# 06_more_pandas

November 16, 2022

## 1 More about pandas

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
```

We've looked at `Series`, `DataFrame`, how to view, select and index data, and how to make plots with `pandas`. Now we will look at time series data, which `pandas` is simply brilliant for working with.

'Time series data' refers to any form of data that is represented or indexed with ordered timestamps. Examples include stock prices, electrical activity in the brain, and temperature throughout the day. As you might have guessed, this involves dates, times, and differences between them.

### 1.1 `Timestamp` and `Timedelta`

In `pandas`, dates and times are represented with `pd.Timestamp`, which is a replacement for Python's native `datetime.datetime` object. The two are interchangeable in many respects, but a key difference is that `pd.Timestamp` uses NumPy's `datetime64` and `timedelta64` data types and incorporates a much wider range of functionality and powerful features for creating and manipulating time series.

Let's create timestamps to mark two memorable points in history: the end of the First and Second World Wars.

[33]:
```
end_of_ww1 = pd.Timestamp('1918-11-11 11:00:00', )  # Armistice Day
end_of_ww2 = pd.Timestamp('1945-05-08 23:01:00')    # VE Day
```

`Timestamp` data have many useful methods. An especially useful method is `.strftime()`, which creates a `str` representation of a timestamp using various formatting codes. Here are some examples.

[34]:
```
print("The First World War ended", end_of_ww1.strftime('%d %B %Y'), "at",␣
 ↪end_of_ww1.strftime('%I:%M %p'))
print("The Second World War ended", end_of_ww2.strftime('%d %B %Y'), "at",␣
 ↪end_of_ww2.strftime('%I:%M %p'))
```

```
The First World War ended 11 November 1918 at 11:00 AM
The Second World War ended 08 May 1945 at 11:01 PM
```

Here, the `%` symbols and subsequent characters tell the `strftime()` method how to represent the timestamp. You may wish to experiment with other formatting codes to format the timestamps in different ways. Copy the above code into the cell below and try to format the dates using abbreviated month names and 24-hour time.

[ ]:
```
```

If you are a history buff, you may already know the day of the week when each of The Great Wars ended. If not, you can find out using the `pd.Timestamp.day_name()` method.

[35]:
```
print('The First World War ended on a', end_of_ww1.day_name())
print('The Second World War ended on a', end_of_ww2.day_name())
```

```
The First World War ended on a Monday
The Second World War ended on a Tuesday
```

If we have two timestamps, we can calculate the difference between them, which is known as a `Timedelta`. Let's do this to find out how much time elapsed between the end of the First and Second World Wars.

[37]:
```
tdelta = end_of_ww2 - end_of_ww1  # Subtract the end of WW1 from the end of WW2
print('Time between the end of the First and Second World Wars:')
print(f'{tdelta.days} days and {tdelta.seconds} seconds')
```

```
Time between the end of the First and Second World Wars:
9675 days and 43260 seconds
```

A `Timedelta` behaves like a number in many respects and can be operated on with most mathematical operators. But there are restrictions. For example, one can divide or multiply a `Timedelta` by any number, but addition and subtraction can only take place with other `Timestamp` or `Timedelta` objects. If you think about it, this makes perfect sense.

[38]:
```
print('Time between end of WW1 and WW2: ', tdelta)
print('Multiplied by 3:                 ', tdelta * 3)
print('Divided by 2:                    ', tdelta / 2)
print('Added to itself:                 ', tdelta + tdelta)
print('Minus twice itself:              ', tdelta - (tdelta * 2))
```

```
Time between end of WW1 and WW2:  9675 days 12:01:00
Multiplied by 3:                  29026 days 12:03:00
Divided by 2:                     4837 days 18:00:30
Added to itself:                  19351 days 00:02:00
Minus twice itself:               -9676 days +11:59:00
```

If you try to perform an operation that is not supported, such as raising a `Timedelta` to some power or multiplying two `Timedelta`'s together, you will get an error.

[43]:
```
tdelta ** 2  # Can not raise a Timedelta to a power...
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/c9/7yddvl1n2ss863cgfngj0wpm0000gp/T/ipykernel_7891/977053077.py in
 ↪<module>
----> 1 tdelta ** 2  # Can not raise a Timedelta to a power…

TypeError: unsupported operand type(s) for ** or pow(): 'Timedelta' and 'int'
```

In case you are wondering, leap years and other time-related idiosyncrasies are dealt with implicitly.

[48]:
```
leap_year = pd.Timestamp('27 Feb 2020')  # 2020 was a leap year
print((leap_year + pd.Timedelta('2D')).strftime('%d %B %Y'))  # Add 2 days
```

```
29 February 2020
```

[49]:
```
not_a_leap_year = pd.Timestamp('27 Feb 2021')  # 2021 wasn't a leap year
print((not_a_leap_year + pd.Timedelta('2D')).strftime('%d %B %Y'))  # Add 2 days
```

```
01 March 2021
```

### 1.2 Working with dates and times

Previously we have encountered functions such as `range()` and `np.linspace()`, which produce sequences of evenly-spaced numeric data. In `pandas`, we can do the same with dates using the `pd.date_range(...)` function. Let's use it to create a timestamp for every day of this year.

[50]:
```
ts = pd.date_range(start='2022-01-01', end='2022-12-31', freq='D',␣
 ↪inclusive='both')
ts
```

[50]:
```
DatetimeIndex(['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-04',
               '2022-01-05', '2022-01-06', '2022-01-07', '2022-01-08',
               '2022-01-09', '2022-01-10',
               …
               '2022-12-22', '2022-12-23', '2022-12-24', '2022-12-25',
               '2022-12-26', '2022-12-27', '2022-12-28', '2022-12-29',
               '2022-12-30', '2022-12-31'],
              dtype='datetime64[ns]', length=365, freq='D')
```

The result is a `DatetimeIndex`, which may be used as the index for a `Series` or `DataFrame`. Let's work some magic on these timestamps to get a list of all future dates for this year in long-format.

[51]:
```
# Get all future dates and format them nicely
ts[ts > pd.Timestamp.now()].strftime('%A %d %B %Y').tolist()
```

[51]:
```
['Thursday 17 November 2022',
 'Friday 18 November 2022',
```

```
 'Saturday 19 November 2022',
 'Sunday 20 November 2022',
 'Monday 21 November 2022',
 'Tuesday 22 November 2022',
 'Wednesday 23 November 2022',
 'Thursday 24 November 2022',
 'Friday 25 November 2022',
 'Saturday 26 November 2022',
 'Sunday 27 November 2022',
 'Monday 28 November 2022',
 'Tuesday 29 November 2022',
 'Wednesday 30 November 2022',
 'Thursday 01 December 2022',
 'Friday 02 December 2022',
 'Saturday 03 December 2022',
 'Sunday 04 December 2022',
 'Monday 05 December 2022',
 'Tuesday 06 December 2022',
 'Wednesday 07 December 2022',
 'Thursday 08 December 2022',
 'Friday 09 December 2022',
 'Saturday 10 December 2022',
 'Sunday 11 December 2022',
 'Monday 12 December 2022',
 'Tuesday 13 December 2022',
 'Wednesday 14 December 2022',
 'Thursday 15 December 2022',
 'Friday 16 December 2022',
 'Saturday 17 December 2022',
 'Sunday 18 December 2022',
 'Monday 19 December 2022',
 'Tuesday 20 December 2022',
 'Wednesday 21 December 2022',
 'Thursday 22 December 2022',
 'Friday 23 December 2022',
 'Saturday 24 December 2022',
 'Sunday 25 December 2022',
 'Monday 26 December 2022',
 'Tuesday 27 December 2022',
 'Wednesday 28 December 2022',
 'Thursday 29 December 2022',
 'Friday 30 December 2022',
 'Saturday 31 December 2022']
```

As you can see, a `DatetimeIndex` is a powerful and flexible tool for representing dates and times. When it really starts to shine is when it is used as the index for time series data, so let's invent some.

Imagine there is a shop that opened at the turn of the millennium which sells apples, bananas and cakes. At the time of opening, these items were sold at the respective prices of £0.45, £0.28, and £0.62 per item, but there's been some inflation over time. Each day, a random(ish) number of each item is sold. Cakes are most popular, then bananas, then apples.

[53]:
```
# Initial cost in year 2000
cost_of_apple = 0.45
cost_of_banana = 0.28
cost_of_cake = 0.62

# Create a range of dates from 2000 to 2022 with 1-day frequency.
# This means we will have a row for every day.
dates = pd.date_range('2000', '2022', freq='1D')
number_of_days = len(dates)

# Create the DataFrame
df = pd.DataFrame(
    {
        'date': dates,
        'apples_sold': np.random.randint(0, 200, number_of_days),
        'bananas_sold': np.random.randint(0, 350, number_of_days),
        'cakes_sold': np.random.randint(0, 700, number_of_days),
        'cost_of_apple' : cost_of_apple,
        'cost_of_banana': cost_of_banana,
        'cost_of_cake': cost_of_cake
    },
)

# Adjust costs for inflation
cost_cols = ['cost_of_apple', 'cost_of_banana', 'cost_of_cake']
inflation = np.exp(np.linspace(0, 1, number_of_days))
df[cost_cols] = df[cost_cols].mul(inflation, axis=0)

# New columns for profit
df['apples_profit'] = df['apples_sold'] * df['cost_of_apple']
df['bananas_profit'] = df['bananas_sold'] * df['cost_of_banana']
df['cakes_profit'] = df['cakes_sold'] * df['cost_of_cake']
df['total_profit'] = df.apples_profit + df.bananas_profit + df.cakes_profit
df = df.round(2)
df
```

[53]:

| | date | apples_sold | bananas_sold | cakes_sold | cost_of_apple | \ |
|---|---|---|---|---|---|---|
| 0 | 2000-01-01 | 175 | 174 | 534 | 0.45 | |
| 1 | 2000-01-02 | 61 | 316 | 394 | 0.45 | |
| 2 | 2000-01-03 | 164 | 320 | 487 | 0.45 | |
| 3 | 2000-01-04 | 55 | 291 | 143 | 0.45 | |
| 4 | 2000-01-05 | 8 | 261 | 630 | 0.45 | |

```
…          …           …            …          …            …
8032 2021-12-28          153          199         467          1.22
8033 2021-12-29          167           67         234          1.22
8034 2021-12-30           85          146         480          1.22
8035 2021-12-31            8           93         338          1.22
8036 2022-01-01           53          103          27          1.22

      cost_of_banana  cost_of_cake  apples_profit  bananas_profit  \
0               0.28          0.62          78.75           48.72
1               0.28          0.62          27.45           88.49
2               0.28          0.62          73.82           89.62
3               0.28          0.62          24.76           81.51
4               0.28          0.62           3.60           73.12
…               …             …             …              …
8032            0.76          1.68         187.06          151.39
8033            0.76          1.68         204.20           50.98
8034            0.76          1.68         103.95          111.10
8035            0.76          1.69           9.78           70.78
8036            0.76          1.69          64.83           78.40

      cakes_profit  total_profit
0           331.08        458.55
1           244.31        360.25
2           302.02        465.46
3            88.69        194.96
4           390.79        467.51
…            …            …
8032        786.66       1125.11
8033        394.22        649.40
8034        808.76       1023.80
8035        569.57        650.13
8036         45.50        188.73

[8037 rows x 11 columns]
```

Now, if we look at the data types of the `DataFrame`, we can see that the `date` column has type `datetime64[ns]`.

[54]:
```
df.dtypes
```

[54]:
```
date              datetime64[ns]
apples_sold                int64
bananas_sold               int64
cakes_sold                 int64
cost_of_apple            float64
cost_of_banana           float64
cost_of_cake             float64
```

```
apples_profit            float64
bananas_profit           float64
cakes_profit             float64
total_profit             float64
dtype: object
```

When working with the world population and titanic datasets, we used the `str` accessor method to interface with additional string methods (e.g., `df['name'].str.contains('Smith')`). For datetime-like data, there is a `dt` accessor method, which opens the door to datetime functionality. Below, we use the `dt` accessor method to create a separate column containing the day of the week, the year, and the fiscal year.

[55]:
```
df['Day'] = df['date'].dt.day_name()  # Get day of week
df['Year'] = df['date'].dt.year  # Get current year
df['Fiscal Year'] = df['date'].dt.to_period('Q-APR').dt.qyear  # Get fiscal year
df.tail()
```

[55]:

| | date | apples_sold | bananas_sold | cakes_sold | cost_of_apple | \ |
|---|---|---|---|---|---|---|
| 8032 | 2021-12-28 | 153 | 199 | 467 | 1.22 | |
| 8033 | 2021-12-29 | 167 | 67 | 234 | 1.22 | |
| 8034 | 2021-12-30 | 85 | 146 | 480 | 1.22 | |
| 8035 | 2021-12-31 | 8 | 93 | 338 | 1.22 | |
| 8036 | 2022-01-01 | 53 | 103 | 27 | 1.22 | |

| | cost_of_banana | cost_of_cake | apples_profit | bananas_profit | \ |
|---|---|---|---|---|---|
| 8032 | 0.76 | 1.68 | 187.06 | 151.39 | |
| 8033 | 0.76 | 1.68 | 204.20 | 50.98 | |
| 8034 | 0.76 | 1.68 | 103.95 | 111.10 | |
| 8035 | 0.76 | 1.69 | 9.78 | 70.78 | |
| 8036 | 0.76 | 1.69 | 64.83 | 78.40 | |

| | cakes_profit | total_profit | Day | Year | Fiscal Year |
|---|---|---|---|---|---|
| 8032 | 786.66 | 1125.11 | Tuesday | 2021 | 2022 |
| 8033 | 394.22 | 649.40 | Wednesday | 2021 | 2022 |
| 8034 | 808.76 | 1023.80 | Thursday | 2021 | 2022 |
| 8035 | 569.57 | 650.13 | Friday | 2021 | 2022 |
| 8036 | 45.50 | 188.73 | Saturday | 2022 | 2022 |

We now have three new and useful columns in our `DataFrame`. At the moment, the original `date` column is just a column of data, but what we really need is for it to be our index.

[59]:
```
# Set the date column as the index
df.index = df.date
```

Now we can index the `DataFrame` using any valid date/time string, of which there are many. For example, if we wanted to look at the information for a particular day, say, 23 April 2012, we could use any of the following:

- `'2012-04-23'`

- 'April 23, 2012'
- '23 Apr 2012'
- '23 April 2012'
- '2012/04/23'

[75]:
```
df.loc['2012/04/23']  # St Georges day, 2012
```

[75]:
```
date              2012-04-23 00:00:00
apples_sold                        19
bananas_sold                      322
cakes_sold                        123
cost_of_apple                    0.79
cost_of_banana                   0.49
cost_of_cake                     1.08
apples_profit                   14.96
bananas_profit                 157.76
cakes_profit                   133.44
total_profit                   306.16
Day                            Monday
Year                             2012
Fiscal Year                      2012
Name: 2012-04-23 00:00:00, dtype: object
```

Try experimenting with different indexing strings!

[ ]:
```

```

If you want the data for an entire year, just pass in the year.

[80]:
```
df.loc['2012']  # Get all data from 2012
```

[80]:
```
                date  apples_sold  bananas_sold  cakes_sold  cost_of_apple  \
date
2012-01-01 2012-01-01          197            35          31           0.78
2012-01-02 2012-01-02            4           338         212           0.78
2012-01-03 2012-01-03          116           299         143           0.78
2012-01-04 2012-01-04          155           309         566           0.78
2012-01-05 2012-01-05           49            89         662           0.78
…                 …            …             …           …              …
2012-12-27 2012-12-27          175           268         538           0.81
2012-12-28 2012-12-28          150           101         349           0.81
2012-12-29 2012-12-29          174            88         347           0.81
2012-12-30 2012-12-30          133           206         599           0.81
2012-12-31 2012-12-31           17           164         200           0.81

            cost_of_banana  cost_of_cake  apples_profit  bananas_profit  \
date
2012-01-01            0.48          1.07         152.95           16.91
```

```
2012-01-02            0.48          1.07           3.11          163.31
2012-01-03            0.48          1.07          90.08          144.48
2012-01-04            0.48          1.07         120.39          149.33
2012-01-05            0.48          1.07          38.06           43.02
…                      …             …             …              …
2012-12-27            0.51          1.12         142.11          135.42
2012-12-28            0.51          1.12         121.83           51.04
2012-12-29            0.51          1.12         141.34           44.48
2012-12-30            0.51          1.12         108.05          104.13
2012-12-31            0.51          1.12          13.81           82.91

            cakes_profit  total_profit        Day  Year  Fiscal Year
date
2012-01-01         33.16        203.02     Sunday  2012         2012
2012-01-02        226.81        393.22     Monday  2012         2012
2012-01-03        153.01        387.57    Tuesday  2012         2012
2012-01-04        605.68        875.40  Wednesday  2012         2012
2012-01-05        708.50        789.58   Thursday  2012         2012
…                  …             …        …    …           …
2012-12-27        601.95        879.48   Thursday  2012         2013
2012-12-28        390.53        563.40     Friday  2012         2013
2012-12-29        388.34        574.15   Saturday  2012         2013
2012-12-30        670.45        882.62     Sunday  2012         2013
2012-12-31        223.88        320.60     Monday  2012         2013

[366 rows x 14 columns]
```

Looks like 2012 was a leap year, because there are 366 days. Let's check this by getting the data for February of that year. To do this, we just need to add the month.

[88]:
```
df.loc['2012-02']  # February 2012
```

[88]:
```
                date  apples_sold  bananas_sold  cakes_sold  cost_of_apple  \
date
2012-02-01 2012-02-01           85            41         271           0.78
2012-02-02 2012-02-02          178           281         337           0.78
2012-02-03 2012-02-03           36           134          47           0.78
2012-02-04 2012-02-04          191            81         223           0.78
2012-02-05 2012-02-05           51           176         314           0.78
2012-02-06 2012-02-06           69           265         190           0.78
2012-02-07 2012-02-07          129           154          20           0.78
2012-02-08 2012-02-08           98            69         301           0.78
2012-02-09 2012-02-09           19           337         351           0.78
2012-02-10 2012-02-10           52           193         652           0.78
2012-02-11 2012-02-11          133            63         199           0.78
2012-02-12 2012-02-12           87           216         310           0.78
2012-02-13 2012-02-13            4            81         199           0.78
```

```
2012-02-14 2012-02-14           0          115         469          0.78
2012-02-15 2012-02-15         112          348         278          0.78
2012-02-16 2012-02-16         155          220          69          0.78
2012-02-17 2012-02-17         176           20         560          0.78
2012-02-18 2012-02-18          50           24         314          0.78
2012-02-19 2012-02-19         122           73         369          0.78
2012-02-20 2012-02-20         111            8         595          0.78
2012-02-21 2012-02-21          24           68         259          0.78
2012-02-22 2012-02-22          46          195         398          0.78
2012-02-23 2012-02-23          64          180          19          0.78
2012-02-24 2012-02-24         115          334         343          0.78
2012-02-25 2012-02-25           4          282         337          0.78
2012-02-26 2012-02-26         115           64         396          0.78
2012-02-27 2012-02-27          48          264         347          0.78
2012-02-28 2012-02-28          58            5         670          0.78
2012-02-29 2012-02-29          38          176         445          0.78

            cost_of_banana  cost_of_cake  apples_profit  bananas_profit  \
date
2012-02-01            0.48          1.07          66.25           19.88
2012-02-02            0.49          1.07         138.75          136.29
2012-02-03            0.49          1.07          28.07           65.00
2012-02-04            0.49          1.07         148.92           39.30
2012-02-05            0.49          1.07          39.77           85.40
2012-02-06            0.49          1.07          53.81          128.59
2012-02-07            0.49          1.07         100.62           74.74
2012-02-08            0.49          1.07          76.45           33.49
2012-02-09            0.49          1.07          14.82          163.59
2012-02-10            0.49          1.08          40.57           93.70
2012-02-11            0.49          1.08         103.79           30.59
2012-02-12            0.49          1.08          67.90          104.89
2012-02-13            0.49          1.08           3.12           39.34
2012-02-14            0.49          1.08           0.00           55.86
2012-02-15            0.49          1.08          87.45          169.06
2012-02-16            0.49          1.08         121.03          106.89
2012-02-17            0.49          1.08         137.45            9.72
2012-02-18            0.49          1.08          39.05           11.66
2012-02-19            0.49          1.08          95.30           35.48
2012-02-20            0.49          1.08          86.72            3.89
2012-02-21            0.49          1.08          18.75           33.06
2012-02-22            0.49          1.08          35.95           94.81
2012-02-23            0.49          1.08          50.02           87.53
2012-02-24            0.49          1.08          89.89          162.44
2012-02-25            0.49          1.08           3.13          137.17
2012-02-26            0.49          1.08          89.91           31.13
2012-02-27            0.49          1.08          37.53          128.44
2012-02-28            0.49          1.08          45.36            2.43
```

```
2012-02-29            0.49          1.08          29.72           85.65

            cakes_profit  total_profit        Day  Year  Fiscal Year
date
2012-02-01        291.01        377.14  Wednesday  2012         2012
2012-02-02        361.93        636.97   Thursday  2012         2012
2012-02-03         50.48        143.55     Friday  2012         2012
2012-02-04        239.56        427.77   Saturday  2012         2012
2012-02-05        337.35        462.52     Sunday  2012         2012
2012-02-06        204.16        386.56     Monday  2012         2012
2012-02-07         21.49        196.85    Tuesday  2012         2012
2012-02-08        323.51        433.45  Wednesday  2012         2012
2012-02-09        377.29        555.71   Thursday  2012         2012
2012-02-10        700.93        835.21     Friday  2012         2012
2012-02-11        213.96        348.34   Saturday  2012         2012
2012-02-12        333.35        506.14     Sunday  2012         2012
2012-02-13        214.01        256.48     Monday  2012         2012
2012-02-14        504.45        560.31    Tuesday  2012         2012
2012-02-15        299.05        555.55  Wednesday  2012         2012
2012-02-16         74.23        302.16   Thursday  2012         2012
2012-02-17        602.55        749.72     Friday  2012         2012
2012-02-18        337.90        388.62   Saturday  2012         2012
2012-02-19        397.14        527.92     Sunday  2012         2012
2012-02-20        640.45        731.06     Monday  2012         2012
2012-02-21        278.82        330.63    Tuesday  2012         2012
2012-02-22        428.51        559.27  Wednesday  2012         2012
2012-02-23         20.46        158.01   Thursday  2012         2012
2012-02-24        369.38        621.71     Friday  2012         2012
2012-02-25        362.97        503.26   Saturday  2012         2012
2012-02-26        426.57        547.61     Sunday  2012         2012
2012-02-27        373.83        539.81     Monday  2012         2012
2012-02-28        721.90        769.69    Tuesday  2012         2012
2012-02-29        479.53        594.90  Wednesday  2012         2012
```

We can also 'slice' a `DataFrame` with a `DatetimeIndex` using date strings. Let's grab the data for the last four days of August 2014.

[85]:
```
df.loc['2014-08-28':'2014-08-31']  # Last 4 days of August 2014
```

[85]:
```
                date  apples_sold  bananas_sold  cakes_sold  cost_of_apple  \
date
2014-08-28 2014-08-28           76           186         569           0.88
2014-08-29 2014-08-29           71            46         662           0.88
2014-08-30 2014-08-30           67           286         487           0.88
2014-08-31 2014-08-31          123           317         163           0.88

            cost_of_banana  cost_of_cake  apples_profit  bananas_profit  \
```

```
date
2014-08-28            0.55          1.21          66.58          101.38
2014-08-29            0.55          1.21          62.20           25.08
2014-08-30            0.55          1.21          58.71          155.93
2014-08-31            0.55          1.21         107.79          172.85

            cakes_profit  total_profit       Day  Year  Fiscal Year
date
2014-08-28        686.75        854.71  Thursday  2014         2015
2014-08-29        799.10        886.38    Friday  2014         2015
2014-08-30        587.93        802.56  Saturday  2014         2015
2014-08-31        196.81        477.45    Sunday  2014         2015
```

To round things off, let's group the data by quarters of the fiscal year (which starts in April) and plot mean profit over time.

[111]:
```
# A new column for quarters of the fiscal year
df['Fiscal Year'] = df['date'].dt.to_period('Q-APR')
df.head()
```

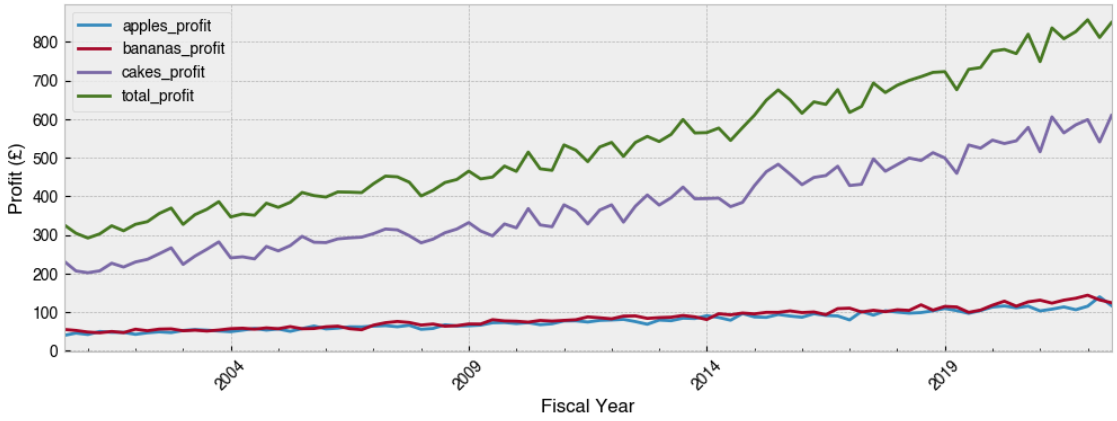
[111]:
```
                date  apples_sold  bananas_sold  cakes_sold  cost_of_apple  \
date
2000-01-01 2000-01-01          175           174         534           0.45
2000-01-02 2000-01-02           61           316         394           0.45
2000-01-03 2000-01-03          164           320         487           0.45
2000-01-04 2000-01-04           55           291         143           0.45
2000-01-05 2000-01-05            8           261         630           0.45

            cost_of_banana  cost_of_cake  apples_profit  bananas_profit  \
date
2000-01-01            0.28          0.62          78.75           48.72
2000-01-02            0.28          0.62          27.45           88.49
2000-01-03            0.28          0.62          73.82           89.62
2000-01-04            0.28          0.62          24.76           81.51
2000-01-05            0.28          0.62           3.60           73.12

            cakes_profit  total_profit        Day  Year Fiscal Year
date
2000-01-01        331.08        458.55   Saturday  2000      2000Q3
2000-01-02        244.31        360.25     Sunday  2000      2000Q3
2000-01-03        302.02        465.46     Monday  2000      2000Q3
2000-01-04         88.69        194.96    Tuesday  2000      2000Q3
2000-01-05        390.79        467.51  Wednesday  2000      2000Q3
```

[112]:
```
# Get the profit columns
profit_cols = df.columns[df.columns.str.endswith('profit')]
(
```

```
    df.groupby('Fiscal Year')[profit_cols]
    .mean()
    .plot(ylabel='Profit (£)', figsize=(12, 4), rot=45)
);
```




If you can master tricks like this in `pandas`, the world of time series data is at your fingertips.

Follow this link to learn more about Time Series / Date functionality in `pandas`