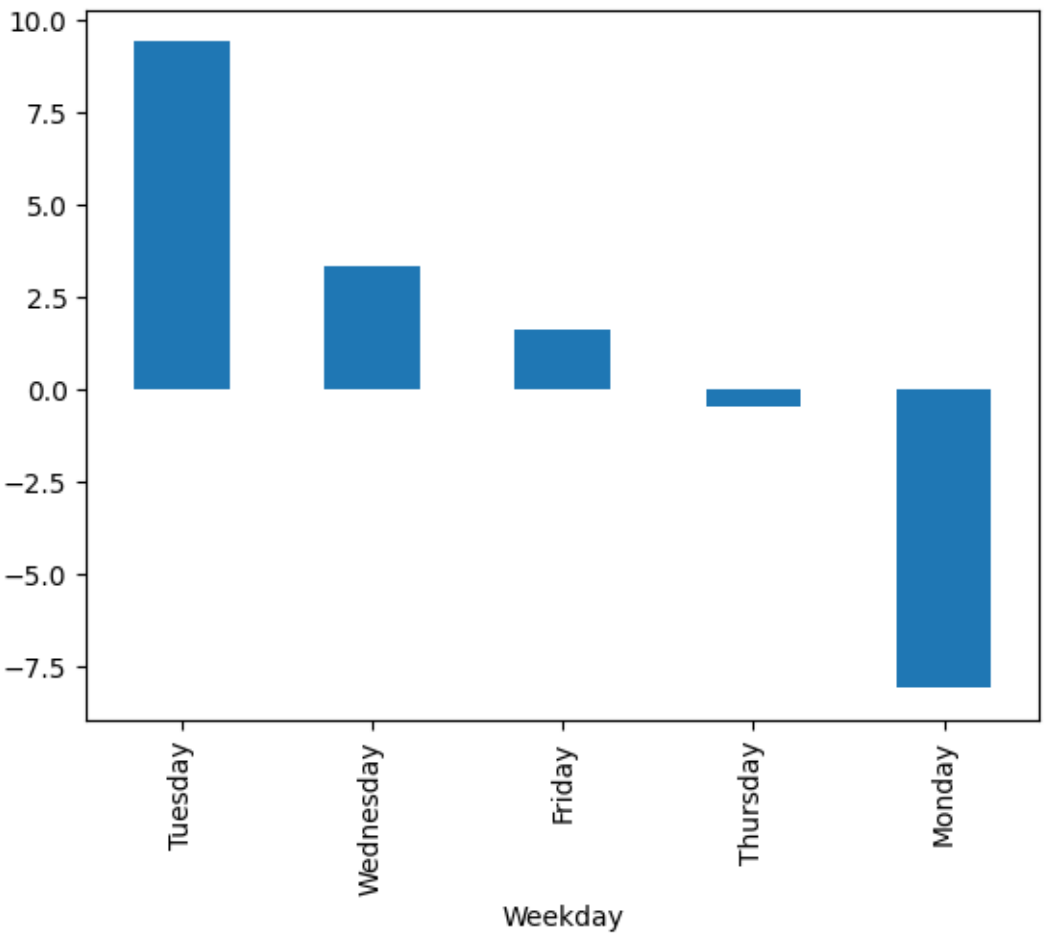
Now to create the same bar chart using `plot` function:

[ ]:
```
bbri.groupby('Weekday').mean(numeric_only=True)['Close_Diff'].plot.bar();
```



Kita juga dapat meningkatkan efisiensi visualisasi dengan nilai rata-rata volume transaksi terlebih dahulu, sehingga batang dari plot kita akan disusun berdasarkan nilainya, bukan berdasarkan urutan abjad pada hari kerja.

[ ]:
```
# berdasarkan urutan nilai terbesar - terkecil
bbri.groupby('Weekday').mean(numeric_only=True)['Close_Diff'].
↪sort_values(ascending=False).plot.bar();
```

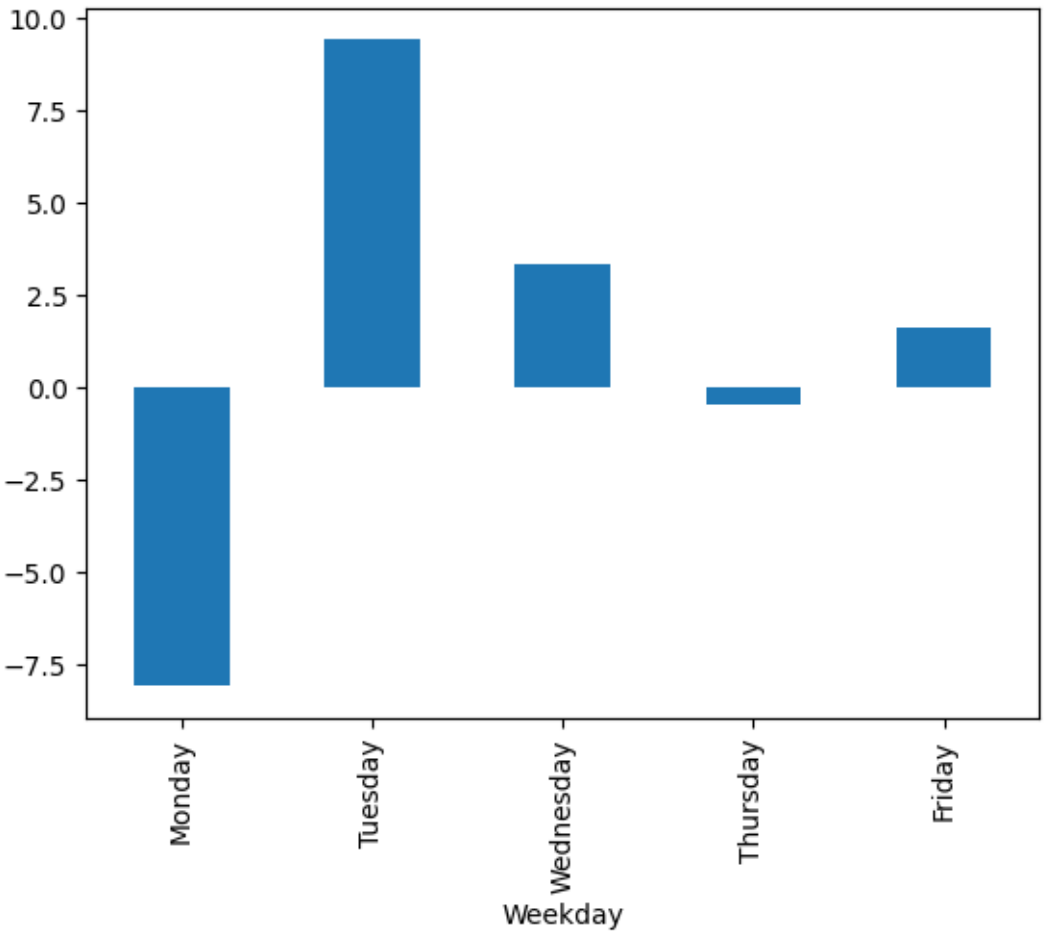


Kita juga dapat membuat indeks yang diurutkan secara manual dengan menentukan urutan hari itu.

[ ]:
```
wday = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]

bbri_wday = bbri.groupby('Weekday').mean(numeric_only=True)['Close_Diff']
```

[ ]:
```
bbri_wday.index = pd.CategoricalIndex(bbri_wday.index,\
                                      categories=wday,\
                                      ordered=True)
```

[ ]:
```
bbri_wday.sort_index().plot.bar();
```



6.2 Menggunakan Barchart Group

Dengan menggunakan `closingprice`, kita dapat mencoba memvisualisasikan menggunakan barchart yang dikelompokkan untuk membandingkan harga penutupan setiap bulan untuk kuartal pertama tahun 2019 dan membandingkannya untuk 3 saham tersebut.

Pertama, lihat `harga penutupan` dan pastikan tidak ada nilai yang hilang pada data. Jika sudah, isi dengan cara yang sesuai

[ ]:
```
closingprice = stock['Close']
closingprice.head()
```

[ ]:
```
Symbols      BBRI.JK   BMRI.JK  BRIS.JK
Date
2020-01-02 4,410.000 3,875.000  332.000
2020-01-03 4,420.000 3,862.500  328.000
2020-01-06 4,370.000 3,800.000  324.000
2020-01-07 4,400.000 3,800.000  318.000
```

```
2020-01-08 4,380.000 3,750.000  312.000
```

[ ]:
```
## Write your solution code here
# Buat kolom baru bernama 'Month', yang menunjukkan nama bulan pada tanggal␣
 ↪tersebut

closingprice.loc[:, 'Month'] = closingprice.index.month_name()
```

[ ]:
```
closingprice
```

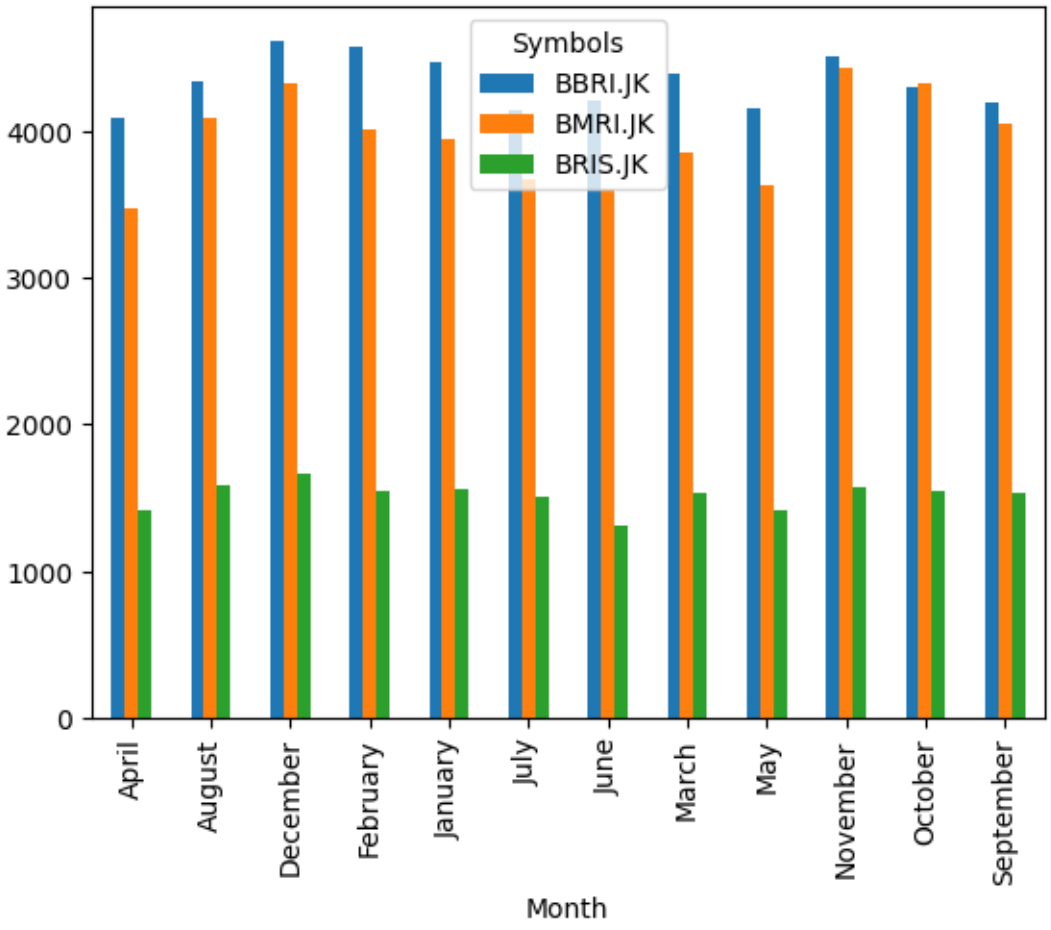
[ ]:
```
Symbols      BBRI.JK   BMRI.JK   BRIS.JK     Month
Date
2020-01-02 4,410.000 3,875.000   332.000   January
2020-01-03 4,420.000 3,862.500   328.000   January
2020-01-06 4,370.000 3,800.000   324.000   January
2020-01-07 4,400.000 3,800.000   318.000   January
2020-01-08 4,380.000 3,750.000   312.000   January
…                …        …        …         …
2023-12-15 5,550.000 5,900.000 1,700.000  December
2023-12-18 5,500.000 5,925.000 1,695.000  December
2023-12-19 5,550.000 5,975.000 1,705.000  December
2023-12-20 5,550.000 5,925.000 1,755.000  December
2023-12-21 5,575.000 5,975.000 1,690.000  December

[970 rows x 4 columns]
```

Setelah kita memiliki kolom `Month`, mari kelompokkan berdasarkan Bulan dan lihat DataFrame yang dihasilkan

[ ]:
```
average_closing = closingprice.groupby('Month').mean()
average_closing
```

[ ]:
```
Symbols     BBRI.JK   BMRI.JK   BRIS.JK
Month
April     4,081.733 3,475.367 1,419.053
August    4,334.204 4,088.415 1,581.646
December  4,610.789 4,328.618 1,662.105
February  4,572.727 4,013.961 1,540.818
January   4,467.738 3,941.964 1,563.738
July      4,135.952 3,672.857 1,502.476
June      4,199.750 3,603.344 1,307.487
March     4,385.581 3,847.384 1,532.256
May       4,156.159 3,627.935 1,408.116
November  4,505.000 4,430.172 1,576.552
October   4,300.305 4,316.921 1,546.829
September 4,194.843 4,051.366 1,528.023
```

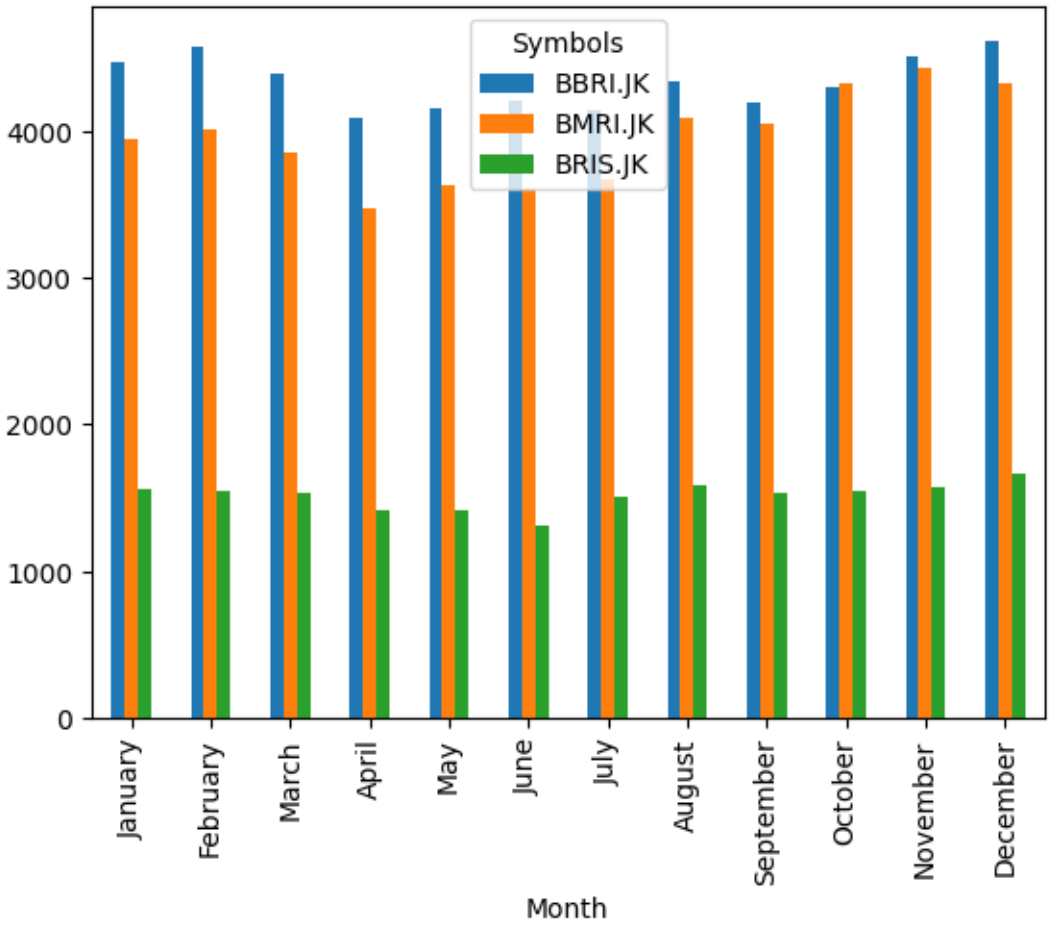
[ ]:
```
average_closing.sort_index().plot.bar();
```



Namun, jika ingin menyusun ulang bulannya, kita harus menetapkan indeks sebagai nilai kategorikal yang diurutkan (Lihat materi Analisis Data Eksplorasi jika Anda perlu mengingatnya).

[ ]:
```
months= ['January', 'February', 'March', 'April', 'May', 'June', 'July',␣
 ↪'August', 'September', 'October', 'November', 'December']
```

[ ]:
```
average_closing.index = pd.CategoricalIndex(average_closing.index,\
                                             categories=months,\
                                             ordered=True)

average_closing.sort_index().plot.bar();
```



Referensi lengkap ke dokumentasi resmi tentang metode ini berada di luar cakupan buku pelajaran ini, tapi layak untuk dibaca.

6.3 Menggabungkan `agg` dan `groupby`

Sejauh ini, kami telah menjelajahi beberapa perangkat agregasi panda, seperti:

- `pd.crosstab()`
- `pd.pivot_table()`

Dalam bab ini, kita akan menjelajahi alat agregasi panda lainnya:

- agregasi `groupby`.

Pembahasan:

(`pivot_table` & `pd.crosstab` kesetaraan)

Metode `pivot_table` dan fungsi `crosstab` keduanya dapat menghasilkan hasil yang sama persis dengan bentuk yang sama. Keduanya berbagi parameter; `indeks`, `kolom`, `nilai`, dan `aggfunc`.

Perbedaan utama yang terlihat adalah `crosstab` adalah sebuah fungsi dan bukan metode

DataFrame. Ini memaksa Anda untuk menggunakan kolom sebagai Seri dan bukan nama string untuk parameternya.

1. Misalkan Anda ingin membandingkan jumlah total transaksi selama Hari Kerja setiap periode kuartal. Buat `pivot_table` yang menyelesaikan masalah!
2. Cobalah untuk mereproduksi hasil yang sama dengan menggunakan `crosstab`
3. Bagaimana jika, alih-alih membandingkan total transaksi, Anda ingin membandingkan total pendapatan dari periode yang sama? Gunakan `pivot_table` dan `crosstab` sebagai solusinya. Diskusikan dengan teman anda, metode mana yang lebih relevan dalam kasus ini?

Perhatikan kelompok berikut berdasarkan operasi:

[ ]:
```
stock.stack().reset_index().groupby('Symbols').mean(numeric_only=True)
```

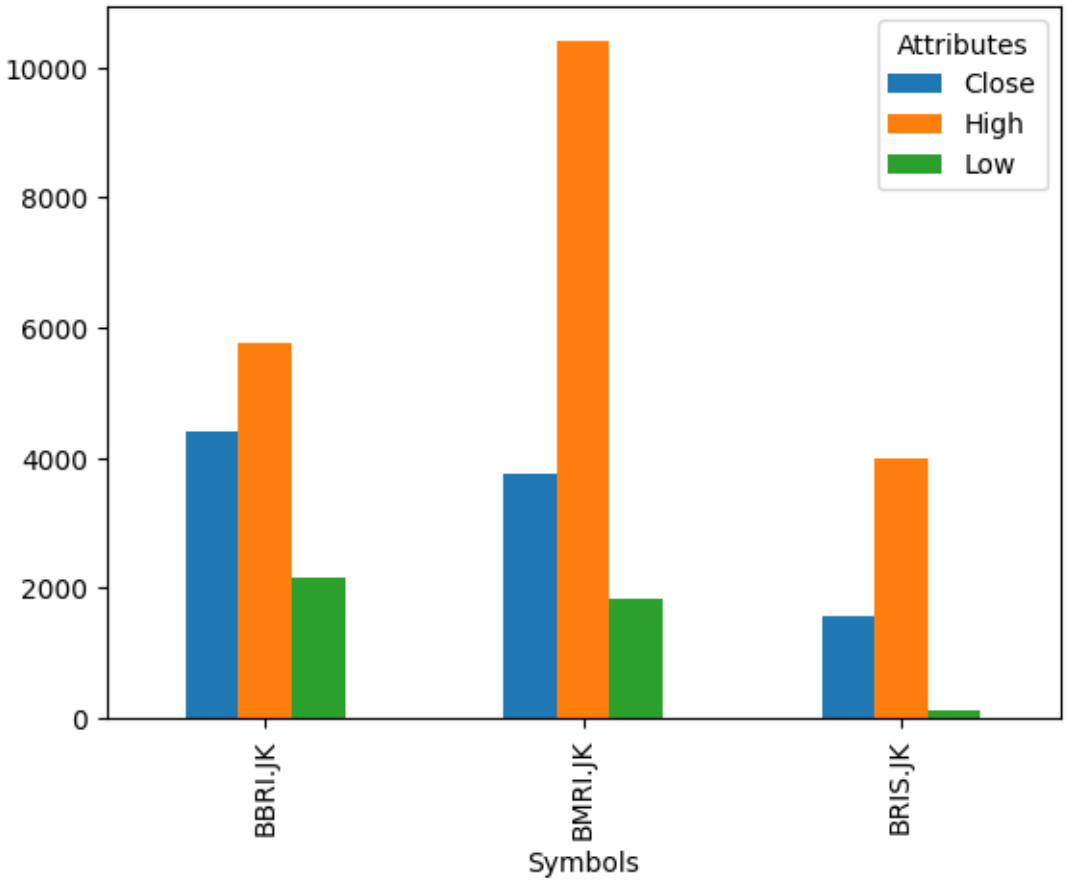
[ ]:
```
Attributes  Adj Close     Close      High       Low      Open          Volume
Symbols
BBRI.JK     4,029.681 4,330.575 4,384.187 4,282.450 4,336.708 163,307,060.979
BMRI.JK     3,704.355 3,957.539 4,007.769 3,908.425 3,960.150 106,337,189.669
BRIS.JK     1,498.105 1,516.107 1,553.779 1,487.457 1,518.201  66,685,960.537
```

[ ]:
```
stock.stack().reset_index().groupby('Symbols').agg({
    'Close': 'mean',
    'High': 'max',
    'Low': 'min'
})
```

[ ]:
```
Attributes     Close       High       Low
Symbols
BBRI.JK    4,330.575  5,750.000 2,160.000
BMRI.JK    3,957.539 10,400.000 1,830.000
BRIS.JK    1,516.107  3,980.000   135.000
```

Katakanlah kita ingin mengetahui sekilas harga saham maksimum, harga saham minimum, dan rata-rata harga penutupan dari ketiga perusahaan tersebut. Untuk melakukannya, kita perlu menggabungkan `groupby` dengan `agg` dan memetakan setiap kolom dengan fungsi agregasinya.

[ ]:
```
stock.stack().reset_index().groupby('Symbols').agg({
    'Close': 'median',
    'High': 'max',
    'Low': 'min'
}).plot.bar();
```



Knowledge Check: Menggunakan `plot`

Pertimbangkan kerangka data berikut:

[ ]:
```
import datetime

stock['YearMonth'] = pd.to_datetime(stock.index.date).to_period('M')
monthly_closing = stock.groupby('YearMonth').mean().loc[:,['Close','Low',␣
↪'High']]
monthly_closing.head()
```

[ ]:
```
Attributes     Close                          Low                          High
Symbols      BBRI.JK   BMRI.JK BRIS.JK   BBRI.JK   BMRI.JK BRIS.JK   BBRI.JK
YearMonth
2020-01    4,550.000 3,847.727 317.909 4,513.182 3,805.114 314.727 4,576.818  \
2020-02    4,477.500 3,862.500 296.900 4,439.500 3,807.500 293.000 4,528.500
2020-03    3,440.952 2,916.667 196.857 3,362.857 2,847.976 187.286 3,615.714
2020-04    2,785.714 2,260.238 194.714 2,727.143 2,196.190 191.095 2,878.095
2020-05    2,530.625 2,045.312 267.500 2,468.750 2,004.062 254.625 2,590.000
```

```
Attributes
Symbols      BMRI.JK BRIS.JK
YearMonth
2020-01    3,892.045 322.909
2020-02    3,900.000 303.800
2020-03    3,025.238 212.810
2020-04    2,337.738 202.286
2020-05    2,096.875 284.500
```

Manakah dari berikut ini yang merupakan plot yang sesuai untuk digunakan?

- ☐ Line plot -> .plot()
- ☐ Scatter plot -> .plot.scatter(x=? , y=?)
- ☐ Bar plot -> .plot.bar()
- ☐ Box plot -> .plot.box()

[ ]:
```
## Your code below
```