Unit 38.

Time Series Data

Unit learning objective (1/3)

Learning objectives

- ✓ Be able to determine whether it is an analysis target using time series data when a data frame is presented.
- ✓ Be able to explain the two data types of time series data.
- ✓ Be able to create date, time, interval, and array data as a time series data type.
- ✓ Be able to index and slice a data frame using DatetimeIndex.
- ✓ Be able to obtain a moving average, a representative descriptive statistic of time series data.
- ✓ Be able to visualize data as an area graph for comparison of two or more series data.

Unit learning objective (2/3)

Learning overview

- ✓ Learn real-world applications of time series data analysis
- ✓ Learn types and elements of time series data types
- ✓ Learn how to generate time series data (date, time, interval, array of time series data)
- ✓ Learn how to index and slice time series data
- ✓ Learn basic descriptive statistics using time series data functions

Concepts you will need to know from previous units

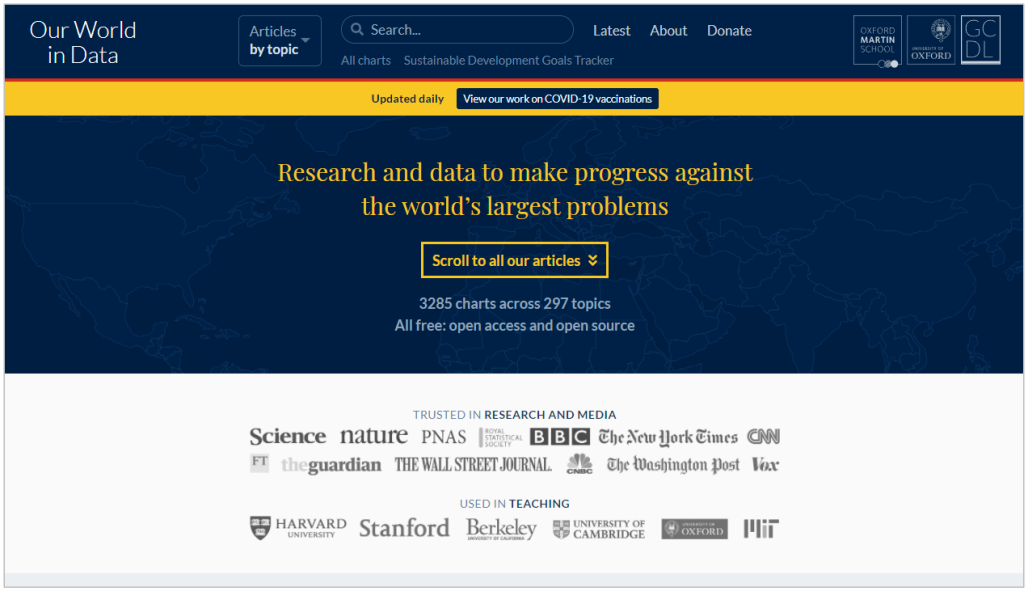
- ✓ How to index and slice of data frames and series
- ✓ How to create an external file as a dataframe
- ✓ How to draw graphs using Matplotlib
- ✓ How to view the descriptive statistics summary of the datafram using Dataframe.info() function
- ✓ How to retrieve missing data and replacing data

Keywords

- Time Series Data
- Timestamp
- Period
- Moving Average
- Area Graph
- Categorical Data

Mission

- In the current global Seafood Production business, are most of the fish caught and processed directly by fishermen, or are they processed through aquaculture?
- This may be different depending on the geographical location of each country or the difference in food culture. An organization called Our World in Data provides related data. Based on this, let's compare what type of raw material supply occupies the most in the current global Seafood Production business.
- In addition, since this data provides data for each country, let's search for the country you want and create related data statistics separately.

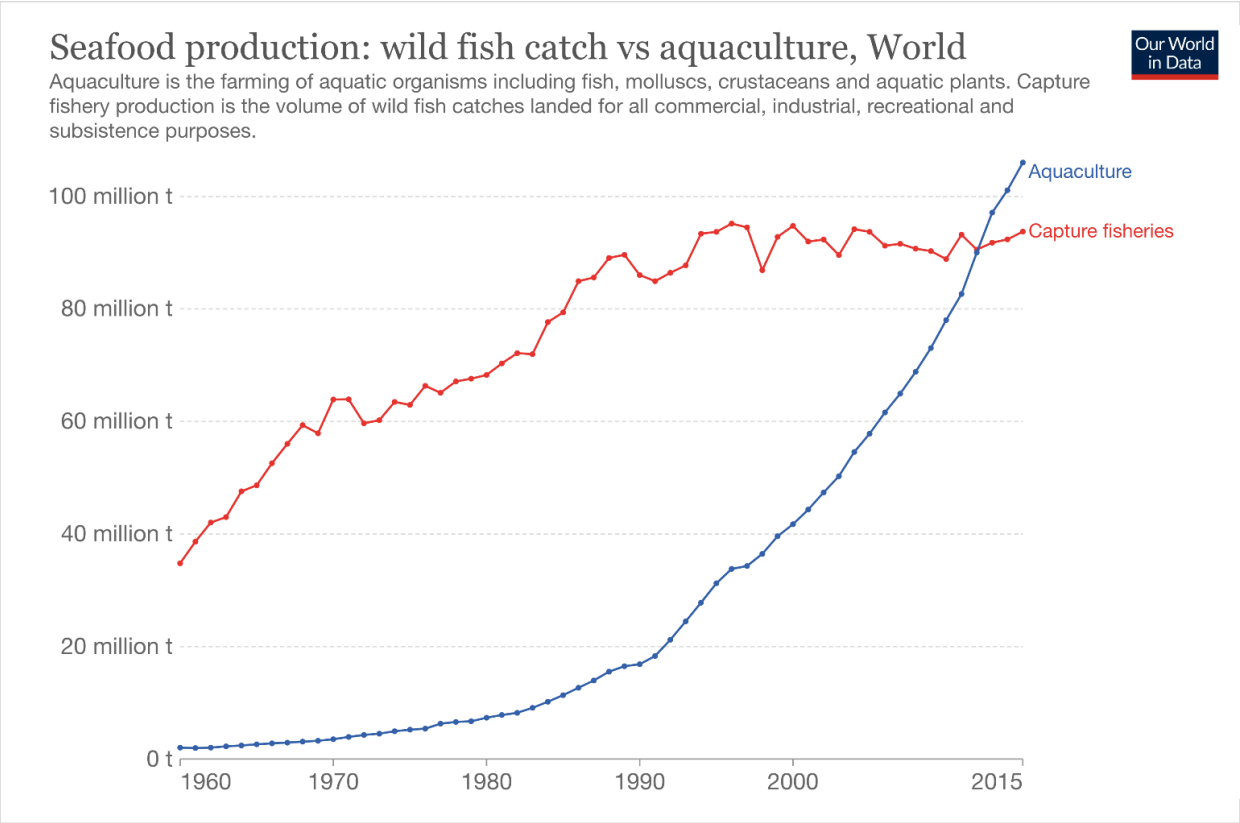


https://ourworldindata.org/

This mission must comply with the tasks below.

- Data resource location: https://ourworldindata.org/fish-and-overfishing
- Data as of November 6, 2021 are downloaded and provided for practice.
  df = pd.read_csv("./data//fish/capture-fisheries-vs-aquaculture.csv")
- Convert the csv file to a data frame.
- Delete unnecessary columns in the data frame.
- Convert the year data to a time series data type and convert it to an index.
- Transform country data into categorical data.
- Check if NaN data exists and replace it.
- Use visualization tools to see trends in aquaculture and direct catch data around the world (All countries) from the 1960s to the present.
- Search for 3 or more individual countries and compare them through visualization.

- The goal of this mission is to produce the same type of results as the graph below.
- The graph below was created by data analysis experts.



Source: Food and Agriculture Organization of the United Nations (via World Bank)

Key concept

1. What is time series data?

- Time series data is a set of time-organized values collected over a certain time or period.
- It can be explained a little more technically:

> "A time series is data about one or more variables measured over a period of time at specific intervals."

- Machine learning technology is widely used for business predictive analysis. It analyzes business problems such as stock price, budget, sales and asset flow, forecast maintenance and sales forecasting, and predicts future indicators by composing related data such as trend, periodic theory, and seasonality together.
- Pandas was created to handle financial data, and time series data analysis can be the core of data analysis using Pandas.
- In addition to the financial field, time series data analysis is used in various fields.

  - Medicine
  - Meteorology
  - Astronomy
  - Economics
  - IOT

2. Time Series Data and Analysis used in the Real World

- The image below is the Manufacturing Purchasing Management Index (PMI) of the United States. Values above 50 indicate economic expansion, and values below 50 indicate contraction. It is a meaningful graph that can be a leading indicator of overall economic performance. This is an example where time series data is used in the real world. Even the predicted and actual figures are compared, and this can also be generated by analyzing time series data accumulated in the past.
- https://www.markiteconomics.com/Public/Release/PressReleases

http://www.Investing.com

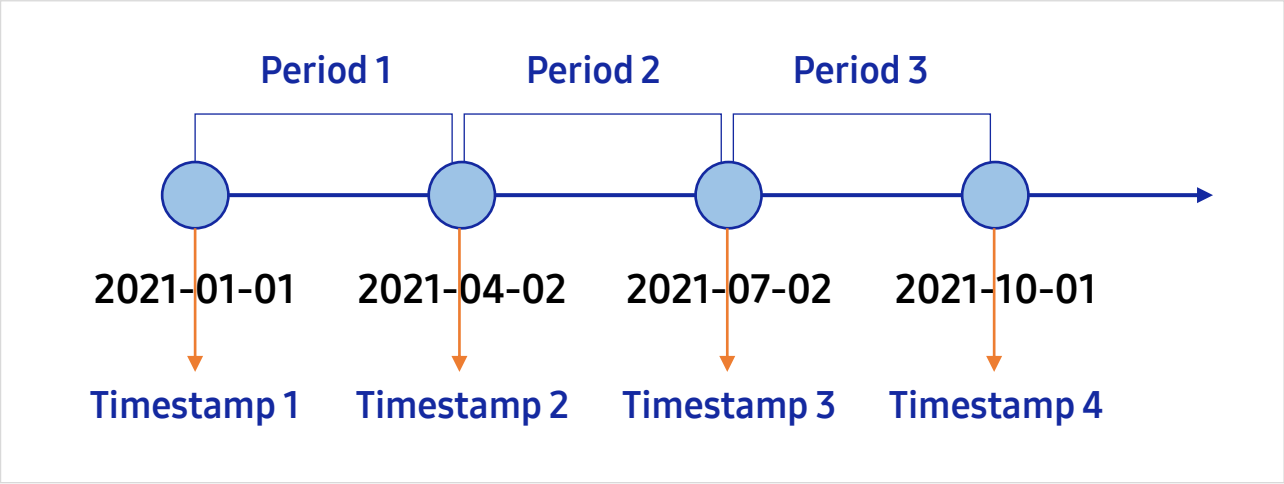
2. Time Series Data and Analysis used in the Real World

- Time series data are sorted by time order, so there are continuous observation values. There is a high probability that each data value has a correlation with each other.
- To understand time series data, you should first understand how pandas represents dates, times, and intervals. Pandas has many functions that can be used when transforming data at different frequencies or using a calendar that reflects business days and holidays for financial calculations.
- Pandas was created to handle financial data. When analyzing financial data, changing time series data into an index within a data frame and using it has many advantages.
- As it will be explained in the example code, when most of the external data is called and the columns of date are checked, there are many cases of string or object type. It starts with converting to Pandas time series data type for effective analysis.

3. Two Time Series Data Types in Pandas

- timestamp: one point in time
- period: two time points. i.e., a constant period between two timestamps

  - Multiple timestamps are gathered to create an array and become a Datetimeindex.

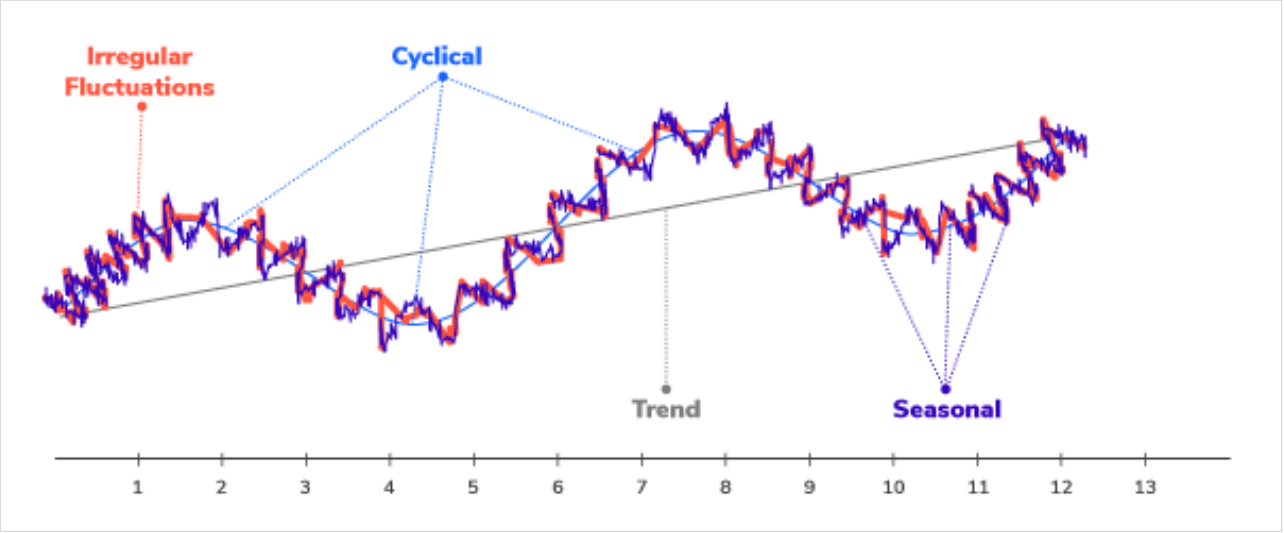


4. Elements of Time Series Data

Trend	A fluctuating pattern in data that appear in the long term (The values of the data are increased or decreased in a reasonably predictable pattern)
Seasonal	A fluctuating pattern in data that appears in a period of unit time such as a week, month, quarter, or half year, etc. (The patterns of the data are repeated over a specific period.)
Cycle	A long-term fluctuation rather than fixed periods that appear in a period of at least two years (The values of the data exhibit rises and falls that are not of a fixed frequency often due to economic conditions.)
Random	A completely irregular pattern that does not belong to the above three categories

4. Elements of Time Series Data

- You need to decompose the components of each of the four representative time series data to discover insights.



- This analysis of time series data is called a time series additive model (Time series additive model).
- Trend factor + Cycle factor + Seasonal factor + Irregular/Random factor

$$Y_t = T_t + C_t + S_t + I_t$$

5. Creating an Interval of Date, Time, Frequency and Time

5.1. Create date and time with python datetime class

- The datetime object is part of Python's datetime library. This class is used to express general and various patterns, such as a specific point in time using both date and time, only the date minus the time, or only the time.
  - https://docs.python.org/ko/3/library/datetime.html#module-datetime
- Most of the functions supported by the date and time class of the datetime library are supported.
- The disadvantage is that it reduces the precision needed for massive computations on time series data. The datetime class receives year, month, day, hour, minute, second, microsecond, and time zone as arguments. The time argument is not a required valu. If empty, 0 is returned as a default value.
  - You can use Pandas by converting datetime to timestamp object.

5. Creating an Interval of Date, Time, Frequency and Time

5.1. Create date and time with python datetime class

```
1 import pandas as pd
2 import numpy as np
3 import datetime
4
5 from datetime import date, datetime, time, timezone
6
7 datetime(2021,7,7)
```

```
datetime.datetime(2021, 7, 7, 0, 0)
```

Line 7

- We need at least 3 parameters corresponding to year, month and day. Hour and minute are returned as 0 by default.

5. Creating an Interval of Date, Time, Frequency and Time

5.1. Create date and time with python datetime class

```
1  datetime(2021,7,7,15,50)
```

```
datetime.datetime(2021, 7, 7, 15, 50)
```

Line 1

- In case of setting the parameters of 15:50 in the hour and minute

```
1  d = date(2021,7,7)
2  t = time(13,26,10)
3  datetime.combine(d,t)
4
```

```
datetime.datetime(2021, 7, 7, 13, 26, 10)
```

Line 4

- If you use the combine() method of the datetime class, you can create a datetime object using an existing date or time object.

5. Creating an Interval of Date, Time, Frequency and Time

5.1. Create date and time with python datetime class

```
1 datetime.now()
```

```
datetime.datetime(2021, 11, 7, 11, 1, 6, 305147)
```

Line 1

- Current date and time, local time

```
1 datetime.now(timezone.utc)
```

```
datetime.datetime(2021, 11, 7, 2, 1, 6, 743268, tzinfo=datetime.timezone.utc)
```

Line 1

- If a time zone is used as a parameter in the now method, a datetime object applied with the time zone is created.

5. Creating an Interval of Date, Time, Frequency and Time

5.1. Create date and time with python datetime class

```
1 datetime.now().date()
```

```
datetime.date(2021, 11, 7)
```

Line 1

- Get the current time and date and returns only the date.

```
1 datetime.now().time()
```

```
datetime.time(11, 1, 7, 587524)
```

Line 1

- Get the current time and date and returns only the time.

5. Creating an Interval of Date, Time, Frequency and Time

5.2. Create a point in time with Pandas timestamp

```
pandas.Timestamp()
```

- The date and time created with datetime can also be created with pandas.Timestamp(). The difference is that the datatype is datetime64, which has higher precision than Python datetime.
- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Timestamp.html

5. Creating an Interval of Date, Time, Frequency and Time

5.2. Create a point in time with Pandas timestamp

pandas.Timestamp()

```
1 pd.Timestamp('2017, 1, 15')
```

```
Timestamp('2017-01-15 00:00:00')
```

Line 1

- Both date and time can be set when creating.

```
1 pd.Timestamp('2017, 5, 3, 12:00')
```

```
Timestamp('2017-05-03 12:00:00')
```

Line 1

- Created by specifying the time.

5. Creating an Interval of Date, Time, Frequency and Time

5.2. Create a point in time with Pandas timestamp

pandas.Timestamp()

```
1 pd.Timestamp('12:00')
```

```
Timestamp('2021-11-07 12:00:00')
```

Line 1

- If only time is specified, today's date is applied as default.

5. Creating an Interval of Date, Time, Frequency and Time

5.3. Create time intervals with Pandas Timedelta

- We have learned to create a moment of date and time so far. Now, we will learn how to express time intervals. timedelta is a subclass of Python's datetime.timedelta and is also supported by Pandas.
- You can use Pandas' Timedelta class for time intervals. The time interval is important for time series data analysis because it is necessary when determining the number of days or analyzing by a specific time interval.

```
pandas.Timedelta(value, unit=None, **kwargs)
```

- https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Timedelta.html

5. Creating an Interval of Date, Time, Frequency and Time

5.3. Create time intervals with Pandas Timedelta

pandas.Timedelta(value, unit=None, **kwargs)

```
1 import pandas as pd
2 import datetime
3 from datetime import datetime
4
5 my_birth = datetime(1973,7,7)
6 today = datetime.today()
7 tommorrow = today + pd.Timedelta(days = 1)
8
9 print(tommorrow - my_birth)
```

```
17656 days, 10:58:46.085481
```

Line 5, 6, 7, 9

- 5: Create a specific date with datetime.
- 6: Create today's date.
- 7: Calculate the day plus one day using timedelta.
- 9: How long is tomorrow from my birthday?

5. Creating an Interval of Date, Time, Frequency and Time

5.3. Create time intervals with Pandas Timedelta

pandas.Timedelta(value, unit=None, **kwargs)

```
1 my_birth-today
```

```
datetime.timedelta(days=-17656, seconds=46873, microseconds=914519)
```

Line 1

- When calculating the number of days between two dates, data is returned in timedelta format.

5. Creating an Interval of Date, Time, Frequency and Time

5.3. Create time intervals with Pandas Timedelta

pandas.Timedelta(value, unit=None, **kwargs)

```
1 today= datetime.today()
2 data = today + pd.Timedelta(hours = 10)
3
4 print(today)
5 print(data)
```

```
2021-11-07 10:58:47.559180
2021-11-07 20:58:47.559180
```

Line 2, 5

- 2: The difference in time can also be calculated arithmetically.
- 5: You can see that 10 hours are added to today.

5. Creating an Interval of Date, Time, Frequency and Time

5.4. Represent a period with Period

pandas.Period()

- https://pandas.pydata.org/docs/reference/api/pandas.Period.html?highlight=pd%20period
- Most of time series data analysis is event analysis for a specific time interval. An example is an analysis of a company's sales over a specific period of time. However, when analyzing events by grouping multiple periods, it is difficult to use only timestamps. Pandas provides a standardized time interval through a class called Period to facilitate this kind of data organization and calculation.
- Period creates a period based on a specified frequency such as daily, weekly, monthly, yearly, quarterly, etc. and provides Timestamp that indicates the start time and end time.
- A period can be created using a timestamp corresponding to a reference point and a frequency that indicates the period.
- If you create a period corresponding to a month based on July 1973, you can do it as follows.

```
1 import pandas as pd
2
3 special_day = pd.Period('1973-7', freq='M')
4 special_day
```

```
Period('1973-07', 'M')
```

5. Creating an Interval of Date, Time, Frequency and Time

5.4. Represent a period with Period

pandas.Period()

- pd.Period has attributes showing start time and end time.
- https://pandas.pydata.org/docs/reference/api/pandas.Period.start_time.html#pandas.Period.start_time

```
1 special_day.start_time
```

```
Timestamp('1973-07-01 00:00:00')
```

Line 1

- Return the start time of that point in time.

```
1 special_day.end_time
```

```
Timestamp('1973-07-31 23:59:59.999999999')
```

Line 1

- Return the end time of that point in time.

5. Creating an Interval of Date, Time, Frequency and Time

5.4. Represent a period with Period

pandas.Period()

Period can be shifted through simple arithmetic operation. You can create a new period object by shifting the frequency. The example below is an example of +2 (shifting two months) because the frequency of special_day is one month.

```
1 n= special_day + 2
2
3
4 print(special_day)
5 print(n)
```

```
1973-07
1973-09
```

Line 1

- You cannot understand that adding 2 makes two months shifted just because 2 means two months. You should understand the way that the period is shifted by the unit that created the period.

5. Creating an Interval of Date, Time, Frequency and Time

5.4. Represent a period with Period

pandas.Period()

```
1 n
```

```
Period('1973-09', 'M')
```

```
1 n.start_time, n.end_time
```

```
(Timestamp('1973-09-01 00:00:00'), Timestamp('1973-09-30 23:59:59.999999999'))
```

Line 1

- See the results, you can see that Pandas is properly judging the full date of September 1973 (there are 30 days).

6. Time Series Data Indexing and Basic Use

6.1. Convert other data types to time series data Timestamp and indexing with DatetimeIndex

- You can convert data type to datetime64 type with timestamp with pandas.to_datetime().

```
pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, format=None, exact=True,
unit=None, infer_datetime_format=False, origin='unix', cache=True)
```

- https://pandas.pydata.org/docs/reference/api/pandas.to_datetime.html?highlight=to_datetime#pandas.to_datetime
- The key to working with time series data in Pandas is indexing using DatetimeIndex objects.
- The indexing is very useful, it automatically sorts data based on date and time.

6. Time Series Data Indexing and Basic Use

6.1. Convert other data types to time series data Timestamp and indexing with DatetimeIndex

pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, format=None, exact=True, unit=None, infer_datetime_format=False, origin='unix', cache=True)

```
1 import pandas as pd
2 wti = pd.read_csv("./data//wti/DCOILWTICO.csv")
3 wti.head()
```

	DATE	DCOILWTICO
0	2016-11-01	46.66
1	2016-11-02	45.32
2	2016-11-03	44.66
3	2016-11-04	44.07
4	2016-11-07	44.88

Line 2

- As example data, we use the international crude oil price data provided by the FED.

6. Time Series Data Indexing and Basic Use

6.1. Convert other data types to time series data Timestamp and indexing with DatetimeIndex

```
pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, format=None, exact=True,
unit=None, infer_datetime_format=False, origin='unix', cache=True)
```

You can see that the data type of the date column is object.

```
1 wti.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1305 entries, 0 to 1304
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   DATE        1305 non-null   object
 1   DCOILWTICO  1305 non-null   object
dtypes: object(2)
memory usage: 20.5+ KB
```

6. Time Series Data Indexing and Basic Use

6.1. Convert other data types to time series data Timestamp and indexing with DatetimeIndex

pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, format=None, exact=True, unit=None, infer_datetime_format=False, origin='unix', cache=True)

```
1 wti['N_DATE'] = pd.to_datetime(wti['DATE'])
2 wti['N_DATE']
```

```
0      2016-11-01
1      2016-11-02
2      2016-11-03
3      2016-11-04
4      2016-11-07
          ...
1300   2021-10-26
1301   2021-10-27
1302   2021-10-28
1303   2021-10-29
1304   2021-11-01
Name: N_DATE, Length: 1305, dtype: datetime64[ns]
```

Line 2

- The data type is changed to datetime64 type.

6. Time Series Data Indexing and Basic Use

6.1. Convert other data types to time series data Timestamp and indexing with DatetimeIndex

```
1 wti.head()
```

	DATE	DCOILWTICO	N_DATE
0	2016-11-01	46.66	2016-11-01
1	2016-11-02	45.32	2016-11-02
2	2016-11-03	44.66	2016-11-03
3	2016-11-04	44.07	2016-11-04
4	2016-11-07	44.88	2016-11-07

Line 1

A column named N_Date is created.

TIP

- Sometimes, in the process of converting to timestamp, an error occurs if the data cannot be converted. In this case, there is a way to force the conversion.
- If you use errors = 'coerce' parameter, NaT is forcibly assigned to data that cannot be converted.

6. Time Series Data Indexing and Basic Use

6.1. Convert other data types to time series data Timestamp and indexing with DatetimeIndex

pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, format=None, exact=True, unit=None, infer_datetime_format=False, origin='unix', cache=True)

```
1 df = wti.drop(['DATE'], axis=1)
2 df.head()
```

	DCOILWTICO	N_DATE
0	46.66	2016-11-01
1	45.32	2016-11-02
2	44.66	2016-11-03
3	44.07	2016-11-04
4	44.88	2016-11-07

Line 1

- Delete the DATE column that was the existing object data type.

6. Time Series Data Indexing and Basic Use

6.1. Convert other data types to time series data Timestamp and indexing with DatetimeIndex

```
pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, format=None, exact=True, unit=None, infer_datetime_format=False, origin='unix', cache=True)
```

```
1 df.set_index('N_DATE', inplace = True)
2 df.head()
```

N_DATE	DCOILWTICO
2016-11-01	46.66
2016-11-02	45.32
2016-11-03	44.66
2016-11-04	44.07
2016-11-07	44.88

Line 1

- Designate the newly created datetime64 column as the index of the data frame.

6. Time Series Data Indexing and Basic Use

6.1. Convert other data types to time series data Timestamp and indexing with DatetimeIndex

```
pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, format=None, exact=True,
unit=None, infer_datetime_format=False, origin='unix', cache=True)
```

Now, the dataframe is very easy to index or slice in chronological order because it supports the time series index class.

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1305 entries, 2016-11-01 to 2021-11-01
Data columns (total 1 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   DCOILWTICO  1305 non-null   object
dtypes: object(1)
memory usage: 20.4+ KB
```

Line 1

- This completes the data frame indexed by DatetimeIndex.

6. Time Series Data Indexing and Basic Use

6.2. Slicing using DatetimeIndex

In the previous lesson, we learned how to slice specific rows and columns of a dataframe. We check how easy and convenient it is to slice from time series data.

```
1 df['2016-11-04':'2016-11-10']
```

N_DATE	DCOILWTICO
2016-11-04	44.07
2016-11-07	44.88
2016-11-08	44.96
2016-11-09	45.20
2016-11-10	44.62

Line 1

- Select a specific date, and you can slice only the data you want very conveniently. Isn't it very intuitive?

7. Creating Time Series Data

7.1. Create time series of specific frequency from Timestamp array

- Time series data can be created not only in units of one unit but also in a specific time interval, that is, in a form with a specific frequency.
- You can use the freq parameter for pd.date_range() to create a time series with a frequency you want. The default is one unit.
- The principle is similar to creating an array of numbers with Python range() .

7. Creating Time Series Data

7.1. Create time series of specific frequency from Timestamp array

```
pandas.date_range(start=None, end=None, periods=None, freq=None, tz=None, normalize=False, name=None,
closed=None, **kwargs)
```

start: the start of date range, end: the end of date range, periods: the number of timestamps to be created

For more details on freq, refer to https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases

If we describe only representative examples of freq,

- D: calendar day frequency
- B: business day (Business day frequency)
- W: weekly frequency
- M: month and frequency
- Ms: month start frequency
- Q: quater and frequency

7. Creating Time Series Data

7.1. Create time series of specific frequency from Timestamp array

pandas.date_range(start=None, end=None, periods=None, freq=None, tz=None, normalize=False, name=None, closed=None, **kwargs)

- Since there are 5 periods, it means to create 5 Timestamps.

```
1 import pandas as pd
2 pd.date_range(start='2021-01-01', end=None, periods=5, freq='D', tz=None)
```

```
DatetimeIndex(['2021-01-01', '2021-01-02', '2021-01-03', '2021-01-04',
               '2021-01-05'],
              dtype='datetime64[ns]', freq='D')
```

Line 2

- In case of a native timezone state where the timezone is not set, count the number of data created.

7. Creating Time Series Data

7.1. Create time series of specific frequency from Timestamp array

pandas.date_range(start=None, end=None, periods=None, freq=None, tz=None, normalize=False, name=None, closed=None, **kwargs)

```
1 import pandas as pd
2 pd.date_range(start='2021-01-01', end='2021-12-31', periods=5, freq=None, tz='Asia/Seoul')
```

```
DatetimeIndex(['2021-01-01 00:00:00+09:00', '2021-04-02 00:00:00+09:00',
               '2021-07-02 00:00:00+09:00', '2021-10-01 00:00:00+09:00',
               '2021-12-31 00:00:00+09:00'],
              dtype='datetime64[ns, Asia/Seoul]', freq=None)
```

Line 2

- In case the time zone is set to Seoul

7. Creating Time Series Data

7.1. Create time series of specific frequency from Timestamp array

pandas.date_range(start=None, end=None, periods=None, freq=None, tz=None, normalize=False, name=None, closed=None, **kwargs)

```
1 import pandas as pd
2 pd.date_range(start='2021-01-01', end=None, periods=5, freq='M', tz=None)
```

```
DatetimeIndex(['2021-01-31', '2021-02-28', '2021-03-31', '2021-04-30',
               '2021-05-31'],
              dtype='datetime64[ns]', freq='M')
```

Line 2

- In case of using the parameter as the frequency based on the end of the month

7. Creating Time Series Data

7.1. Create time series of specific frequency from Timestamp array

pandas.date_range(start=None, end=None, periods=None, freq=None, tz=None, normalize=False, name=None, closed=None, **kwargs)

```
1 import pandas as pd
2 pd.date_range(start='2021-01-01', end=None, periods=5, freq='Ms', tz=None)
```

```
DatetimeIndex([       '2021-01-01 00:00:00', '2021-01-01 00:00:00.001000',
               '2021-01-01 00:00:00.002000', '2021-01-01 00:00:00.003000',
               '2021-01-01 00:00:00.004000'],
              dtype='datetime64[ns]', freq='L')
```

Line 2

- In case of using the frequency as a parameter based on the month start date

7. Creating Time Series Data

7.2. Create time series of specific frequency from Period array

You can create time series data containing multiple periods and frequencies with period_range() . It returns PeriodIndex as a result.

```
pandas.period_range(start=None, end=None, periods=None, freq=None, name=None)
```

https://pandas.pydata.org/docs/reference/api/pandas.period_range.html

```
1  import pandas as pd
2
3  p_data= pd.period_range(start='2020-1-1',
4                          end= '2020-12-31',
5                          periods = None,
6                          freq='M')
7
8  p_data
```

```
PeriodIndex(['2020-01', '2020-02', '2020-03', '2020-04', '2020-05', '2020-06',
             '2020-07', '2020-08', '2020-09', '2020-10', '2020-11', '2020-12'],
            dtype='period[M]')
```

Line 3, 4

- 3: Start of date range
- 4: End of date range

7. Creating Time Series Data

7.2. Create time series of specific frequency from Period array

You can create time series data containing multiple periods and frequencies with period_range() . It returns PeriodIndex as a result.

```
pandas.period_range(start=None, end=None, periods=None, freq=None, name=None)
```

https://pandas.pydata.org/docs/reference/api/pandas.period_range.html

```
1  import pandas as pd
2
3  p_data= pd.period_range(start='2020-1-1',
4                          end= '2020-12-31',
5                          periods = None,
6                          freq='M')
7
8  p_data
```

```
PeriodIndex(['2020-01', '2020-02', '2020-03', '2020-04', '2020-05', '2020-06',
             '2020-07', '2020-08', '2020-09', '2020-10', '2020-11', '2020-12'],
            dtype='period[M]')
```

Line 5, 6

- 5: Number of frequencies to generate
- 6: Length of period

7. Creating Time Series Data

7.2. Create time series of specific frequency from Period array

You can create time series data containing multiple periods and frequencies with period_range() . It returns PeriodIndex as a result.

```
pandas.period_range(start=None, end=None, periods=None, freq=None, name=None)
```

https://pandas.pydata.org/docs/reference/api/pandas.period_range.html

```
1  import pandas as pd
2
3  p_data= pd.period_range(start='2020-1-1',
4                          end= '2020-12-31',
5                          periods = None,
6                          freq='M')
7
8  p_data
```

```
PeriodIndex(['2020-01', '2020-02', '2020-03', '2020-04', '2020-05', '2020-06',
             '2020-07', '2020-08', '2020-09', '2020-10', '2020-11', '2020-12'],
            dtype='period[M]')
```

Line 8

- The label of the index is period

7. Creating Time Series Data

7.2. Create time series of specific frequency from Period array

pandas.period_range(start=None, end=None, periods=None, freq=None, name=None)

```
1 for i in p_data:
2     print("{0} {1}".format(i.start_time,i.end_time))
```

```
2020-01-01 00:00:00 2020-01-31 23:59:59.999999999
2020-02-01 00:00:00 2020-02-29 23:59:59.999999999
2020-03-01 00:00:00 2020-03-31 23:59:59.999999999
2020-04-01 00:00:00 2020-04-30 23:59:59.999999999
2020-05-01 00:00:00 2020-05-31 23:59:59.999999999
2020-06-01 00:00:00 2020-06-30 23:59:59.999999999
2020-07-01 00:00:00 2020-07-31 23:59:59.999999999
2020-08-01 00:00:00 2020-08-31 23:59:59.999999999
2020-09-01 00:00:00 2020-09-30 23:59:59.999999999
2020-10-01 00:00:00 2020-10-31 23:59:59.999999999
2020-11-01 00:00:00 2020-11-30 23:59:59.999999999
2020-12-01 00:00:00 2020-12-31 23:59:59.999999999
```

Line 2

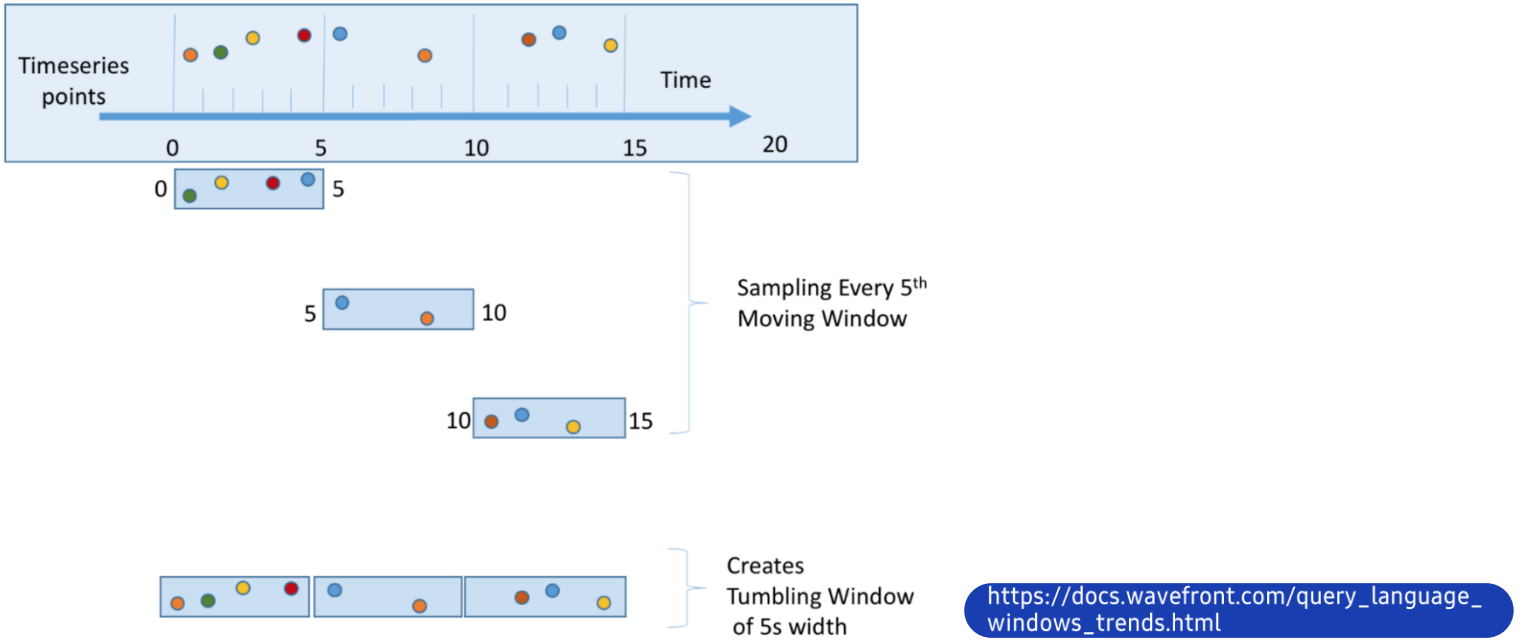
- See the data carefully. It automatically creates when each month actually starts and ends.

8. Moving Statistics Function (Rolling Window Calculations)

- Pandas provides a function that can easily calculate the moving statistics (Rolling window) for DataFrame and Series.
- The expression “window” refers to a period or section in which specific data is expressed.
- It is a format in which the window is automatically moved according to the specified interval, the statistics are calculated accordingly, and the entire time series data is applied.

8. Moving Statistics Function (Rolling Window Calculations)

To explain by referring to the image below, the moving average is the most used method for time series financial data analysis in order to smooth the short-term volatility of the target data and analyze long-term trends.



https://docs.wavefront.com/query_language_windows_trends.html

TIP

- Smoothing means smooth processing by removing small fluctuations or discontinuities that are not good in the data due to noise when sampling from large sample data.

8. Moving Statistics Function (Rolling Window Calculations)

DataFrame rolling function : pandas.DataFrame.rolling( )

- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rolling.html

Series rolling function : pandas.Series.rolling( )

- https://pandas.pydata.org/docs/reference/api/pandas.Series.rolling.html?highlight=series%20rolling#pandas.Series.rolling

Representative method

.rolling().mean()	Average in window
.rolling().std()	Standard deviation in window
.rolling().var()	Dispersion in window
.rolling().sum()	Total in window
.rolling().min()	Minimum value in window
.rolling().max()	Maximum value in window

8. Moving Statistics Function (Rolling Window Calculations)

Let's practice the stock price data using the moving average concept as an example.

```
1  import pandas as pd
2  import datetime
3  from datetime import date, datetime, time
4  import matplotlib.pyplot as plt
5
6  import pandas_datareader.data as web
7  start = datetime(2020,1,1)
8  end = datetime(2020,12,31)
9
10 sec = web.DataReader("005930.KS",
11                      'yahoo',
12                      start,
13                      end)
14 sec.head()
```

Date	High	Low	Open	Close	Volume	Adj Close
2020-01-02	56000.0	55000.0	55500.0	55200.0	12993228.0	52058.132812
2020-01-03	56600.0	54900.0	56000.0	55500.0	15422255.0	52341.058594
2020-01-06	55600.0	54600.0	54900.0	55500.0	10278951.0	52341.058594
2020-01-07	56400.0	55600.0	55700.0	55800.0	10009778.0	52623.980469
2020-01-08	57400.0	55900.0	56200.0	56800.0	23501171.0	53567.062500

8. Moving Statistics Function (Rolling Window Calculations)

Let's practice the stock price data using the moving average concept as an example.

```
1  import pandas as pd
2  import datetime
3  from datetime import date, datetime, time
4  import matplotlib.pyplot as plt
5
6  import pandas_datareader.data as web
7  start = datetime(2020,1,1)
8  end = datetime(2020,12,31)
9
10 sec = web.DataReader("005930.KS",
11                      'yahoo',
12                      start,
13                      end)
14 sec.head()
```

Line 6, 7, 10

- 6: It is a Korean stock data set provided by Yahoo Finance in datareader.
- 7: If you want to practice with a wide window, it is better to specify a longer period.
- 10: Ticker code. Samsung

8. Moving Statistics Function (Rolling Window Calculations)

Let's practice the stock price data using the moving average concept as an example.

```
1  import pandas as pd
2  import datetime
3  from datetime import date, datetime, time
4  import matplotlib.pyplot as plt
5
6  import pandas_datareader.data as web
7  start = datetime(2020,1,1)
8  end = datetime(2020,12,31)
9
10 sec = web.DataReader("005930.KS",
11                      'yahoo',
12                      start,
13                      end)
14 sec.head()
```

Line 11, 12, 13

- 11: Data specified from Yahoo Finance.
- 12: Start date of search, use timestamp stored in start variable
- 13: Search end date, using the timestamp stored in the end variable

8. Moving Statistics Function (Rolling Window Calculations)

Let's practice the stock price data using the moving average concept as an example.

```
1 sec.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 248 entries, 2020-01-02 to 2020-12-30
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   High       248 non-null    float64
 1   Low        248 non-null    float64
 2   Open       248 non-null    float64
 3   Close      248 non-null    float64
 4   Volume     248 non-null    float64
 5   Adj Close  248 non-null    float64
dtypes: float64(6)
memory usage: 13.6 KB
```

Line 1

- You can see that this data set is already in DatetimeIndex.

8. Moving Statistics Function (Rolling Window Calculations)

Let's practice the stock price data using the moving average concept as an example.

```
1 ma5 = sec['Adj Close'].rolling(window=5).mean()
2 ma5.head(10)
```

```
Date
2020-01-02             NaN
2020-01-03             NaN
2020-01-06             NaN
2020-01-07             NaN
2020-01-08    52586.258594
2020-01-09    53227.555469
2020-01-10    53982.021094
2020-01-13    54830.794531
2020-01-14    55622.983594
2020-01-15    56037.939063
Name: Adj Close, dtype: float64
```

Line 1

- It is easy to calculate the moving average data over 5 days.

8. Moving Statistics Function (Rolling Window Calculations)

Let's visualize each moving average data.

```
1  close = sec['Adj Close']
2
3  ma5 = sec['Adj Close'].rolling(window=5).mean()
4  ma10 = sec['Adj Close'].rolling(window=10).mean()
5  ma60 = sec['Adj Close'].rolling(window=60).mean()
6  ma120 = sec['Adj Close'].rolling(window=20).mean()
7
8  plt.figure(figsize = (15,10))
9
10 plt.plot(close, label='close', linewidth=4)
11 plt.plot(ma5, label='5 window')
12 plt.plot(ma10, label='10 window')
13 plt.plot(ma60, label='60 window')
14 plt.plot(ma120, label='120 window')
15
16 plt.legend()
17 plt.title('Rolling window calculations')
18 plt.xlabel('Date')
19 plt.ylabel('close price')
```

```
Text(0, 0.5, 'close price')
```

Line 1, 3

- 1: To compare with the moving average in the graph, only the closing price is stored separately.
- 3: Average over Windows 5 days

8. Moving Statistics Function (Rolling Window Calculations)

Let's visualize each moving average data.

```
close = sec['Adj Close']

ma5 = sec['Adj Close'].rolling(window=5).mean()
ma10 = sec['Adj Close'].rolling(window=10).mean()
ma60 = sec['Adj Close'].rolling(window=60).mean()
ma120 = sec['Adj Close'].rolling(window=20).mean()

plt.figure(figsize = (15,10))

plt.plot(close, label='close', linewidth=4)
plt.plot(ma5, label='5 window')
plt.plot(ma10, label='10 window')
plt.plot(ma60, label='60 window')
plt.plot(ma120, label='120 window')

plt.legend()
plt.title('Rolling window calculations')
plt.xlabel('Date')
plt.ylabel('close price')
```

```
Text(0, 0.5, 'close price')
```

8. Moving Statistics Function (Rolling Window Calculations)

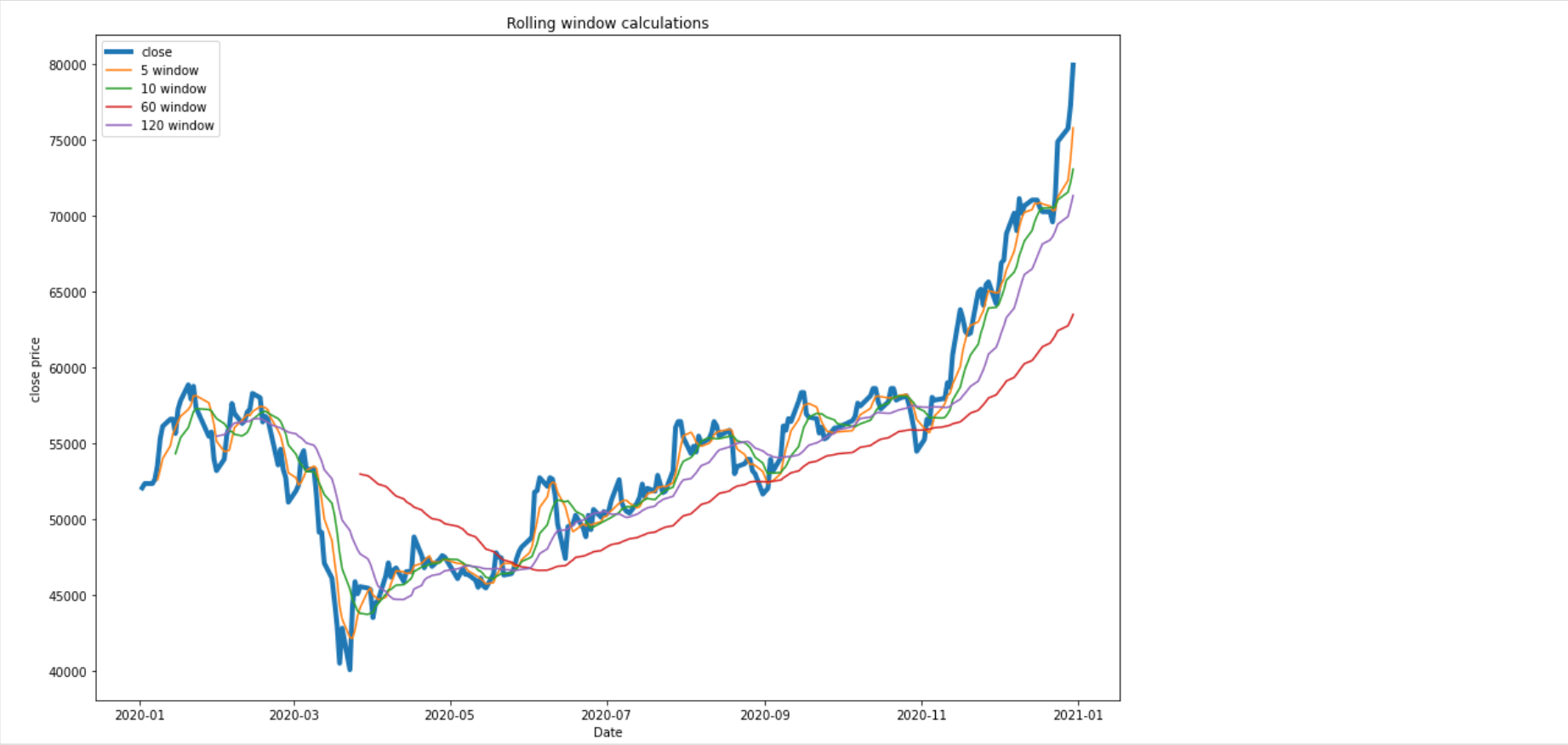
Let's visualize each moving average data.

Line 1, 3, 4, 5, 6, 8, 10, 11, 12, 13, 14

- 1: To compare with the moving average in the graph, only the closing price is stored separately.
- 3: Average over Windows 5 days
- 4: Average over Windows 10 days
- 5: Average over Windows 60 days
- 6: Average over Windows 120 days
- 8: Set the size of the chart
- 10: Express the closing price data as a line graph and name the label close. Since it is the reference data, it is expressed a little thicker.
- 11: Express the daily moving average data as a line graph and name the label 5 window.
- 12: Express the 10-day moving average data as a line graph and name the label 10 window.
- 13: Express the 60-day moving average data as a line graph and name the label 60 window.
- 14: Express the 120-day moving average data as a line graph and name the label 120 window.

8. Moving Statistics Function (Rolling Window Calculations)

The figure below is the result graph.



Let's code

Step 1

Data acquisition and data frame transformation

```
1 import pandas as pd
2 import numpy as np
3 import datetime
4 import matplotlib.pyplot as plt
5
6 from datetime import date, datetime, time, timezone
7
8 df = pd.read_csv("./data//fish/capture-fisheries-vs-aquaculture.csv")
9 df.head()
```

	Entity	Code	Year	Aquaculture production (metric tons)	Capture fisheries production (metric tons)
0	Afghanistan	AFG	1969	60.0	400.0
1	Afghanistan	AFG	1970	60.0	400.0
2	Afghanistan	AFG	1971	60.0	500.0
3	Afghanistan	AFG	1972	60.0	500.0
4	Afghanistan	AFG	1973	60.0	500.0

Line 8

- https://ourworldindata.org/fish-and-overfishing

Step 1

Data acquisition and data frame transformation

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14674 entries, 0 to 14673
Data columns (total 5 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Entity                                  14674 non-null  object
 1   Code                                    11901 non-null  object
 2   Year                                    14674 non-null  int64
 3   Aquaculture production (metric tons)    11657 non-null  float64
 4   Capture fisheries production (metric tons)  14516 non-null  float64
dtypes: float64(2), int64(1), object(2)
memory usage: 573.3+ KB
```

Line 1

- Let's check the data type of each column of the data frame.

Step 2

Preprocessing including data cleaning

```
1
2 df.drop(['Code'], axis=1, inplace=True)
3 df.head()
```

	Entity	Year	Aquaculture production (metric tons)	Capture fisheries production (metric tons)
0	Afghanistan	1969	60.0	400.0
1	Afghanistan	1970	60.0	400.0
2	Afghanistan	1971	60.0	500.0
3	Afghanistan	1972	60.0	500.0
4	Afghanistan	1973	60.0	500.0

Line 1

- Delete unnecessary columns. Let's delete the code column in the statistics we want to do because we don't need it.

Step 2

Preprocessing including data cleaning

```
1  # Search NaN data. You can check the number of missing data for each column.
2  df.isnull().sum()
```

```
Entity                                          0
Year                                            0
Aquaculture production (metric tons)         3017
Capture fisheries production (metric tons)    158
dtype: int64
```

Step 2

Preprocessing including data cleaning

```
1 change_value=0
2 df.fillna(change_value, inplace= True)
3 df.isnull().sum()
```

```
Entity                                    0
Year                                      0
Aquaculture production (metric tons)      0
Capture fisheries production (metric tons)    0
dtype: int64
```

Line 1~3

- 1: Create a variable with the data you want to replace
- 2: Replace NaN with 0 in order not to affect the sum statistic.
- 3: When the substitution result was confirmed, all NaN data were substituted with 0, and there is currently no number of NaNs.

Step 2

Change to time series data type and replace index

```
1 df['new_Year'] = pd.to_datetime(df['Year'].astype(str), format='%Y')
2 df.set_index('new_Year', inplace= True)
3 df.drop(['Year'], axis=1, inplace=True)
4
5 df.head()
```

new_Year	Entity	Aquaculture production (metric tons)	Capture fisheries production (metric tons)
1969-01-01	Afghanistan	60.0	400.0
1970-01-01	Afghanistan	60.0	400.0
1971-01-01	Afghanistan	60.0	500.0
1972-01-01	Afghanistan	60.0	500.0
1973-01-01	Afghanistan	60.0	500.0

Line 1, 2

- 1: Year is an int64 type. An error occurs if you change it to datetime. It is necessary to change the data type.
- 2: Specify the new_Year column changed in the time series format as an index.

Step 2

Change to time series data type and replace index

```
1 df['new_Year'] = pd.to_datetime(df['Year'].astype(str), format='%Y')
2 df.set_index('new_Year', inplace= True)
3 df.drop(['Year'], axis=1, inplace=True)
4
5 df.head()
```

new_Year	Entity	Aquaculture production (metric tons)	Capture fisheries production (metric tons)
1969-01-01	Afghanistan	60.0	400.0
1970-01-01	Afghanistan	60.0	400.0
1971-01-01	Afghanistan	60.0	500.0
1972-01-01	Afghanistan	60.0	500.0
1973-01-01	Afghanistan	60.0	500.0

Line 3, 5

- 3: Delete the existing Year column because it is no longer needed.
- 5: Confirm the above processing result.

Step 2

Change to time series data type and replace index

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 14674 entries, 1969-01-01 to 1969-01-01
Data columns (total 3 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Entity                                  14674 non-null  object
 1   Aquaculture production (metric tons)    14674 non-null  float64
 2   Capture fisheries production (metric tons)  14674 non-null  float64
dtypes: float64(2), object(1)
memory usage: 458.6+ KB
```

Line 1

- See the result, and you can confirm that the datetimeindex has been changed.

Step 2

Change to time series data type and replace index

```
1 new_df=df.sort_index()
2 new_df.head()
```

new_Year	Entity	Aquaculture production (metric tons)	Capture fisheries production (metric tons)
1960-01-01	Central African Republic	0.0	2000.0
1960-01-01	Argentina	0.0	104732.0
1960-01-01	Saint Vincent and the Grenadines	0.0	400.0
1960-01-01	Turks and Caicos Islands	0.0	830.0
1960-01-01	Lebanon	0.0	2100.0

Line 1

- Sorting dataframes based on set index

Step 2

Change to time series data type and replace index

```
1 new_df['Entity']=new_df['Entity'].astype('category')
2 new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 14674 entries, 1960-01-01 to 2018-01-01
Data columns (total 3 columns):
 #   Column                                    Non-Null Count  Dtype
---  ------                                    --------------  -----
 0   Entity                                    14674 non-null  category
 1   Aquaculture production (metric tons)      14674 non-null  float64
 2   Capture fisheries production (metric tons)  14674 non-null  float64
dtypes: category(1), float64(2)
memory usage: 382.7 KB
```

Line 2

- You can see that the data type for the country name has been changed from object to categorical.

Step 3

```
1 new_df['Entity'].value_counts()
```

```
Afghanistan                                59
Middle East & North Africa (IDA & IBRD)    59
Middle income                              59
Morocco                                    59
Mozambique                                 59
                                           ..
Montenegro                                 13
Saint Martin (French part)                 12
Sint Maarten (Dutch part)                   8
Curacao                                     8
South Sudan                                 7
Name: Entity, Length: 264, dtype: int64
```

Line 1

- Let's check how many unique data there are in the column with country name. There are data for a total of 264 countries.

Step 3

- By adding up each country's catch and aquaculture production by year, we can visualize this to see trends in global catch and aquaculture.
- If you see the data, there are separate data for each country and year. This can be solved by calculating the sum of the world (The sum of data from each country.) for each year through the group operation of pandas based on the year and visualizing this.

```
1 g = new_df.groupby(['new_Year'])
```

Line 1

- Groups are grouped based on the new_Year column and stored them in a new data frame called g.

Step 3

```
1 g.head()
```

new_Year	Entity	Aquaculture production (metric tons)	Capture fisheries production (metric tons)
1960-01-01	Central African Republic	0.000000e+00	2.000000e+03
1960-01-01	Argentina	0.000000e+00	1.047320e+05
1960-01-01	Saint Vincent and the Grenadines	0.000000e+00	4.000000e+02
1960-01-01	Turks and Caicos Islands	0.000000e+00	8.300000e+02
1960-01-01	Lebanon	0.000000e+00	2.100000e+03
...	...	...	...
2018-01-01	Middle East & North Africa (excluding high inc...	2.060145e+06	3.023863e+06
2018-01-01	Thailand	8.908640e+05	1.727179e+06
2018-01-01	Nicaragua	2.946840e+04	5.455400e+04
2018-01-01	Kyrgyzstan	2.558700e+03	1.900000e+01
2018-01-01	North America	6.595080e+05	6.252438e+06

Line 1

- If you check the saved result, you can see that the data frame was created based on the year.

Step 3

```
1  #
2  for key,group in g:
3      print('+key:', key)
4      print('+number:', len(group))
5      print(group.head())
6      print('\n')
```

```
+key: 1960-01-01 00:00:00
+number: 232
                                Entity  \
new_Year
1960-01-01         Central African Republic
1960-01-01                        Argentina
1960-01-01  Saint Vincent and the Grenadines
1960-01-01         Turks and Caicos Islands
1960-01-01                          Lebanon

           Aquaculture production (metric tons)  \
new_Year
1960-01-01                                   0.0
1960-01-01                                   0.0
1960-01-01                                   0.0
1960-01-01                                   0.0
1960-01-01                                   0.0

           Capture fisheries production (metric tons)
new_Year
```

Line 1

- Print the contents of the g object using a loop.

Step 4

```
1 world_total = g.sum()
2 world_total.head()
```

new_Year	Aquaculture production (metric tons)	Capture fisheries production (metric tons)
1960-01-01	19150871.0	2.760285e+08
1961-01-01	17848583.0	3.106441e+08
1962-01-01	18297227.0	3.424042e+08
1963-01-01	20460244.0	3.496285e+08
1964-01-01	22491831.0	3.947974e+08

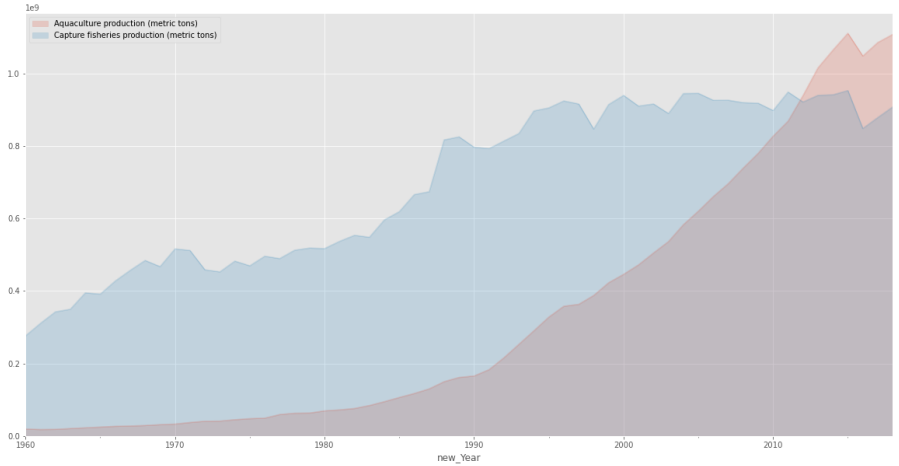
Line 1

- Through group operations for each created group, the sum of each year is obtained and a new data frame is created.

Step 4

Visualize global data

```
plt.style.use('ggplot')
world_total.plot(kind='area',
                 alpha=0.2,
                 stacked=False,
                 figsize = (20,10) )

plt.legend()
plt.show()
```



- Compare the graph we created with the graph we created from Our World data. You can make a result to the level that experts process and visualize.

Step 4

Visualize global data

```
1 plt.style.use('ggplot')
2 world_total.plot(kind='area',
3                  alpha=0.2,
4                  stacked=False,
5                  figsize = (20,10) )
6
7 plt.legend()
8 plt.show()
```

Line 1, 2, 3, 4, 5, 7, 8

- 1: Specify the style of the graph
- 2: Draw an area graph in which the rest of the line graph is colored.
- 3: Adjust the opacity of the color to increase the visibility of overlapping graphs.
- 4: Select the option with False
- 5: Specify the size of the graph
- 7: Show legend
- 8: If you check the results, you can see that the world has been increasing the amount of artificial aquaculture rather than the amount caught little by little starting in 2010.

Step 5

Let's search for a specific country name and visualize it. You can do this easily if you remember the practice of searching for artist names in the DataFrame lecture.

```
1 country = new_df['Entity'].value_counts()
2 print(country)
3 print("Data type =>", type(country))
```

```
Afghanistan                              59
Middle East & North Africa (IDA & IBRD)  59
Middle income                            59
Morocco                                  59
Mozambique                               59
                                         ..
Montenegro                               13
Saint Martin (French part)               12
Sint Maarten (Dutch part)                 8
Curacao                                   8
South Sudan                               7
Name: Entity, Length: 264, dtype: int64
Data type => <class 'pandas.core.series.Series'>
```

Line 1~3

- 1: Create only non-duplicate names as series data in the country column to search for a country.
- 2: Print only non-duplicate country names.
- 3: If you check the processed data type, you can see that it has been successfully converted into a series.

Step 5

```
1 'South Korea'in country
```

```
True
```

Line 1

- If it is True when the country name you want is searched, it means that there is data for the country.

Step 5

Search for 3 or more countries with different cultural and geographic requirements, visualize and get data insights.

```
1 s= new_df.loc[new_df['Entity']=='South Korea']
2 c= new_df.loc[new_df['Entity']=='China']
3 a= new_df.loc[new_df['Entity']=='Afghanistan']
```

Line 1~3

- 1: Create a separate data frame after searching for the country name you want in the series data created earlier.
- 2: Create a separate data frame after searching for the country name you want in the series data created earlier.
- 3: Create a separate data frame after searching for the country name you want in the series data created earlier.

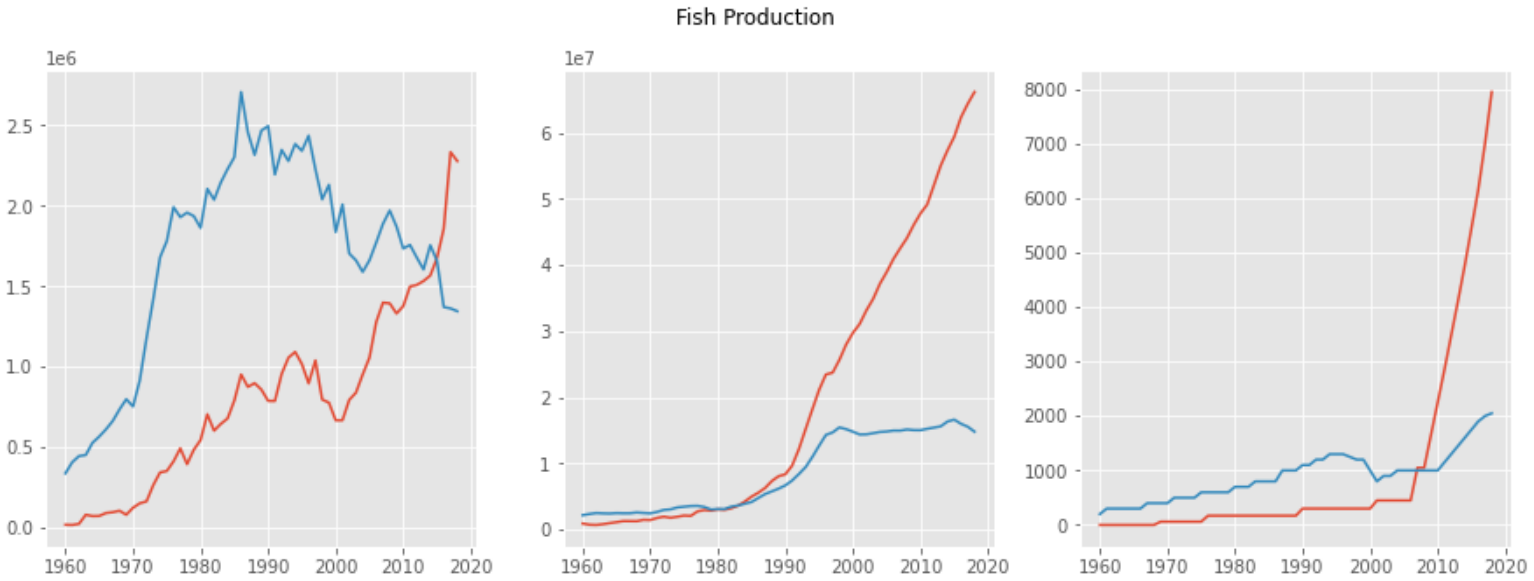
Step 5

```
s_y = s[['Aquaculture production (metric tons)','Capture fisheries production (metric tons)']]
s_x = s.index

c_y = c[['Aquaculture production (metric tons)','Capture fisheries production (metric tons)']]
c_x = c.index

a_y = a[['Aquaculture production (metric tons)','Capture fisheries production (metric tons)']]
a_x = a.index
```

Step 5

```
1 import matplotlib.pyplot as plt
2 fig, axs = plt.subplots(1, 3, figsize=(15, 5))
3 axs[0].plot(s_x, s_y)
4 axs[1].plot(c_x, c_y)
5 axs[2].plot(a_x, a_y)
6
7 fig.suptitle('Fish Production')
```

```
Text(0.5, 0.98, 'Fish Production')
```



Pair programming

Pair Programming Practice

Guideline, mechanisms & contingency plan

Preparing pair programming involves establishing guidelines and mechanisms to help students pair properly and to keep them paired. For example, students should take turns “driving the mouse.” Effective preparation requires contingency plans in case one partner is absent or decides not to participate for one reason or another. In these cases, it is important to make it clear that the active student will not be punished because the pairing did not work well.

Pairing similar, not necessarily equal, abilities as partners

Pair programming can be effective when students of similar, though not necessarily equal, abilities are paired as partners. Pairing mismatched students often can lead to unbalanced participation. Teachers must emphasize that pair programming is not a “divide-and-conquer” strategy, but rather a true collaborative effort in every endeavor for the entire project. Teachers should avoid pairing very weak students with very strong students.

Motivate students by offering extra incentives

Offering extra incentives can help motivate students to pair, especially with advanced students. Some teachers have found it helpful to require students to pair for only one or two assignments.

Pair Programming Practice

Prevent collaboration cheating

The challenge for the teacher is to find ways to assess individual outcomes, while leveraging the benefits of collaboration. How do you know whether a student learned or cheated? Experts recommend revisiting course design and assessment, as well as explicitly and concretely discussing with the students on behaviors that will be interpreted as cheating. Experts encourage teachers to make assignments meaningful to students and to explain the value of what students will learn by completing them.

Collaborative learning environment

A collaborative learning environment occurs anytime an instructor requires students to work together on learning activities. Collaborative learning environments can involve both formal and informal activities and may or may not include direct assessment. For example, pairs of students work on programming assignments; small groups of students discuss possible answers to a professor's question during lecture; and students work together outside of class to learn new concepts. Collaborative learning is distinct from projects where students "divide and conquer." When students divide the work, each is responsible for only part of the problem solving and there are very limited opportunities for working through problems with others. In collaborative environments, students are engaged in intellectual talk with each other.

Q1. Discuss and practice the data you practiced in the key concept section of this lecture with your learning colleagues as shown below.

- Change it to another company's data.
- Try slicing based on the learning date.

Q2. Access the University of California, Irvine, one of the open datasets used in the previous lecture, Data Organization Learning, and explore together with your learning colleagues which data is good data to utilize the advantages of time series data analysis.