COMP41680

Time Series Analysis

Derek Greene

UCD School of Computer Science

Spring 2023

Static v Dynamic Data

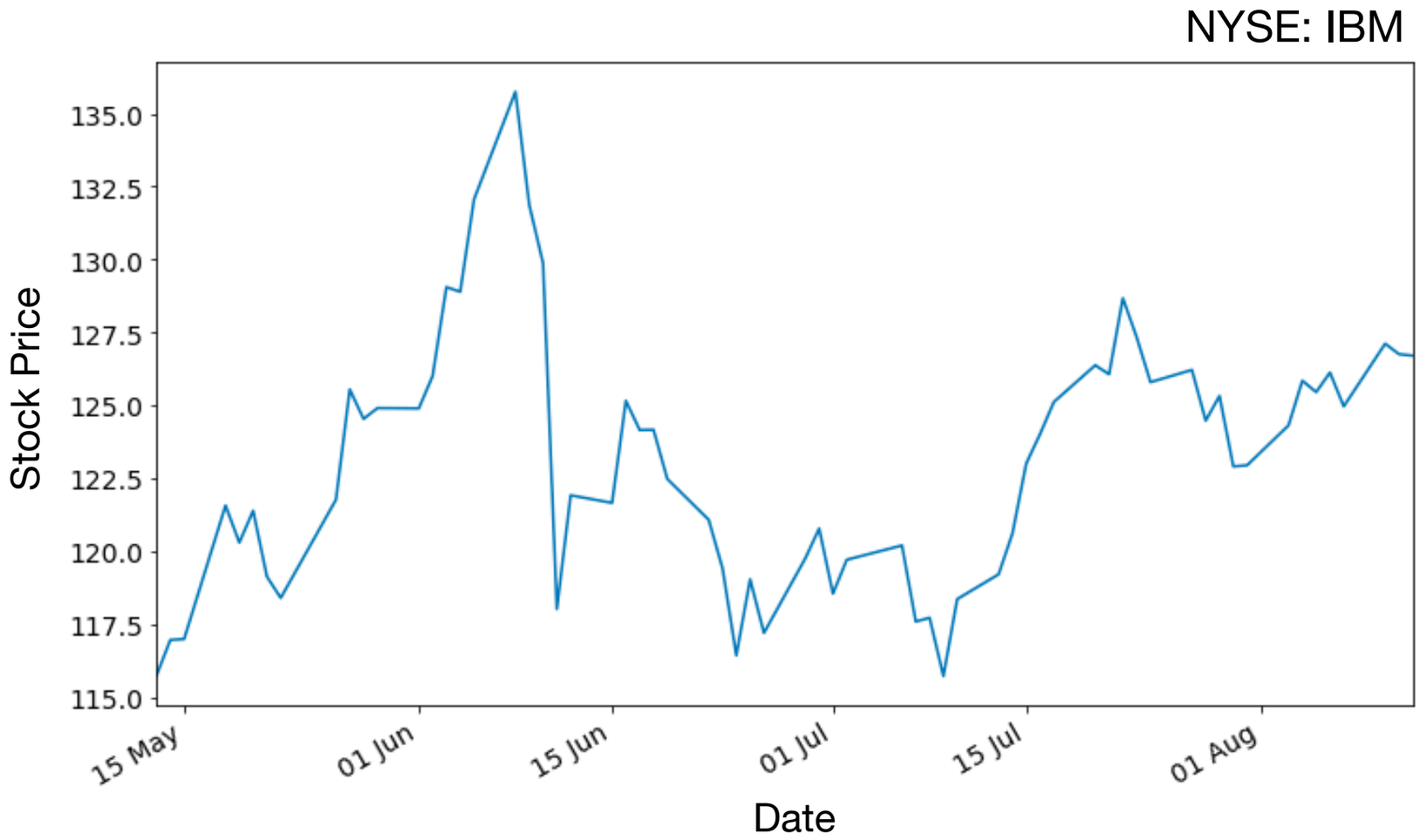
- We have focused on static datasets, which are collected at a fixed point in time and then no longer updated.
- In many situations we do not have a single static dataset. The data is dynamic, arriving periodically or in a continuous stream.
- In some dynamic scenarios, we will need to apply analysis methods that can be efficiently updated to include new data.
- Sometimes we will only want to take into account recent history, rather than all older historical data which may longer be relevant.

- **Example**: "Want to develop a retail sales model that remains accurate as the business gets larger… However, the underlying source is changing; the business is growing, and so your guess of the sales given the previous few days of sales is probably going to be different from last year. So last year's data, when the business was small, is really not relevant to this year, when the business is large." - http://blog.bigml.com

Time Series Data

- Time series: A dataset consisting of the values of a function sampled across different points in time. Most commonly, a time series is a sequence taken at equally spaced points in time.
- Time series data arise in many applications, from financial trading to natural occurring phenomena.

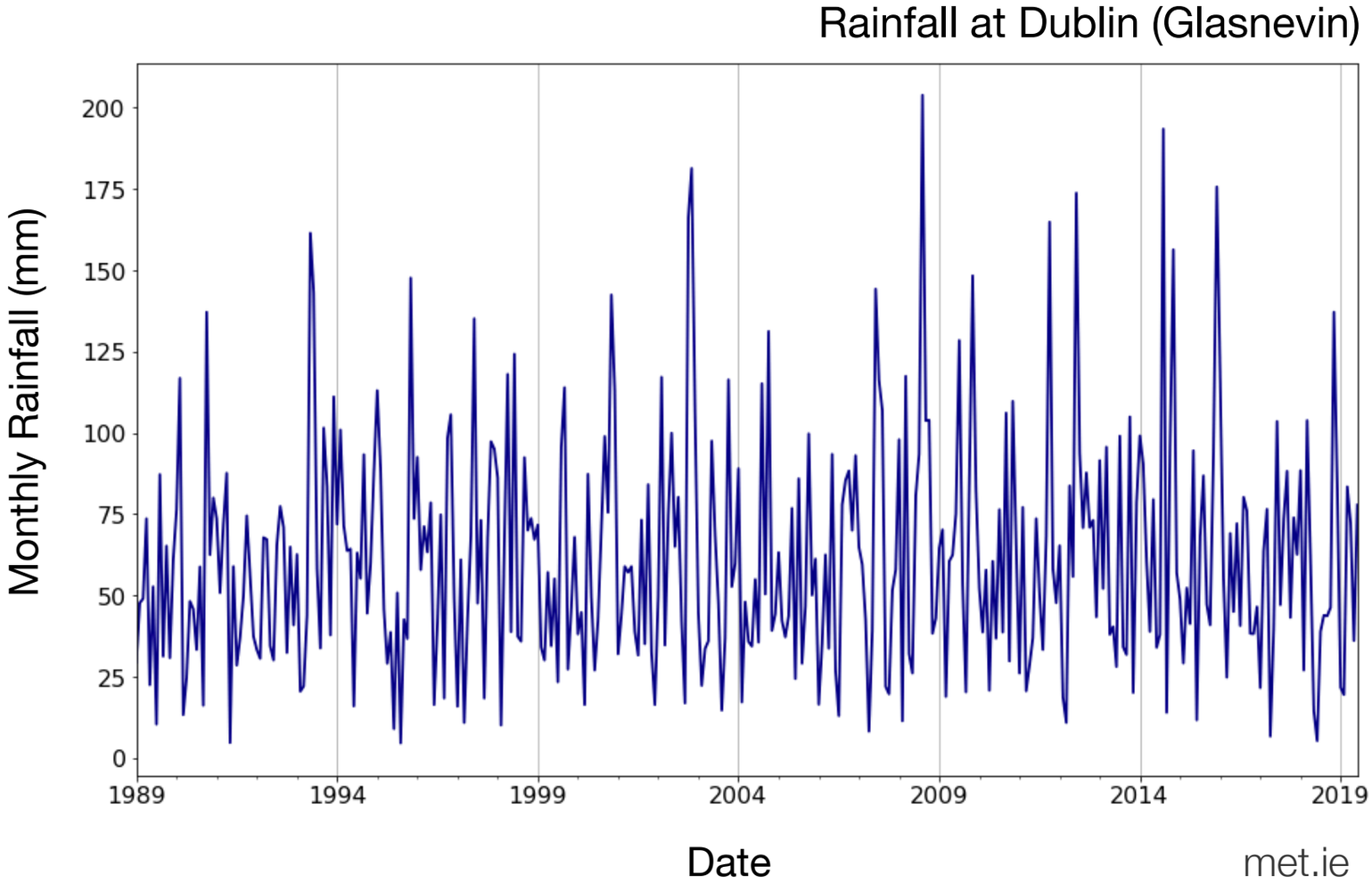
Date	Stock Price
2020-05-13	115.73
2020-05-14	116.95
2020-05-15	116.98
2020-05-18	121.56
2020-05-19	120.29
2020-05-20	121.38
2020-05-21	119.12
2020-05-22	118.39
2020-05-26	121.76
...	...



Time Series Data

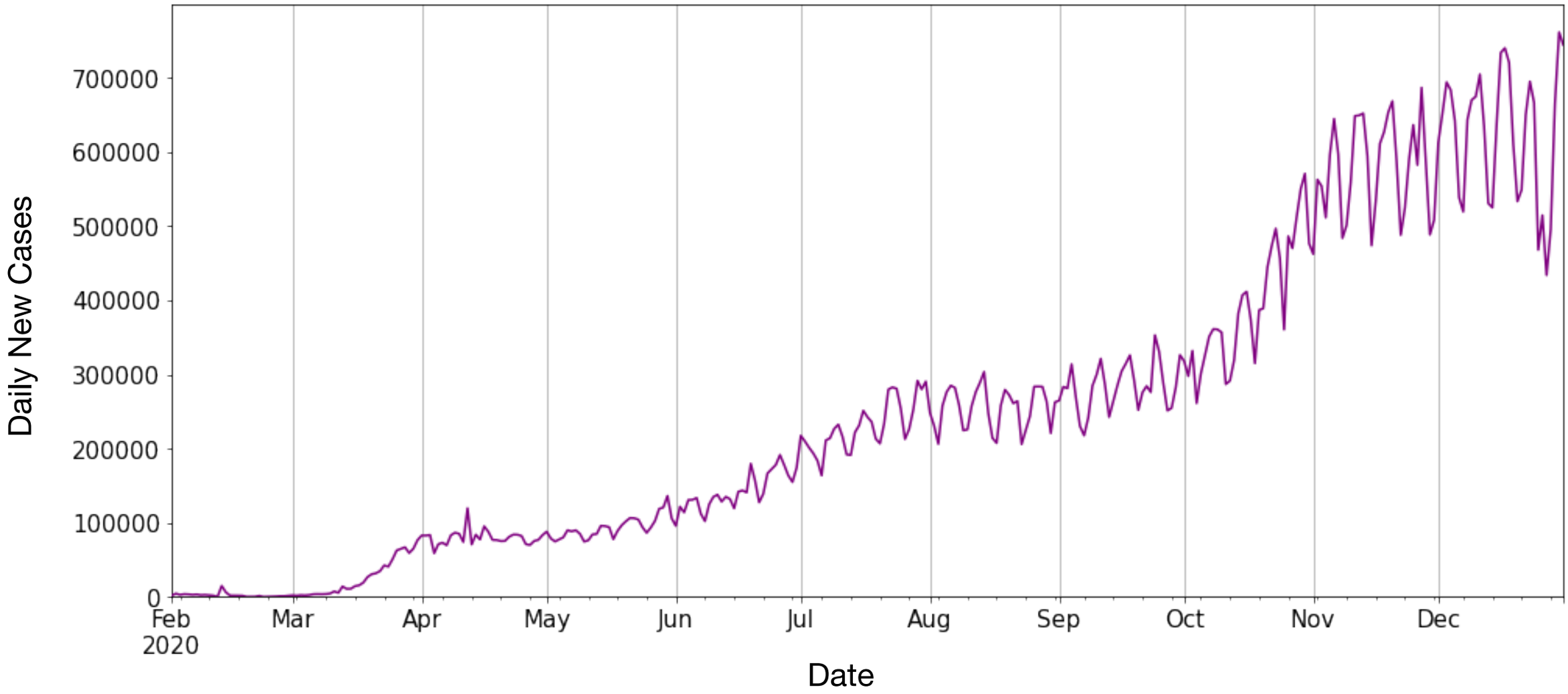
- Time series: A dataset consisting of the values of a function sampled across different points in time. Most commonly, a time series is a sequence taken at equally spaced points in time.
- Time series data arise in many applications, from financial trading to natural occurring phenomena.

Date	Rainfall
1989-01	29.50
1989-02	47.70
1989-03	49.00
1989-04	73.60
1989-05	22.50
1989-06	52.70
1989-07	10.40
1989-08	87.20
1989-09	31.30
...	...



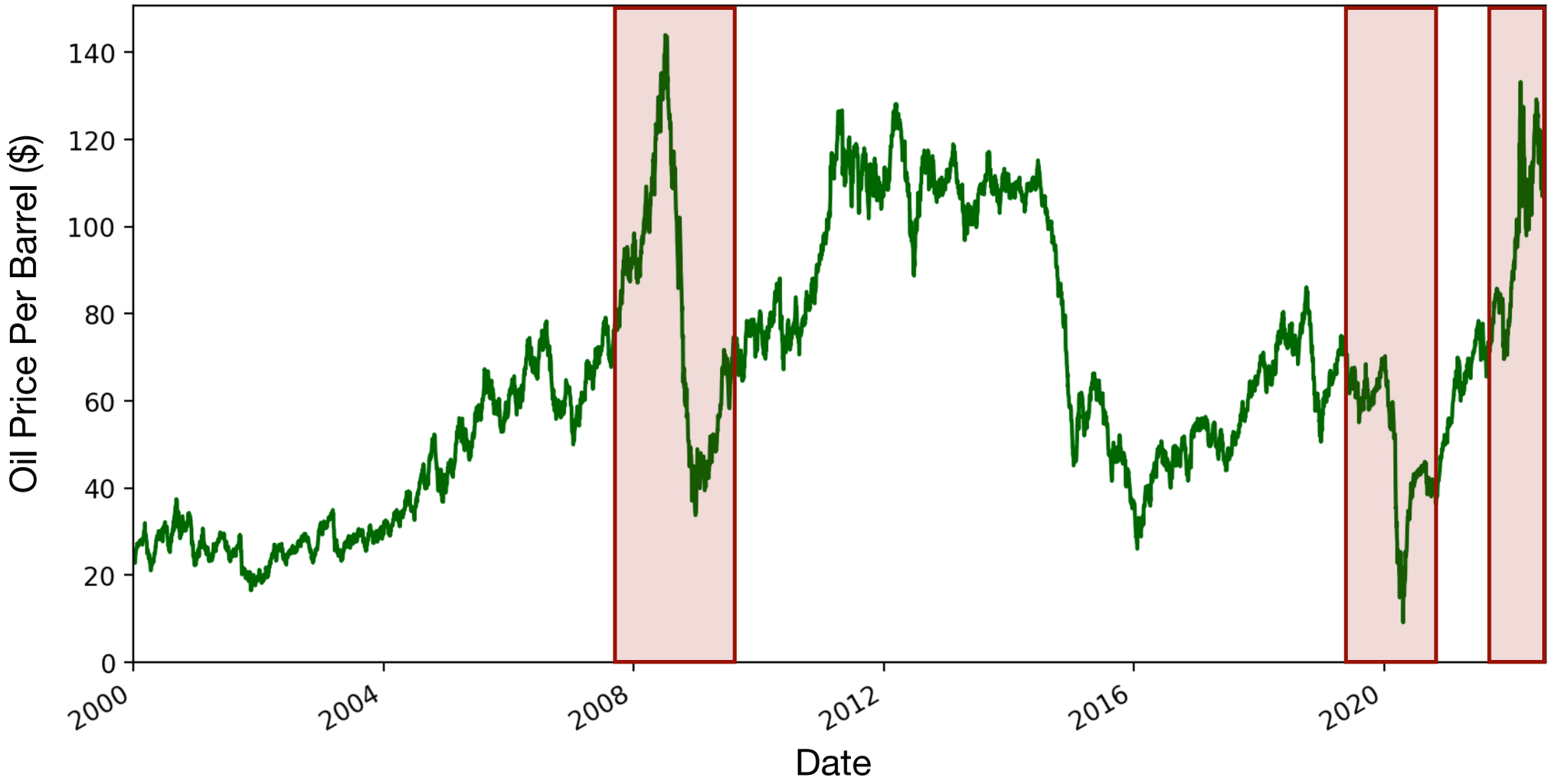
Time Series Data

- We will often plot a time series, and then look for trends, periodic behaviour, seasonal variations, step changes, outliers.
- **Example:** Daily number of COVID-19 cases per day in 2020, represented as a time series, indicating rate of increase in cases.



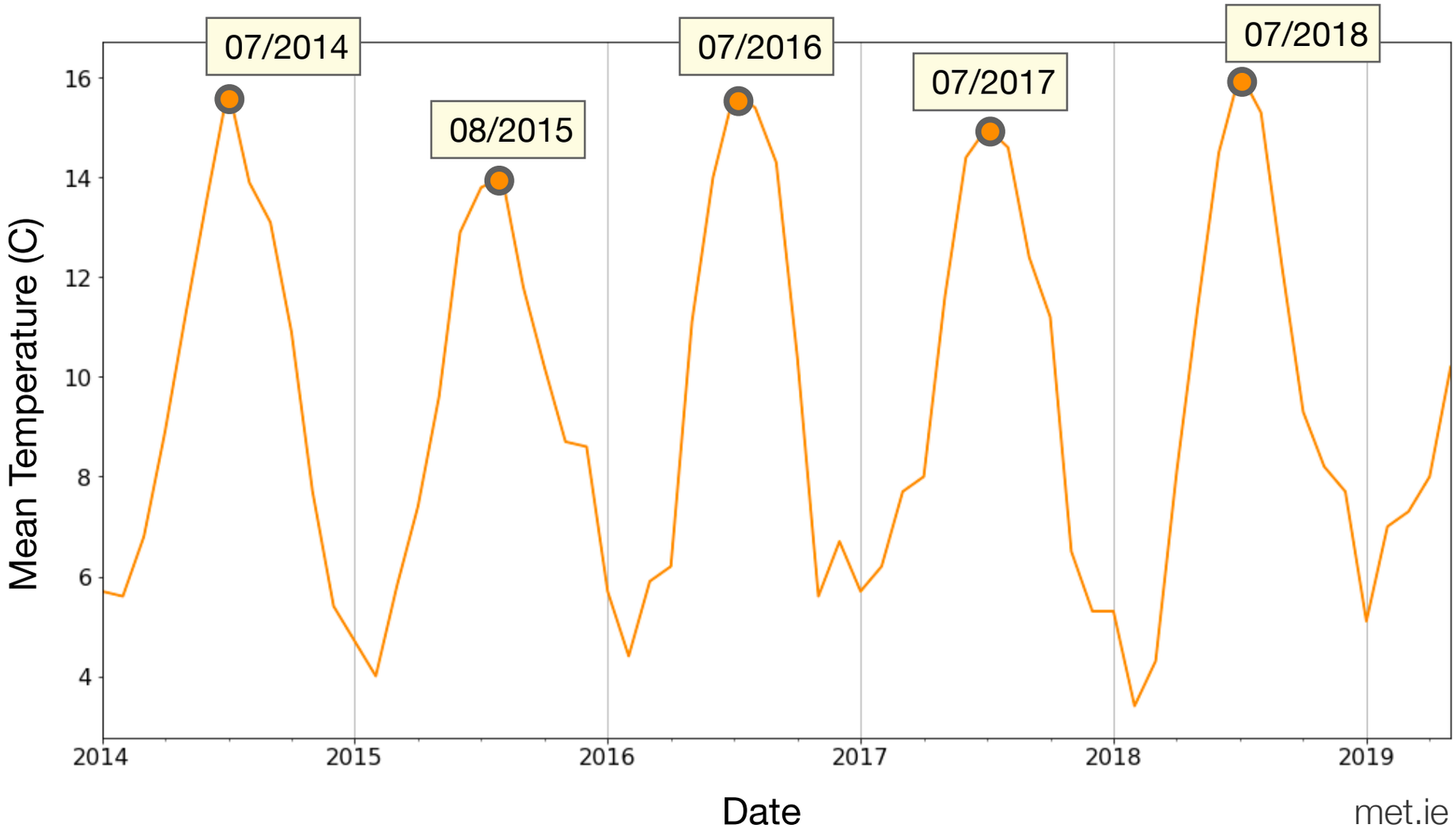
Time Series Data

- We will often plot a time series, and then look for trends, periodic behaviour, seasonal variations, step changes, outliers.
- **Example:** Time series of monthly Brent crude oil prices per barrel since 2000, with major change points highlighted.



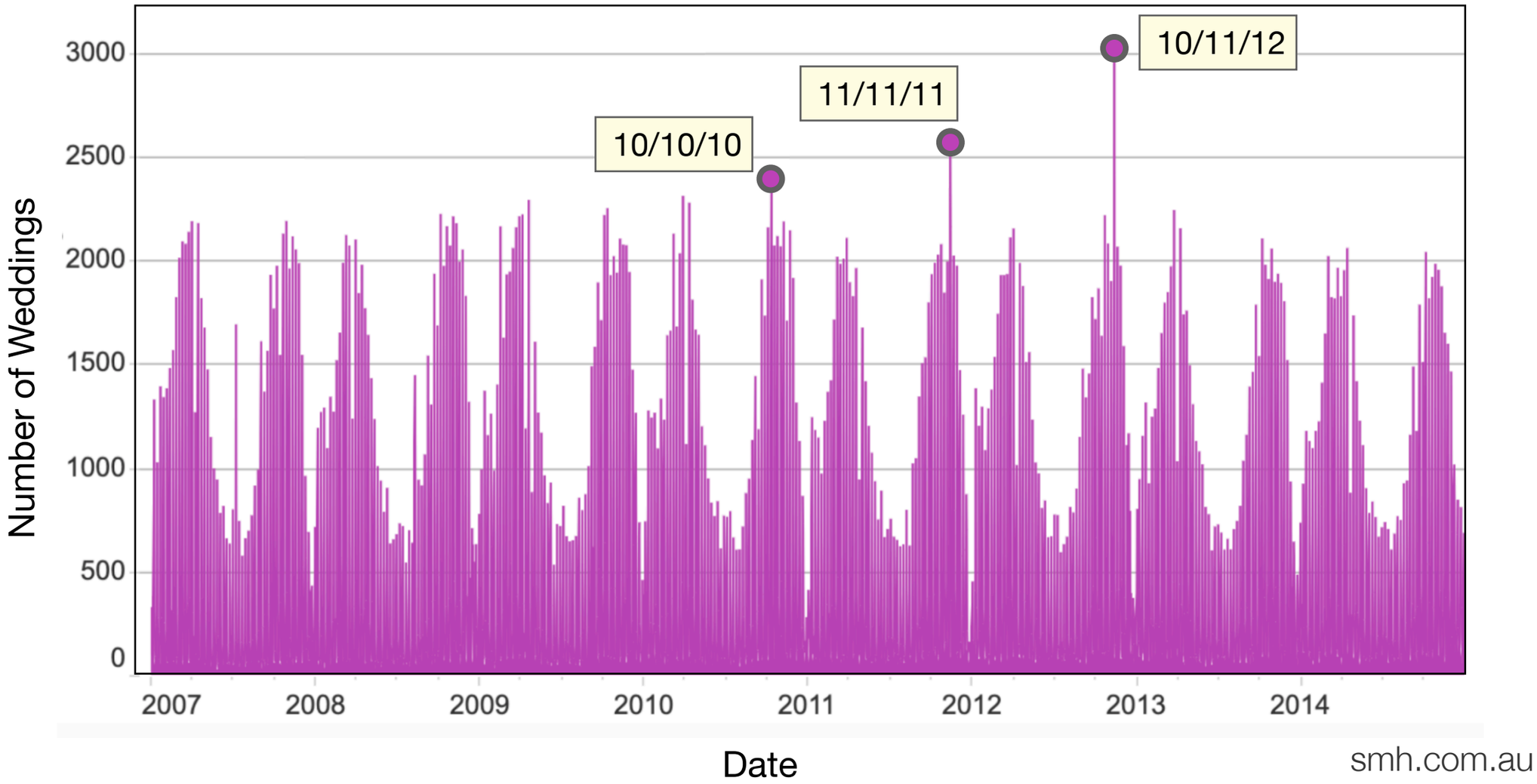
Time Series Data

- We will often plot a time series, and then look for trends, periodic behaviour, seasonal variations, step changes, outliers.
- **Example:** Time series of the mean monthly air temperature at Dublin Airport shows a strong seasonal signal.



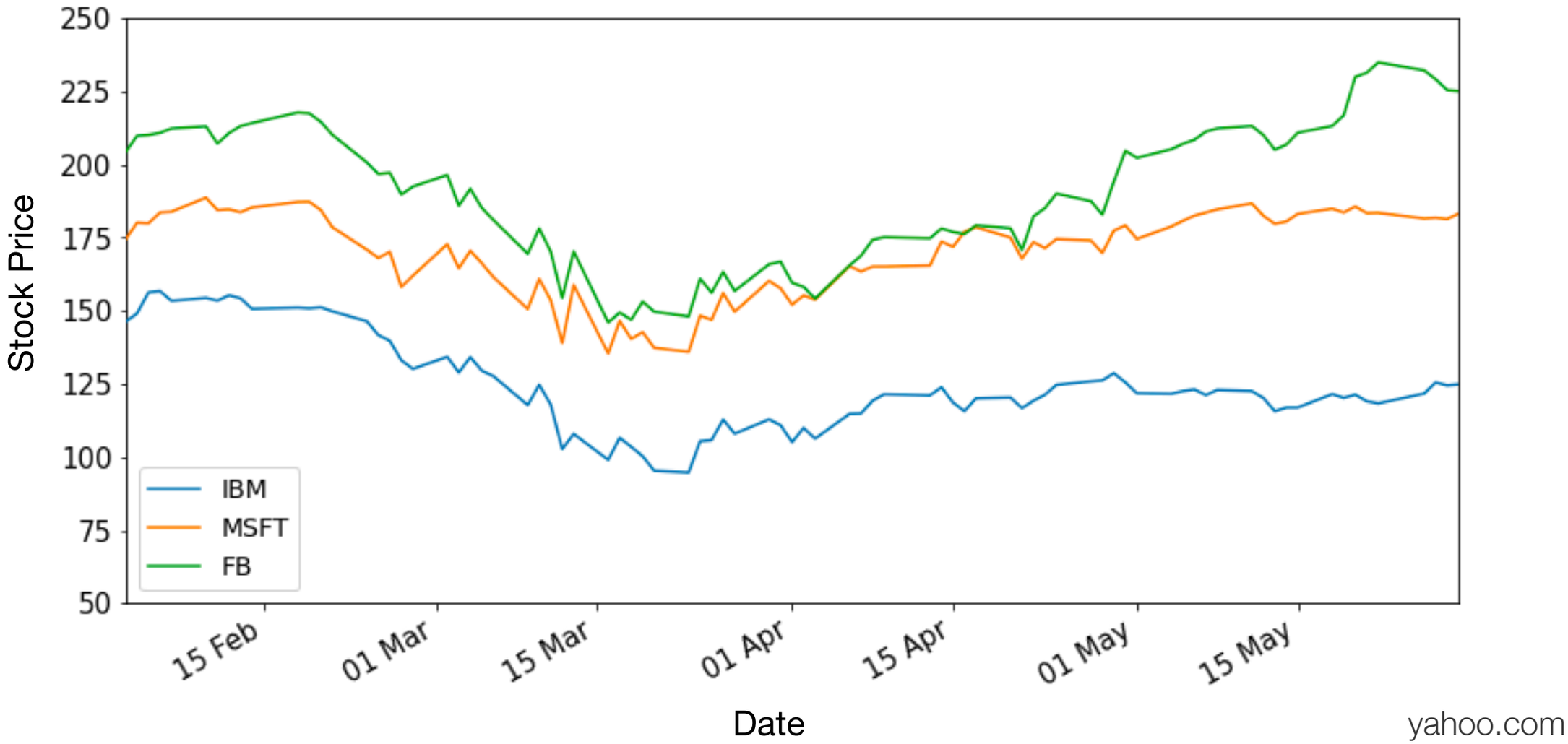
Time Series Data

- We will often plot a time series, and then look for trends, periodic behaviour, seasonal variations, step changes, outliers.
- **Example:** Time series of the number of weddings in Australia reveals peaks on repeating or sequential number patterns.



Comparing Time Series Data

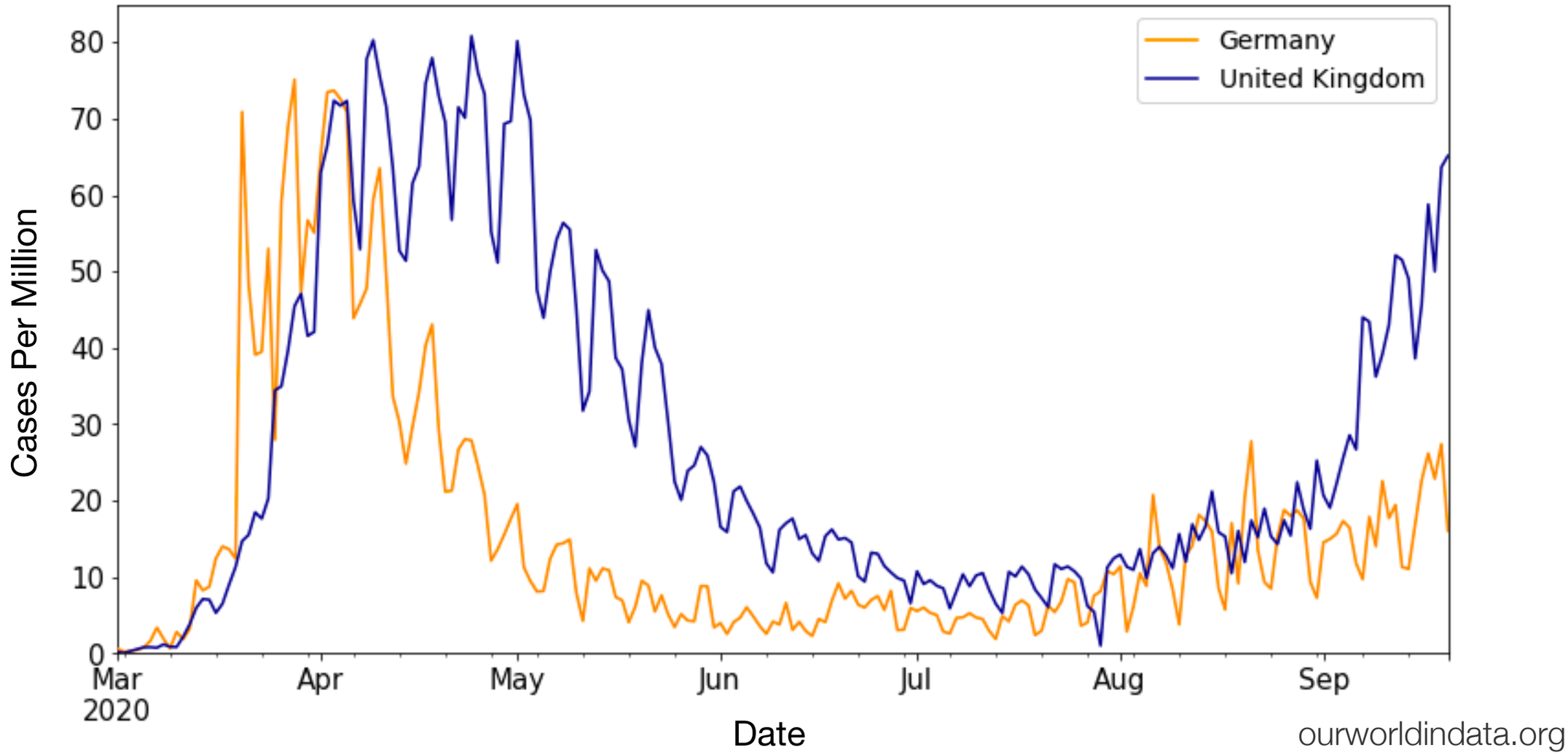
- In real-world applications we will often be tracking multiple variables over time, each represented by a different time series.
- **Example:** Comparison of stock closing prices for IBM, Microsoft, and Facebook over a 3 month period.



yahoo.com

Comparing Time Series Data

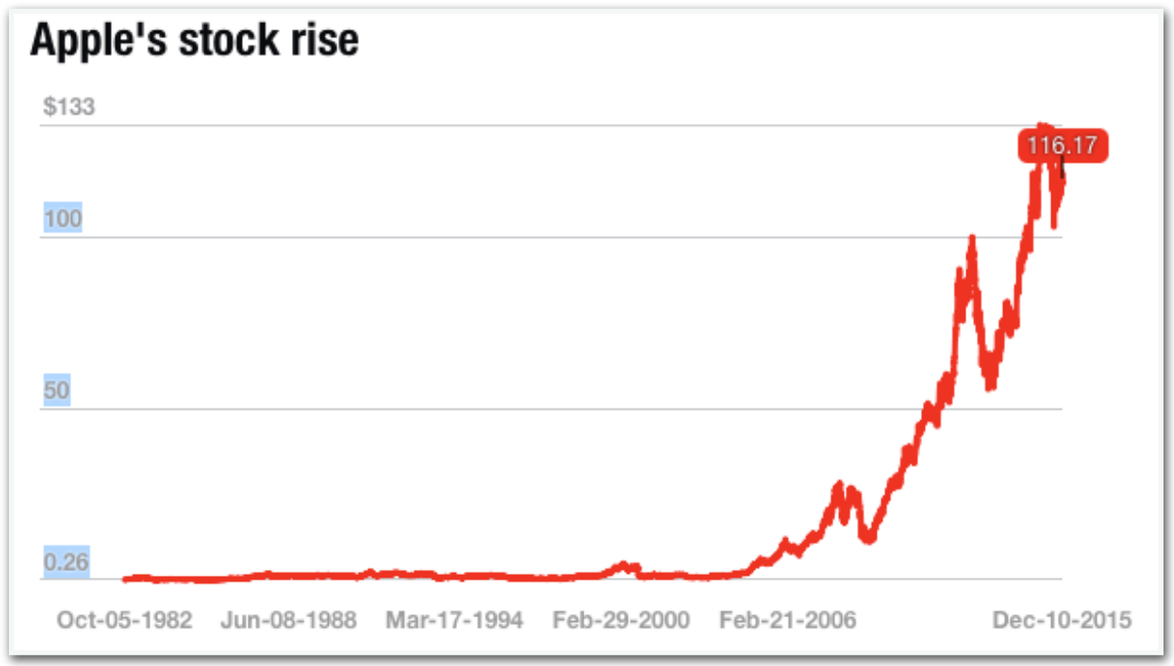
- In real-world applications we will often be tracking multiple variables over time, each represented by a different time series.
- **Example:** Comparison of daily number of COVID-19 cases per million for Germany and United Kingdom during mid-2020.



Characterising Time Series Data

- Key questions when looking at time series data:
  - Is the source sampled at equally-spaced intervals?
  - How long and/or rapidly growing is the available series?
  - What is the best resolution/frequency to inspect the data?
  - Are there any noisy or outlying values in the series?
  - Are there any missing values in the data?

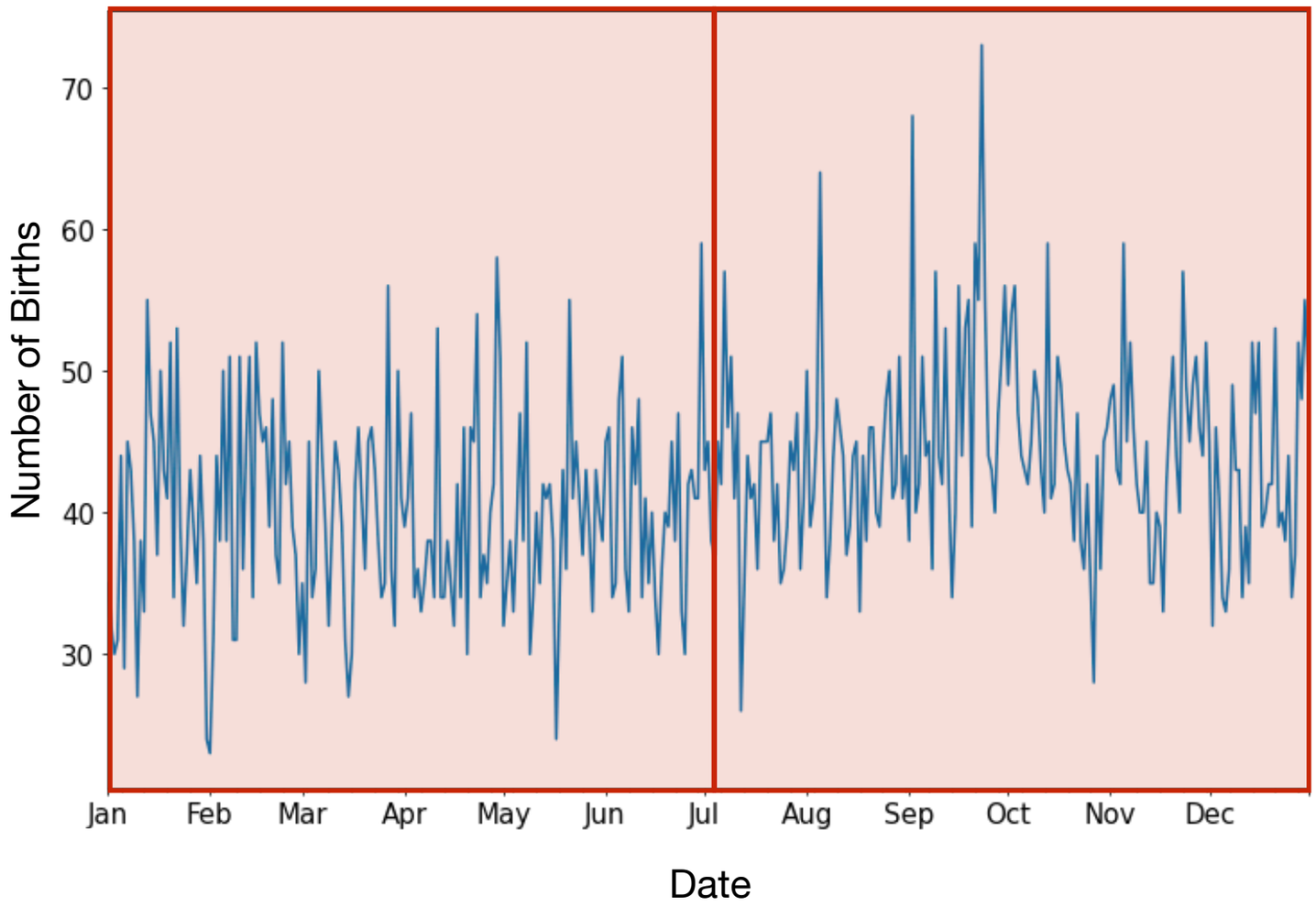
http://money.cnn.com



http://fortune.com

Stationary Time Series

- A time series is stationary if its properties do not depend on the time at which the series is observed.
- Summary statistics calculated on a stationary time series will be broadly consistent over time (e.g. mean, variance etc).



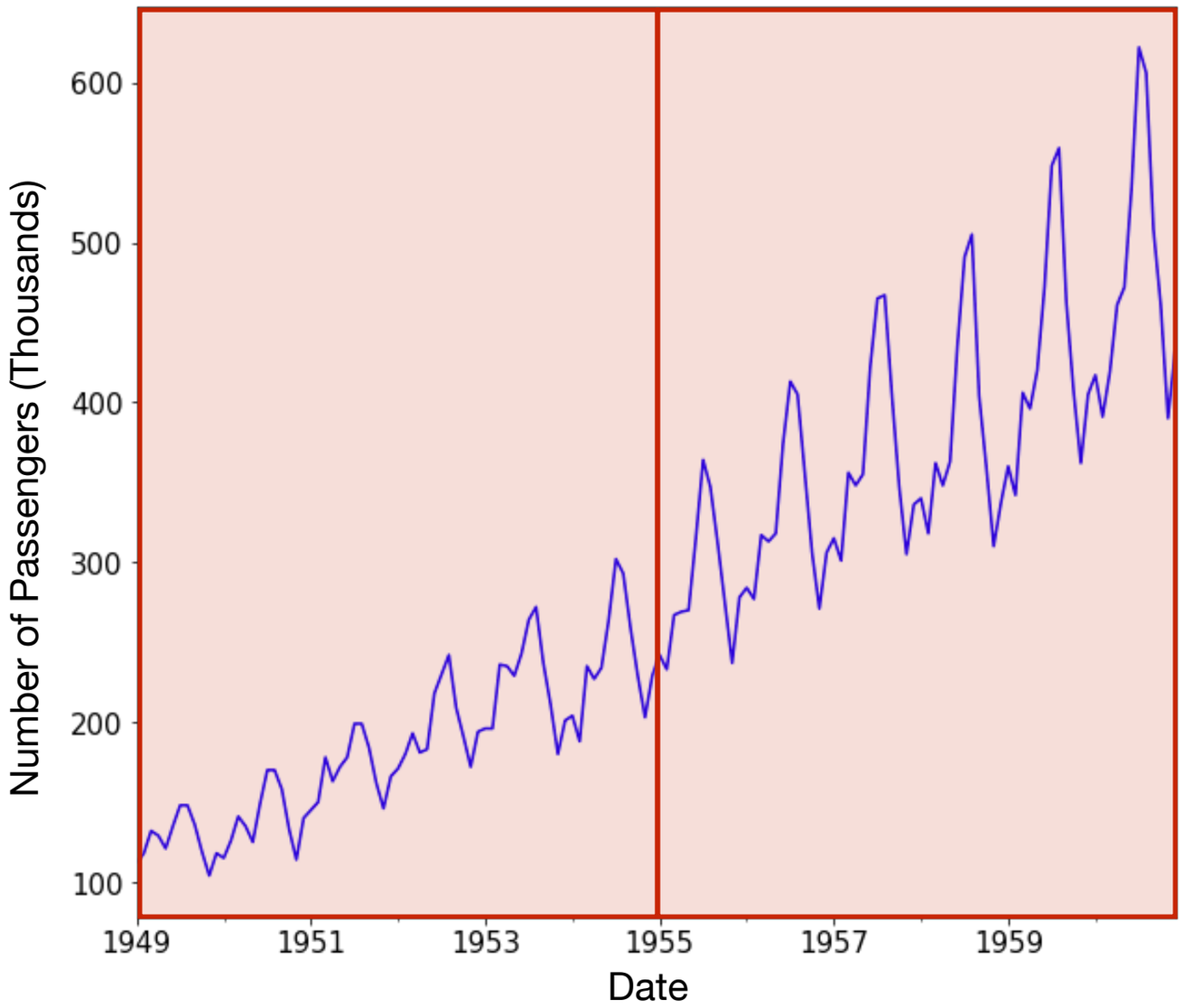
Example: Daily female births in California in 1959

01 January - 30 June
Mean=39.7
Variance=49.7

01 July - 31 December
Mean=44.2
Variance=48.7

Non-Stationary Time Series

- Many real-world time series datasets are non-stationary, as their statistical properties change over time - e.g. the mean and variance will not remain the same.



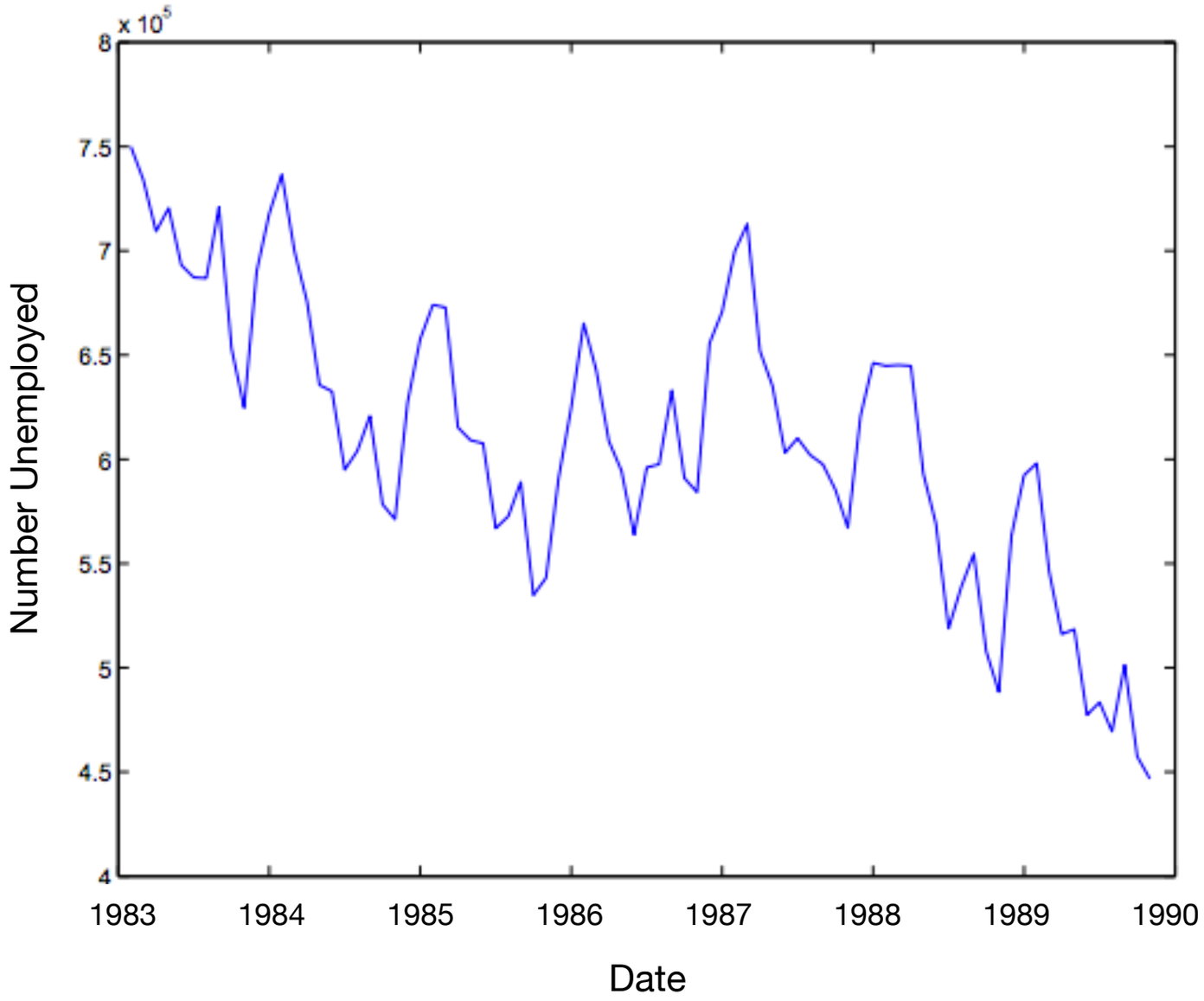
Example: International airline passenger monthly totals in thousands (1949-1960)

1949-1954
Mean=182.9
Variance=2275.7

1955-1960
Mean=377.7
Variance=7471.7

Non-Stationary Time Series

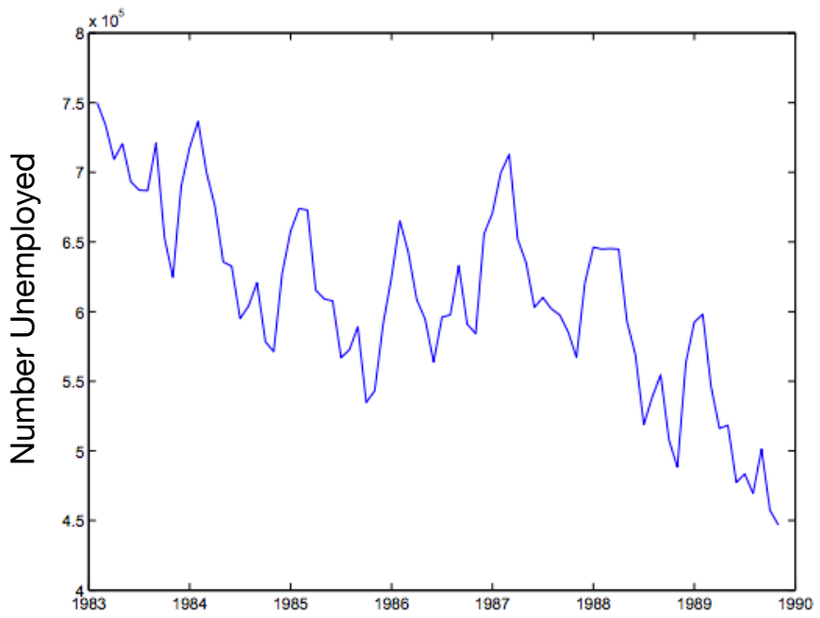
- Many real-world time series datasets are non-stationary, as their statistical properties change over time - e.g. the mean and variance will not remain the same.

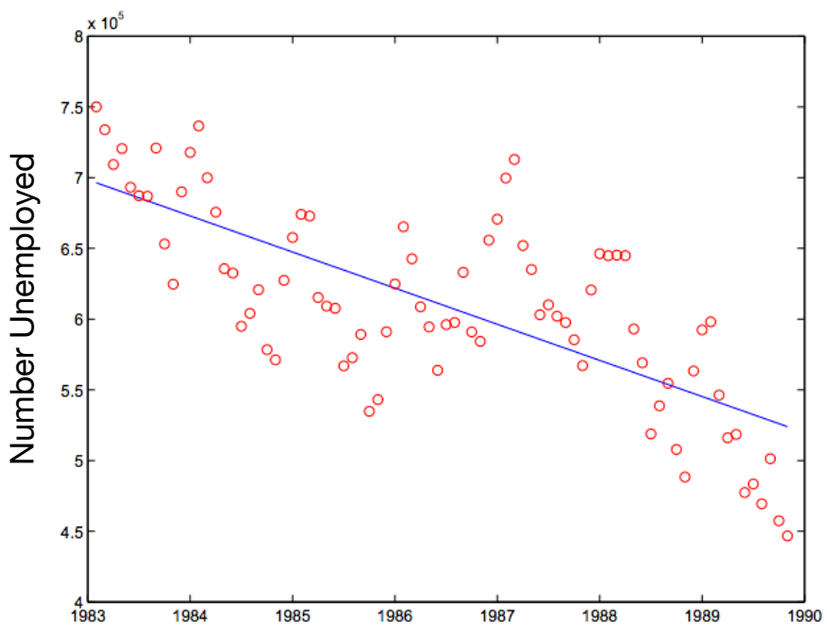


Example: Monthly number of people unemployed in Australia (Hipel & McLeod, 1994)

Time Series Components

- There are many reasons for a time series to be non-stationary. Most time series analysis is based on the modelling assumption that the observed series is the sum of three components:
  1. Trend: Shows the general tendency of the series to increase or decrease over a long period of time.

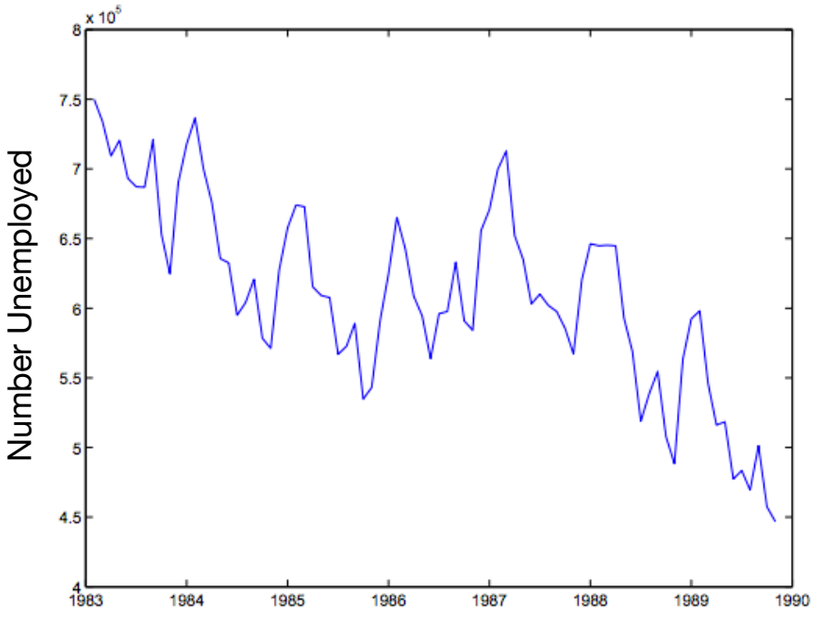


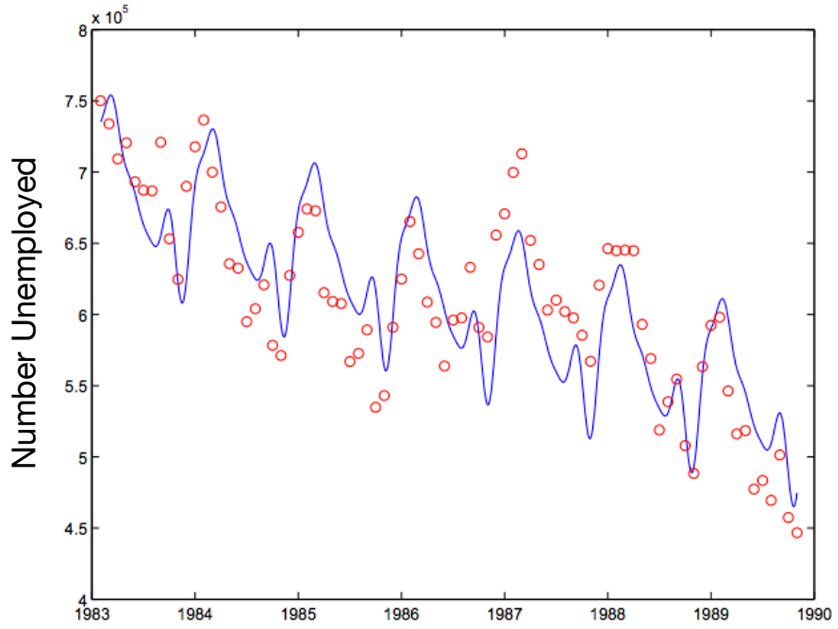


Downward trend in Australian unemployment during late 1980s

Time Series Components

- There are many reasons for a time series to be non-stationary. Most time series analysis is based on the modelling assumption that the observed series is the sum of three components:
  1. Trend: Shows the general tendency of the series to increase or decrease over a long period of time.
  2. Seasonality: Peaks and troughs that occur in a regular intervals - including daily, weekly, monthly, or yearly cycles.

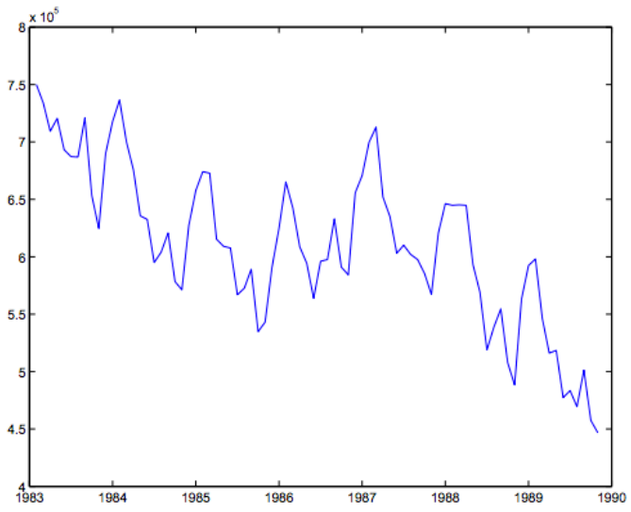


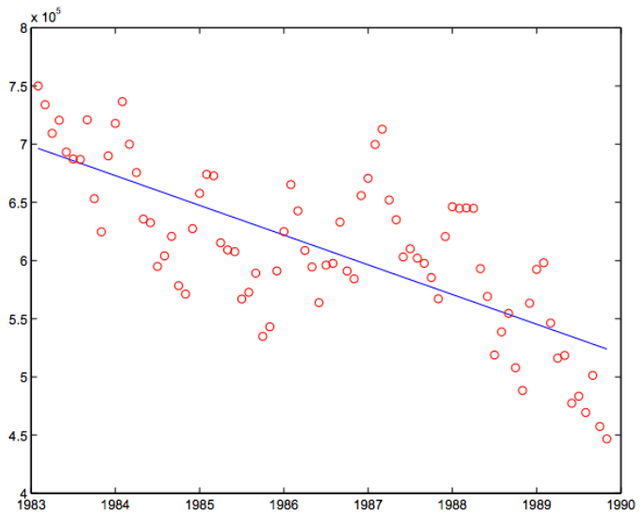


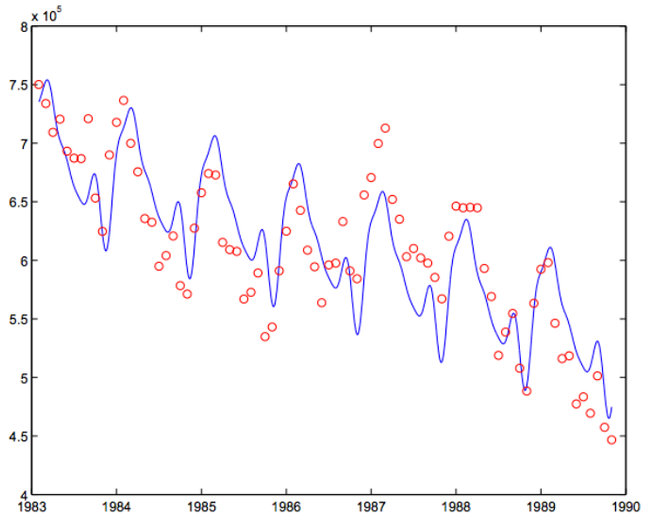
Employment in many Australian industry sectors is seasonal

Time Series Components

- There are many reasons for a time series to be non-stationary. Most time series analysis is based on the modelling assumption that the observed series is the sum of three components:
  1. Trend: Shows the general tendency of the series to increase or decrease over a long period of time.
  2. Seasonality: Peaks and troughs that occur in a regular intervals - including daily, weekly, monthly, or yearly cycles.
  3. Noise: The random fluctuations in the data which are left when all the components have been removed.







Time Series Data Formats

- Most basic format for time series data involves observations with a date/time field and a single numeric target. The format of the date will depend on the data source and application domain.
- Data preprocessing steps will often be required to extract dates in a suitable format, or change the frequency of the data.

Date	Stock Price
2019-05-02	139.59
2019-05-03	140.25
2019-05-06	140.38
2019-05-07	137.64
2019-05-08	138.00
2019-05-09	135.34
2019-05-10	135.32
2019-05-13	131.42
2019-05-14	133.31
...	...

Month	Passengers
1949-01	112
1949-02	118
1949-03	132
1949-04	129
1949-05	121
1949-06	135
1949-07	148
1949-08	148
1949-09	136
...	...

Time	Contacts
2020-08-12 13:39:21+01:00	8
2020-08-12 16:33:14+01:00	11
2020-08-12 20:40:22+01:00	8
2020-08-12 22:43:20+01:00	5
2020-08-13 12:13:17+01:00	12
2020-08-14 18:27:49+01:00	24
2020-08-14 20:29:43+01:00	7
2020-08-17 22:46:02+01:00	26
2020-08-18 10:08:46+01:00	4
...	...

Python Date and Time Types

- The built-in Python `datetime` module includes types for date and time data, as well as calendar-related functionality.

Type	Description
`datetime.date`	An idealised date, independent of any particular time of day.
`datetime.time`	An idealised time, independent of any particular day
`datetime.datetime`	A combination of a date and a time. Most frequently used.

```
from datetime import datetime
datetime.now()
```

```
datetime.datetime(2020, 8, 13, 18, 28, 34, 403741)
```

We can get the current local date and time by calling `datetime.now()`

- We can create a new `datetime` value by specifying either the date alone or both a date and a time:

```
datetime(2020, 1, 7)
```

```
datetime.datetime(2020, 1, 7, 0, 0)
```

```
datetime(2020, 1, 7, 13, 30)
```

```
datetime.datetime(2020, 1, 7, 13, 30)
```

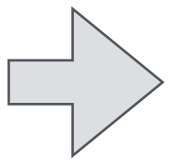
Python Date and Time Types

- From a `datetime` value we can access its individual parts - i.e. the date and the time components.

```
from datetime import datetime
d = datetime.now()
d
```

```
datetime.datetime(2020, 8, 13, 18, 30, 8, 583908)
```

```
print(d.year)
print(d.month)
print(d.day)
print(d.hour)
print(d.minute)
print(d.second)
print(d.microsecond)
```



```
2020
8
13
18
30
8
583908
```

Extract the individual values of every part of the datetime.

Get an associated date or a time value.

```
d.date()
```

```
datetime.date(2020, 8, 13)
```

```
d.time()
```

```
datetime.time(18, 30, 8, 583908)
```

Python Date and Time Types

- A `timedelta` is a Python data type which stores the temporal difference between two `datetime` values:

```
d1 = datetime(2020, 1, 7, 11, 15)
d2 = datetime(2020, 2, 15, 11, 35)
diff = d2 - d1
diff.days, diff.seconds
```

```
(39, 1200)
```

Subtract one date from another to get a time delta, expressed in days and seconds

- We can add or subtract a `timedelta` value to an existing `datetime` value to get a new shifted date:

```
from datetime import timedelta
d1 + timedelta(3)
```

```
datetime.datetime(2020, 1, 10, 11, 15)
```

Add 3 days and 0 seconds to the existing date

```
d1 + timedelta(0,60)
```

```
datetime.datetime(2020, 1, 7, 11, 16)
```

Add 0 days and 60 seconds to the existing date

Converting Between Strings and Dates

- Python `datetime` values can be formatted as strings with `strftime()` and special formatting codes.
- Each code is a placeholder for a date or time component of a datetime value.

```
from datetime import datetime
d = datetime(2020, 3, 1, 9, 30)
```

The date formatting codes change how the datetime value is displayed.

```
d.strftime('%d/%m/%y')
d.strftime('%Y-%m-%d')
d.strftime('%H:%M:%S')
d.strftime('%Y-%m-%d %H:%M')
d.strftime('%d %B %Y')
d.strftime('%a %d %Y')
```

```
01/03/20
2020-03-01
09:30:00
2020-03-01 09:30
01 March 2020
Thu 01 2020
```

Code	Meaning
`%Y`	4-digit year
`%y`	2-digit year
`%m`	2-digit month
`%d`	2-digit day
`%H`	Hour (24 hour clock)
`%h`	Hour (12 hour clock)
`%M`	2-digit minute
`%S`	seconds
`%a`	short day name
`%A`	long day name
`%b`	short month name
`%B`	long month name

Converting Between Strings and Dates

- We can also use the same codes to parse and extract dates from strings with the `strptime()` function.
- The specification of codes needs to match the format of the string.

```
s = "20-01-03"
datetime.strptime(s, '%y-%m-%d')
```

```
datetime.datetime(2020, 1, 3, 0, 0)
```

```
s = "01/03/2020"
datetime.strptime(s, '%d/%m/%Y')
```

```
datetime.datetime(2020, 3, 1, 0, 0)
```

```
s = "01/03/20 14:56"
datetime.strptime(s, '%d/%m/%y %H:%M')
```

```
datetime.datetime(2020, 3, 1, 14, 56)
```

Code	Meaning
`%Y`	4-digit year
`%y`	2-digit year
`%m`	2-digit month
`%d`	2-digit day
`%H`	Hour (24 hour clock)
`%h`	Hour (12 hour clock)
`%M`	2-digit minute
`%S`	seconds
`%a`	short day name
`%A`	long day name
`%b`	short month name
`%B`	long month name

Time Series in Pandas

- Most basic kind of time series data in Pandas is a Series indexed by timestamps, often `datetime` values.

```
dates = [datetime(2016,4,2), datetime(2016,4,4), datetime(2016,4,6),
   datetime(2016,4,8), datetime(2016,4,10)]
sales = [10,11,12,16,11]
```

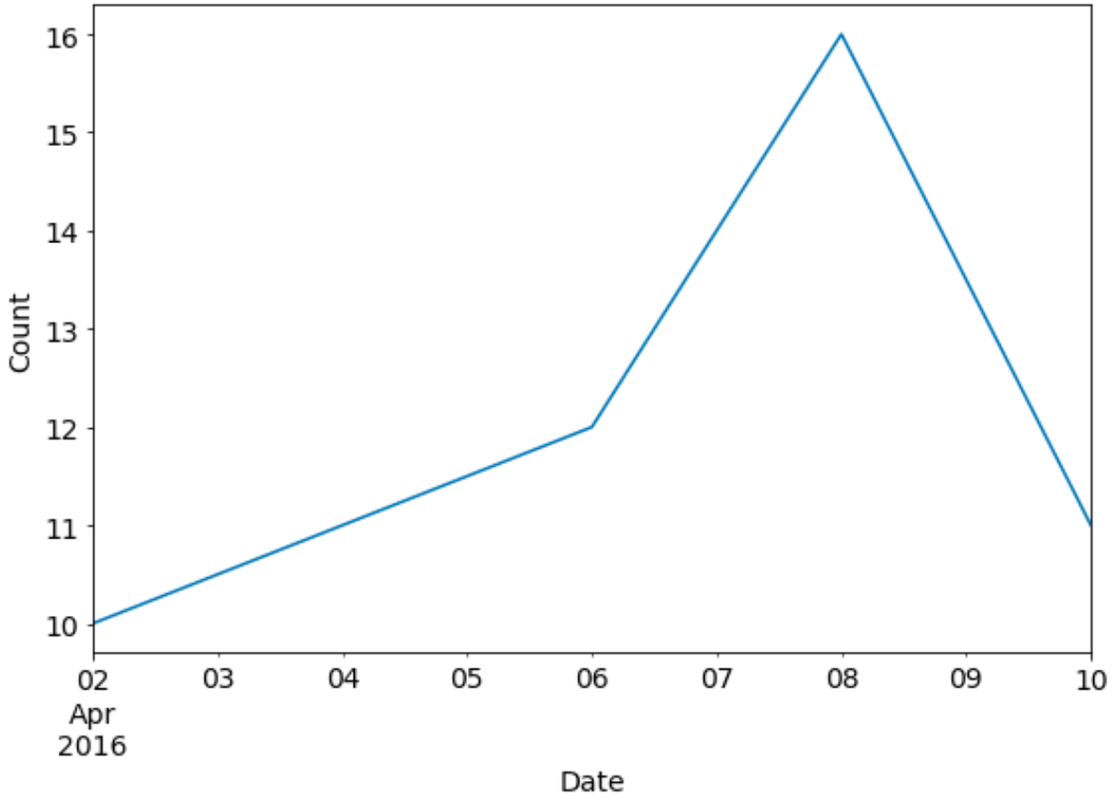
```
ts = pd.Series(sales, index=dates)
ts
```

```
2016-04-02    10
2016-04-04    11
2016-04-06    12
2016-04-08    16
2016-04-10    11
```

Create a Series with 5 values, each indexed by a different `datetime` value

```
ax = ts.plot(figsize=(9,6),fontsize=14)
ax.set_xlabel("Date",fontsize=14)
ax.set_ylabel("Count",fontsize=14)
```

We can visualise this simple time series with a Pandas line plot



Time Series in Pandas

- A Pandas time series can be indexed and sliced in the same way as a normal Series:

```
ts["2016-04-02"]
```

```
10
```

```
ts["2016-04-02":"2016-04-04"]
```

```
2016-04-02    10
2016-04-04    11
```

- For longer time series, we can easily select slices of data for a specific month or year:

```
import numpy as np
lts = pd.Series(np.random.randn(500), index=pd.date_range('1/1/2020', periods=500))
```

```
lts["2020"].head()
```

```
2020-01-01   -1.785851
2020-01-02   -0.792898
2020-01-03    0.714274
2020-01-04   -0.271149
2020-01-05    0.660691
```

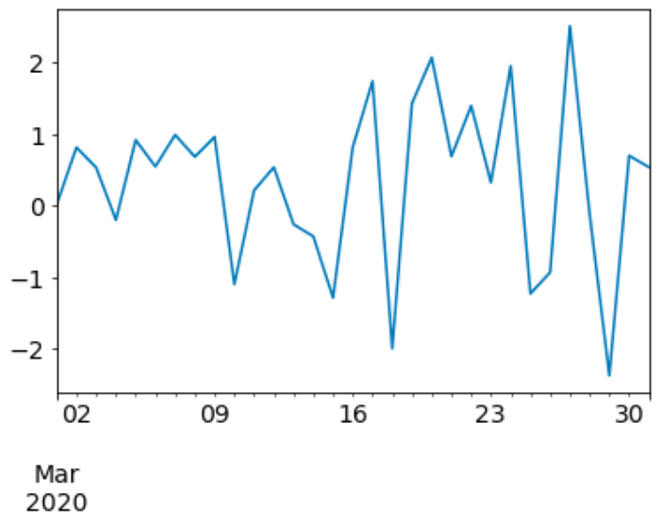
Get all of 2020

```
lts["2020-03"].head()
```

```
2020-03-01    0.014804
2020-03-02    0.813409
2020-03-03    0.530261
2020-03-04   -0.200965
2020-03-05    0.916792
```

Get March 2020

```
lts["2020-03"].plot()
```



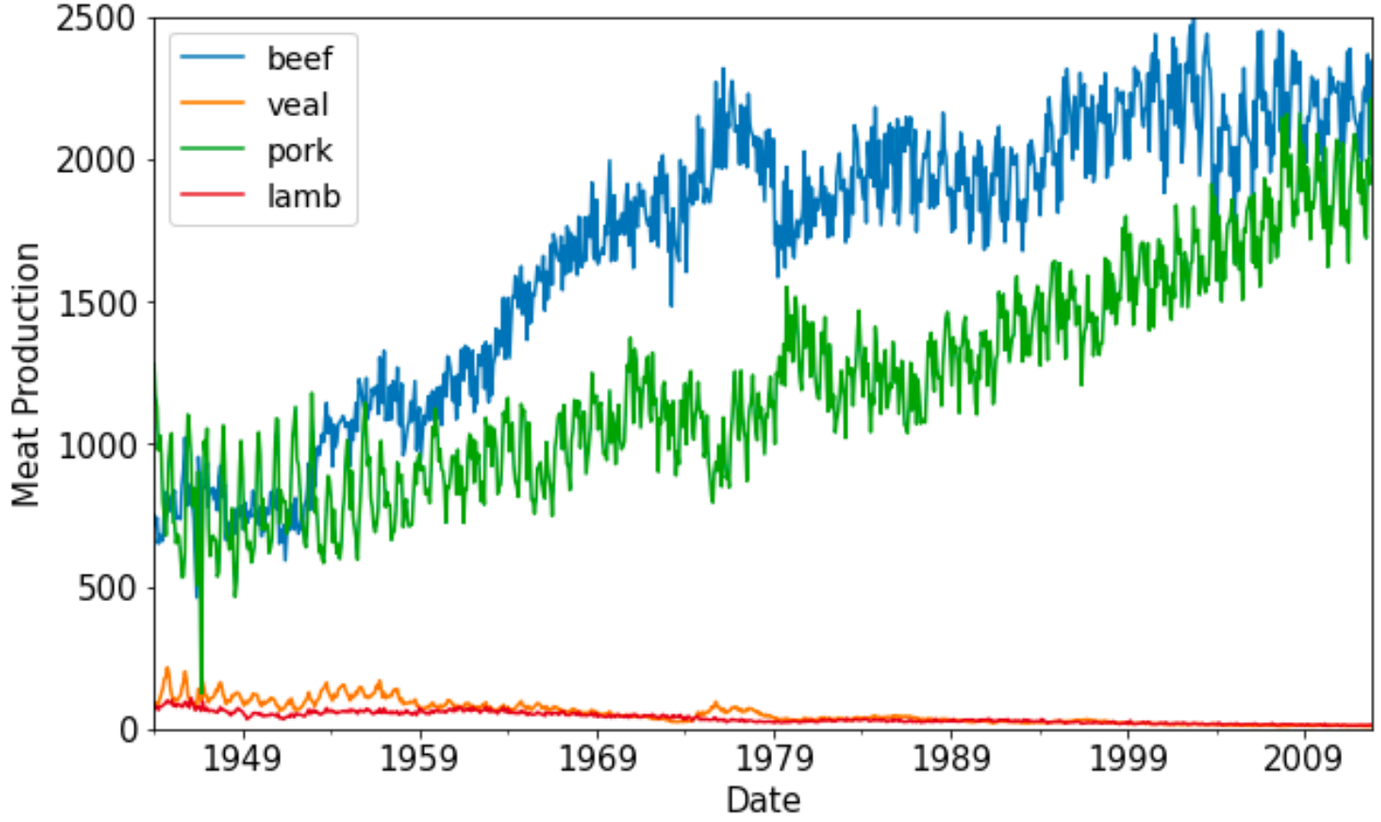
Time Series in Pandas

- In real-world applications we will usually be tracking multiple variables over time, each represented by a different series.
- Load a time series dataset from a CSV file as a Pandas Data Frame. Specify `parse_dates` to parse the index field as a date:

```
df = pd.read_csv("agri-meat.csv",index_col="date",parse_dates=True)
df.head()
```

date	beef	veal	pork	lamb
1944-01-01	751.0	85.0	1280.0	89.0
1944-02-01	713.0	77.0	1169.0	72.0
1944-03-01	741.0	90.0	1128.0	75.0
1944-04-01	650.0	89.0	978.0	66.0
1944-05-01	681.0	106.0	1029.0	78.0

```
ax = df.plot(figsize=(10, 6))
ax.set_xlabel("Date")
ax.set_ylabel("Meat Production")
```



Time Series in Pandas

- Pandas has functionality for aggregating date and time based data. For example, we can group the data by year:

```
df_year = df.groupby(df.index.year).sum()
df_year.head()
```

Group by the "year" component of the index, which is a date

date	beef	veal	pork	lamb
1944	8801.0	1629.0	11502.0	1001.0
1945	9936.0	1552.0	8843.0	1030.0
1946	9010.0	1329.0	9220.0	946.0
1947	10096.0	1493.0	8811.0	779.0
1948	8766.0	1323.0	8486.0	728.0

```
def to_decade(date_value):
    return (date_value.year // 10) * 10
df_decade = df.groupby(to_decade).sum()
df_decade.head()
```

If we want to group the data by decade, we need to define a custom aggregation function which will take the year of a date and "floor" it - i.e. round it down to the nearest 10

	beef	veal	pork	lamb
1940	55751.0	8566.0	55737.0	5071.0
1950	119161.0	12693.0	98450.0	6724.0
1960	177754.0	8577.0	116587.0	6873.0
1970	228947.0	5713.0	132539.0	4256.0
1980	230100.0	4278.0	150528.0	3394.0

Sampling Frequency

- A key characteristic of a time series is how frequently observations are spaced in time. Normally observations will be evenly spaced.
- Sampling frequency refers to how often the observations of a time series occur. For analysis, we may wish to alter the frequency.
- Resampling: Process of converting time series data from one frequency to another. This is done in Pandas via the `resample()` function.

- We can downsample - i.e. aggregate data to a lower frequency:

```
ts = pd.Series(np.random.randn(120), index=pd.date_range('1/1/2020', periods=120))
```

```
ts.head()
```

```
2020-01-01    0.707904
2020-01-02   -0.641869
2020-01-03    0.045906
2020-01-04   -1.378475
2020-01-05   -0.486489
```

```
ts.resample("M").mean()
```

```
2020-01-31   -0.019206
2020-02-29   -0.025153
2020-03-31    0.110398
2020-04-30    0.037361
Freq: M, dtype: float64
```

Convert from day to month ("M") frequency, by taking the mean of the values in each month.

Sampling Frequency

- A key characteristic of a time series is how frequently observations are spaced in time. Normally observations will be evenly spaced.
- Sampling frequency refers to how often the observations of a time series occur. For analysis, we may wish to alter the frequency.
- Resampling: Process of converting time series data from one frequency to another. This is done in Pandas via the `resample()` function.

- We can also upsample - i.e. convert lower to higher frequency:

```
ts = pd.Series(np.random.randn(5), index=pd.date_range('1/1/2020', periods=5))
```

```
ts.head()
```

```
2020-01-01   -0.118638
2020-01-02   -1.387410
2020-01-03   -0.077690
2020-01-04    0.168886
2020-01-05    1.049955
```

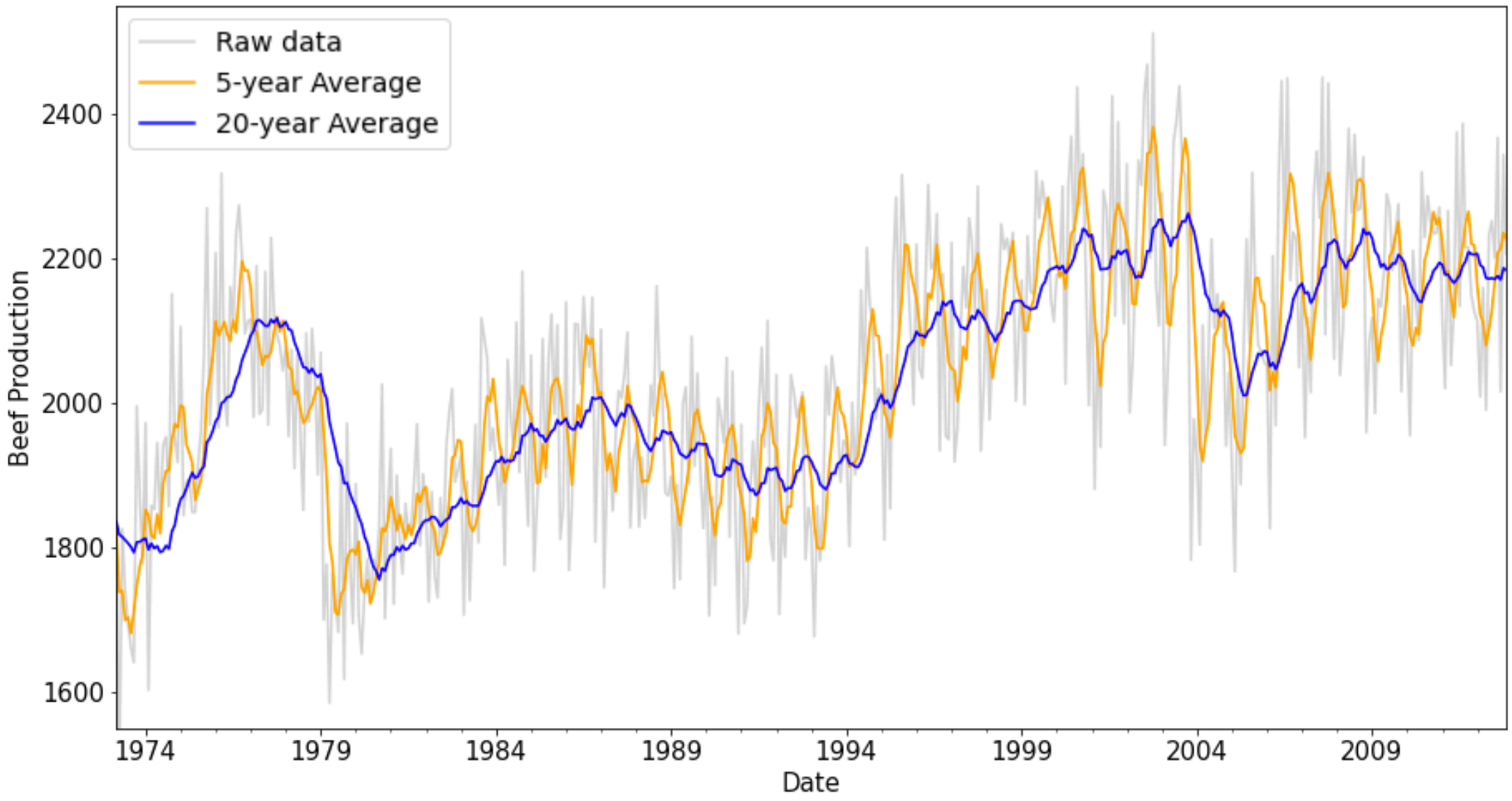
```
ts.resample("H").mean()
```

```
2020-01-01 00:00:00   -0.118638
2020-01-01 01:00:00         NaN
2020-01-01 02:00:00         NaN
2020-01-01 03:00:00         NaN
2020-01-01 04:00:00         NaN
```

Upsample to hourly frequency ("H"). The rows that are added in between have missing values.

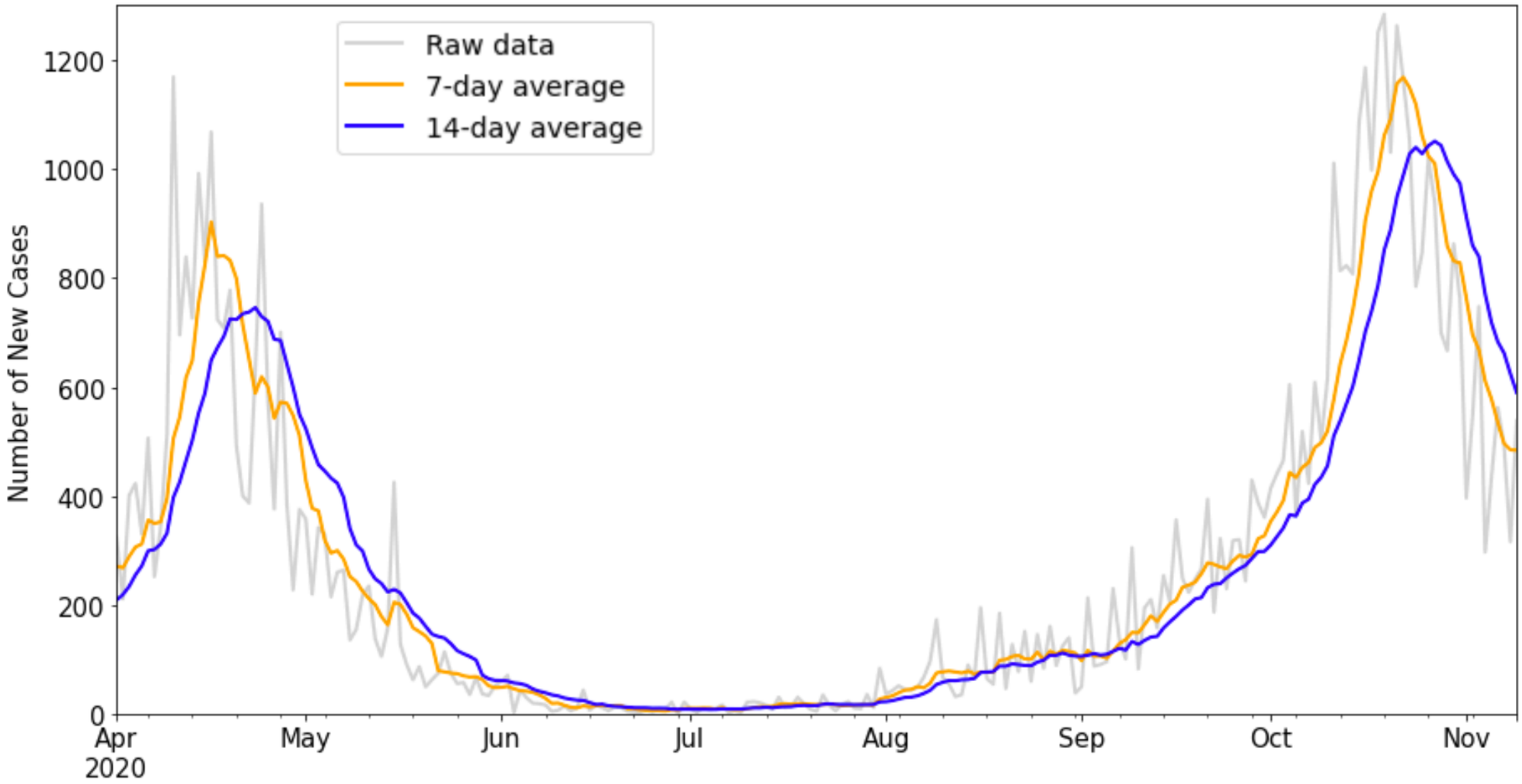
Smoothing Time Series Data

- A moving average (also called rolling mean) is often applied to time series to smooth short-term variations and highlight the underlying trend in the series.
- **Example:** US beef production (1973-2012)



Smoothing Time Series Data

- A moving average (also called rolling mean) is often applied to time series to smooth short-term variations and highlight the underlying trend in the series.
- **Example:** Ireland COVID-19 daily new cases (Apr-Nov 2020)



Moving Average

- A simple moving average divides a series into overlapping regions of a fixed size, called "windows" (e.g. 7 days, 14 days, 1 month).
- Each value is calculated as the average of all the observations in the current window. We then"roll" the window forward by one observation and calculate the next value.

Month	Sales	Moving Average (3 Months)
January	3537	-
February	1827	-
March	1467	2277
April	3807	2367
May	2097	2457
June	1737	2547
July	4077	2637
August	2367	2727
September	2007	2817
October	4347	2907
November	2637	2997
December	2277	3087

Original series has 12 monthly observations for product sales.

We could compute a moving average, where the window size is 3 months (i.e. each window has 3 observations).

$$\frac{3537 + 1827 + 1467}{3} = 2277$$

$$\frac{1827 + 1467 + 3807}{3} = 2367$$

Moving Averages in Python

- In Pandas we can compute a moving average for a Series or Data Frame by calling its `rolling()` function and specifying a window size (i.e. number of observations).
- We apply the function `mean()` to the result to produce a new series which contains the averages.

```
df = pd.read_csv("stock-google-close.csv", index_col="Date", parse_dates=True)
ma3 = df["Close"].rolling(3).mean()
```

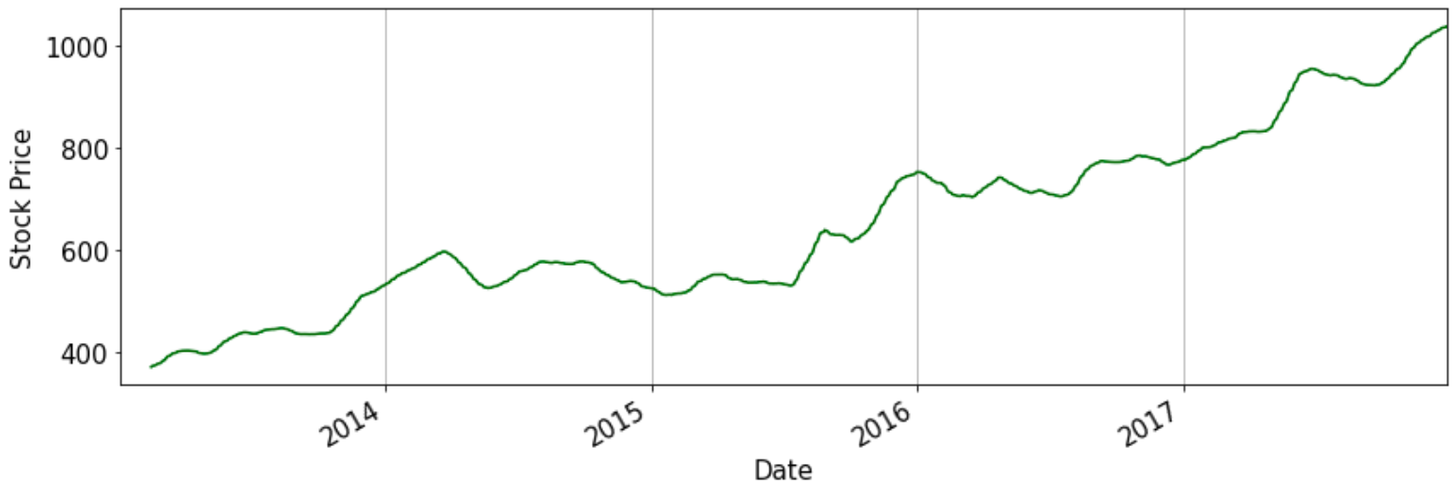
Apply moving average to daily Google stock price

Window size 3: average 1st, 2nd & 3rd value, then 2nd, 3rd & 4th...

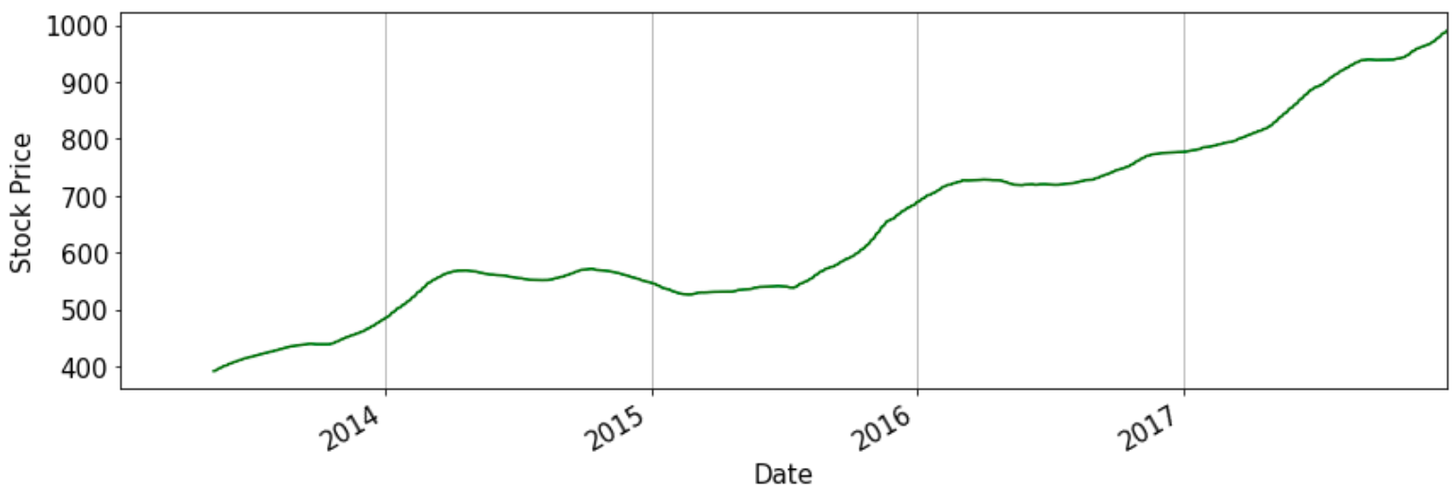
i.e. 3-day moving average

Example: Moving Averages

Google closing stock prices:
7-day moving average



Google closing stock prices:
30-day moving average

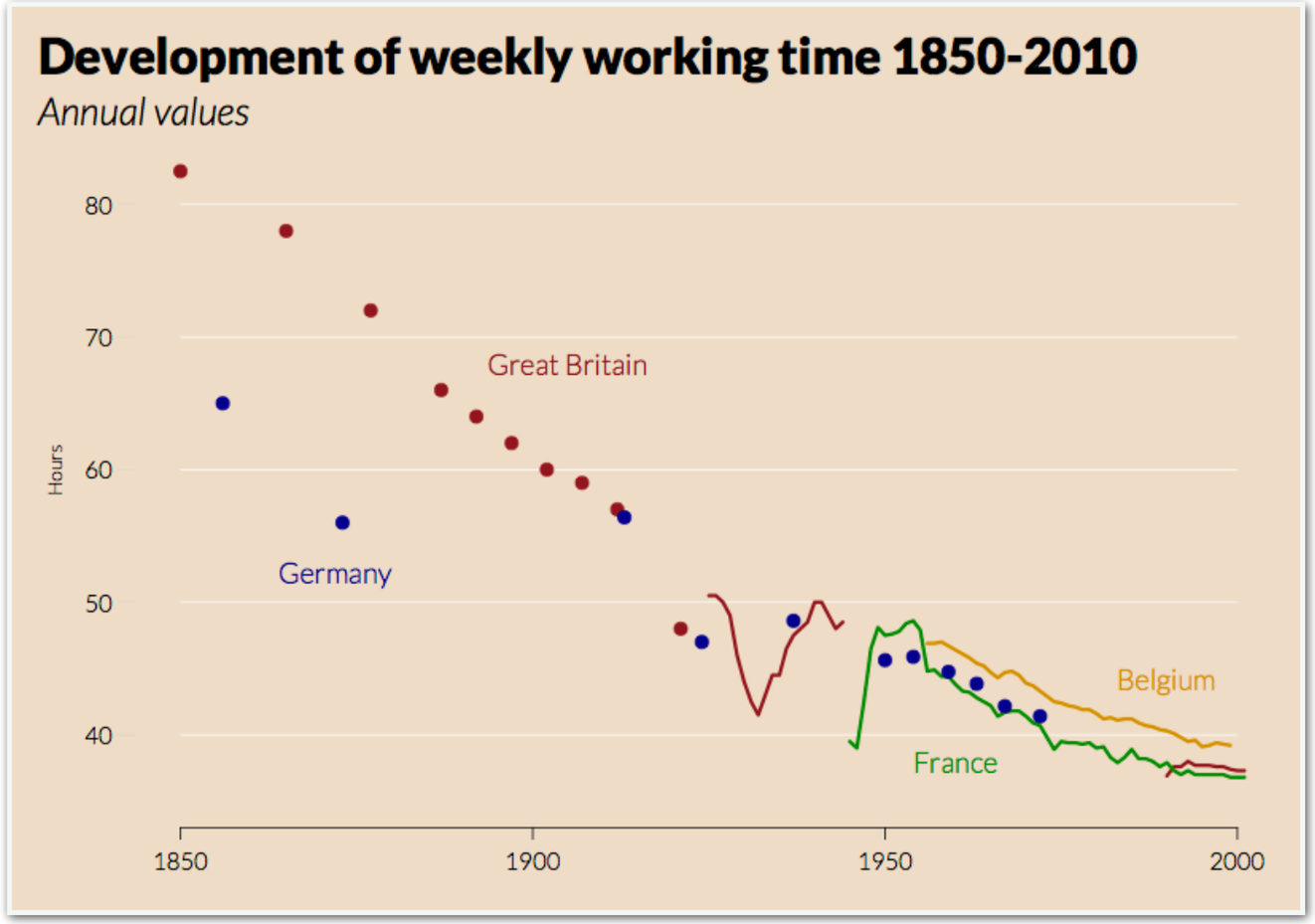


Google closing stock prices:
90-day moving average

A larger window will lead
to a smoother series.

Missing Data in Time Series

- In many time series, values might be missing for certain time periods.
- If there are missing values, there are two ways to deal with the incomplete data:
  1. Drop the entire record.
  2. Impute or estimate the missing information.

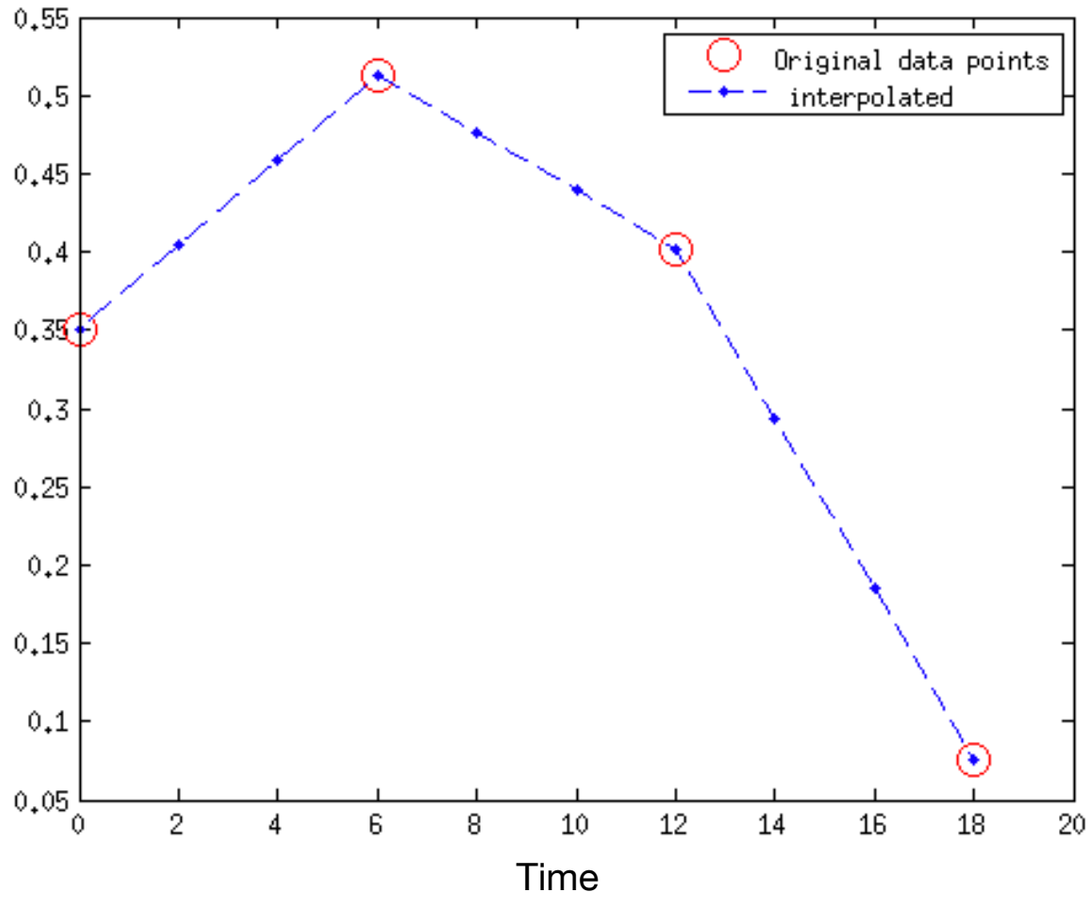


Source: Data Visualisation with R

- To estimate missing values, we can apply standard methods (e.g. replace with time series mean or median value; replace with values randomly selected from the remaining data).

Missing Data in Time Series

- There are time series-specific methods to estimate missing values.
  - Last observation carried forward (LOCF): Use the most recent previous non-missing value to estimate the current missing value.
  - Next observation carried backward (NOCB): Use the next non-missing value to estimate the current missing value.
  - Linear interpolation: Fill in the missing data by assuming that the relationship between the variables follows a line.
- Note: these methods may not work well if there is a strong seasonal component in the time series.



Time Series Forecasting

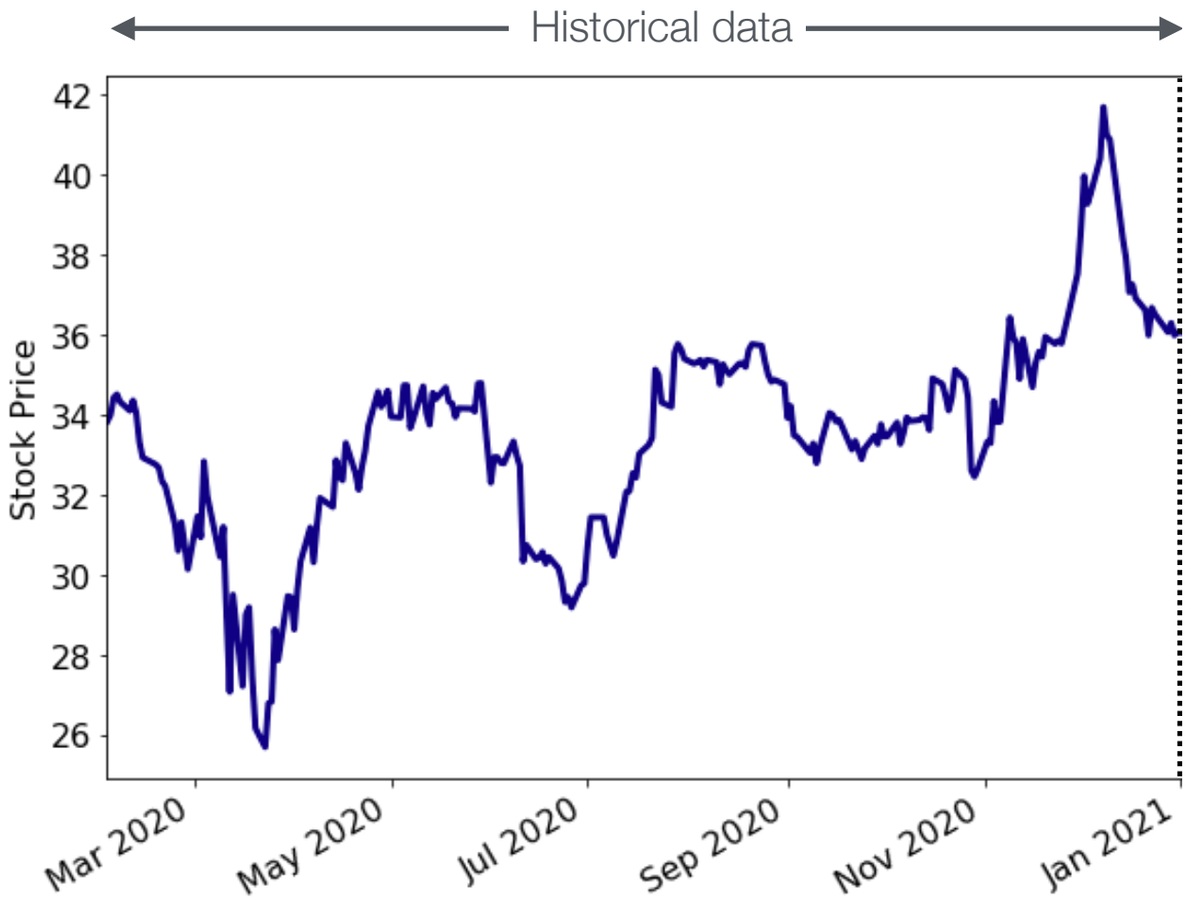
- Time series forecasting uses a model to predict future values based on previously observed values in a time series.
- Input: Past data provided to a model to make a forecast.
- Output: Prediction for a future time point beyond the input data.
- Accuracy of the prediction will depend on the availability of past data, the trend in the data, and the forecasting horizon.

Pfizer daily stock prices

Input: Historic stock data from 03/02/2020 to 31/12/2020

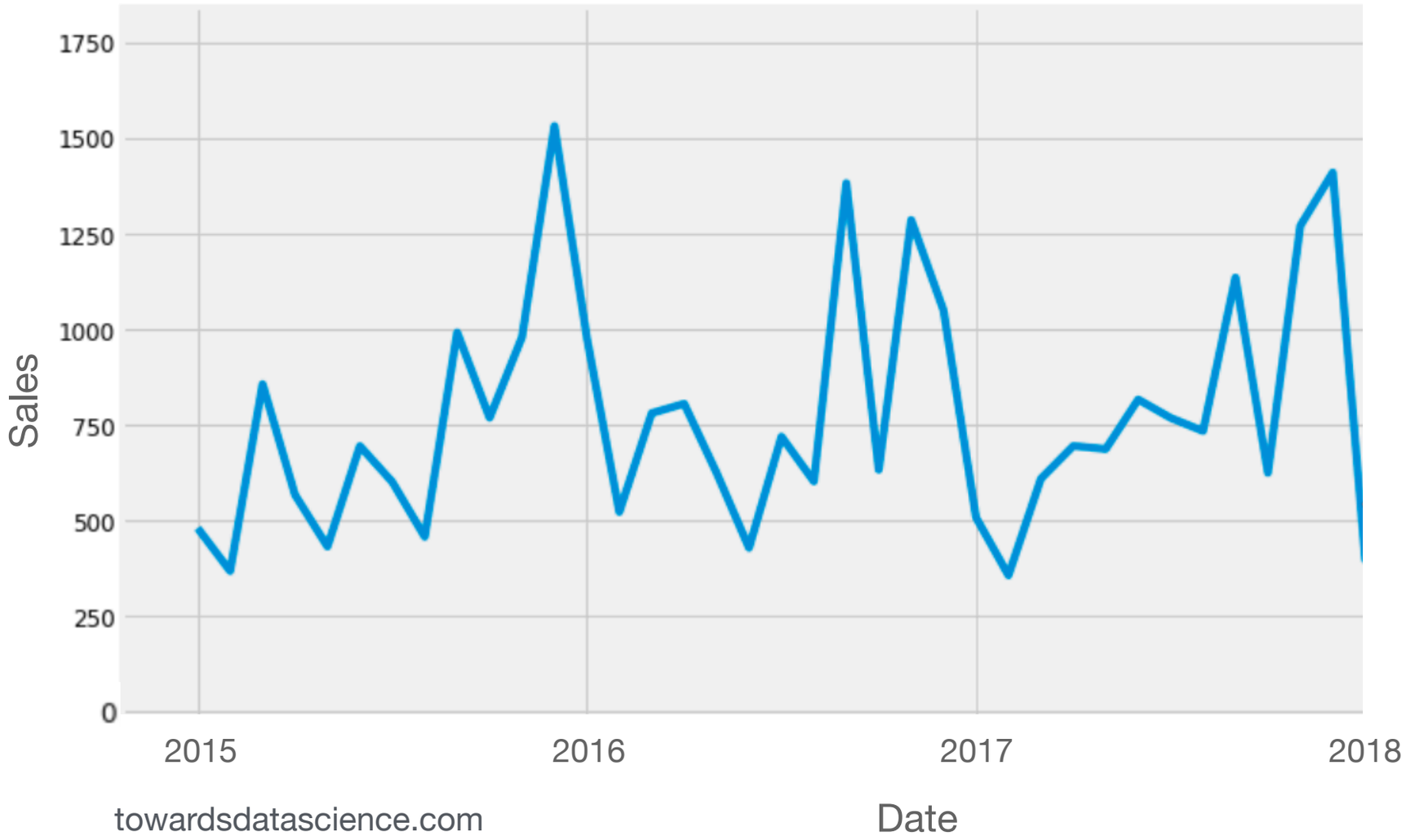
Horizon: 1 day

Output: Forecasts for 01/01/2021 to 10/05/2021



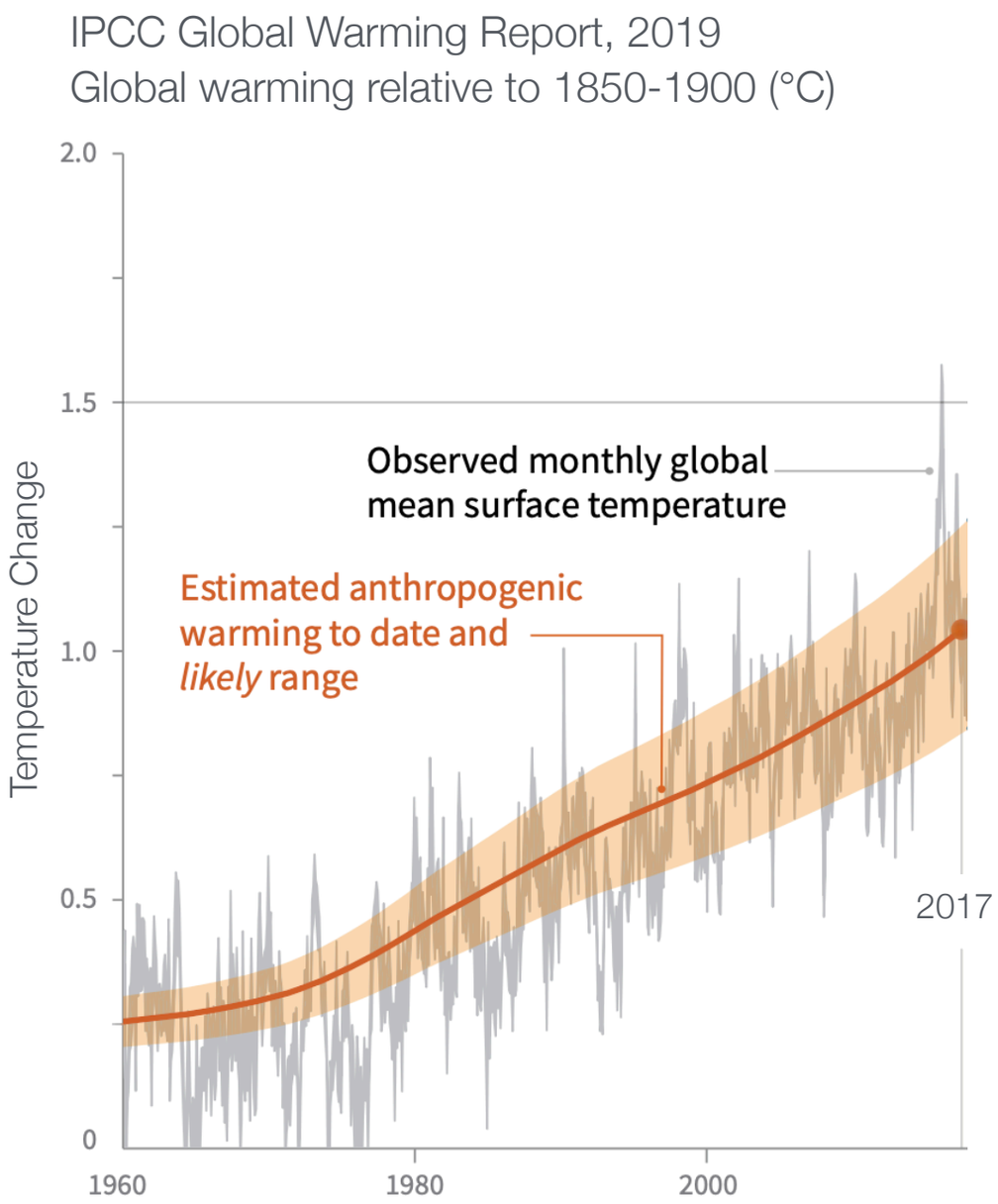
Time Series Forecasting

- **Application:** Predict future daily sales based on historic sales data.
- We could apply a variety of different approaches to make these predictions (e.g. ARIMA, Prophet, deep learning LSTM methods).



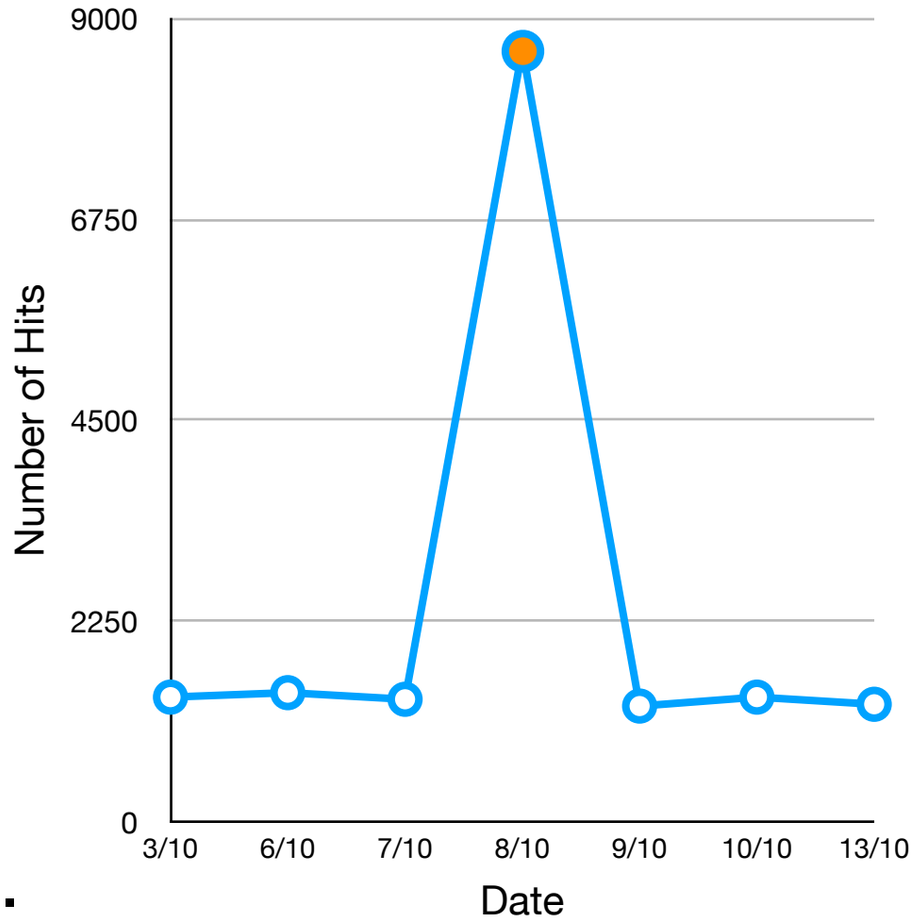
Time Series Forecasting

- **Application:** Predict the future effects of global warming on global temperature change to inform policymakers.



Outliers in Time Series Data

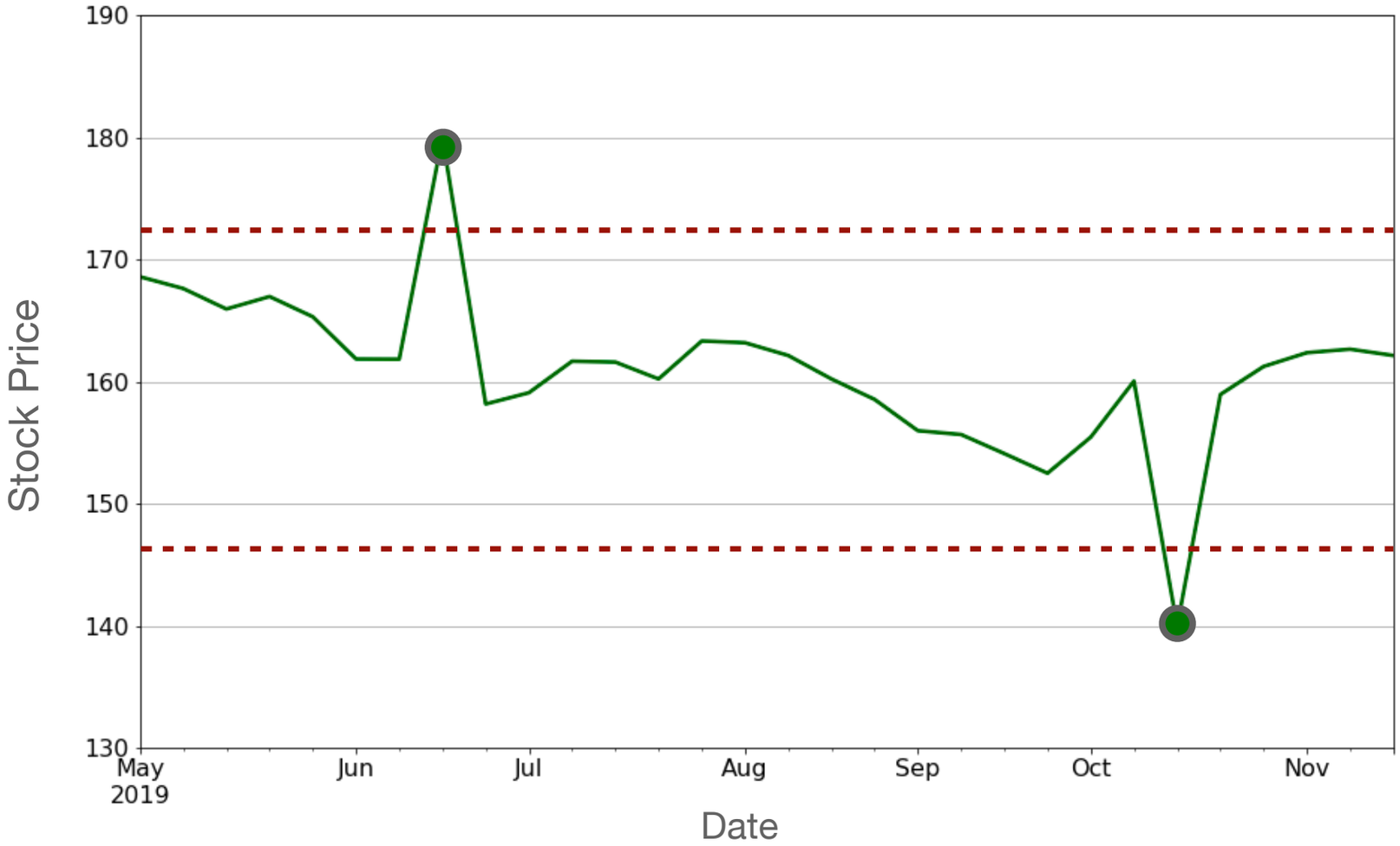
- An outlier in a time series is an observation that significantly differs from the other observations.
- Detecting outliers is often important:
  - Can affect the performance of time series forecasting techniques.
  - Might represent errors in the data, or might occur for specific underlying reasons which need to be investigated.



- **Examples:**
  - Web analytics: Spot unexpected growth or drop in website hits.
  - Financial data: Detection of price manipulation or fraudulent behaviour in banking and stock market exchange.
  - Aviation: Aircraft sensor monitoring to observe potential emerging malfunctions.

Outliers in Time Series Data

- Outliers might be visible as unexpected spikes or troughs in a time series plot. Many more complex detection methods exist.
- A common approach: observations above n standard deviations from the mean are considered outliers (e.g. $n=2$, $n=3$).



Calculate series mean and standard deviation

$$\sigma = 6.51$$

$$\mu = 160.93$$

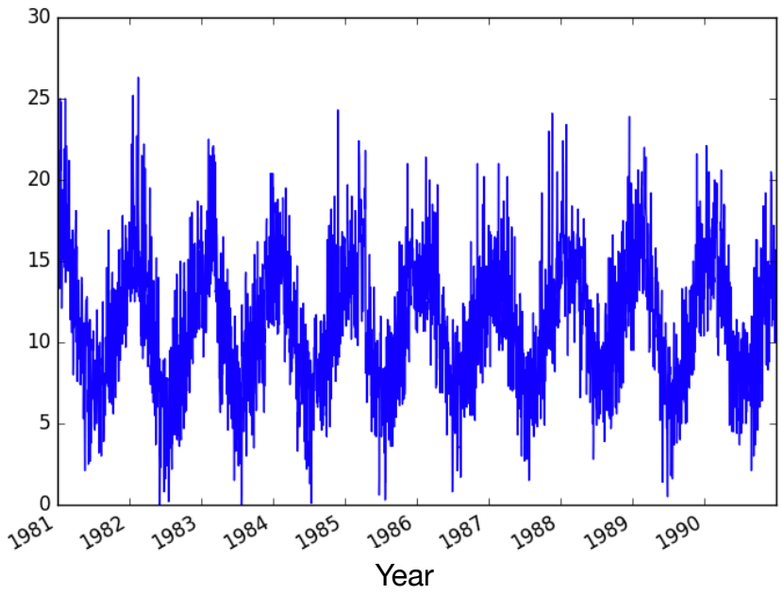
If $n=2$, outliers fall outside the range

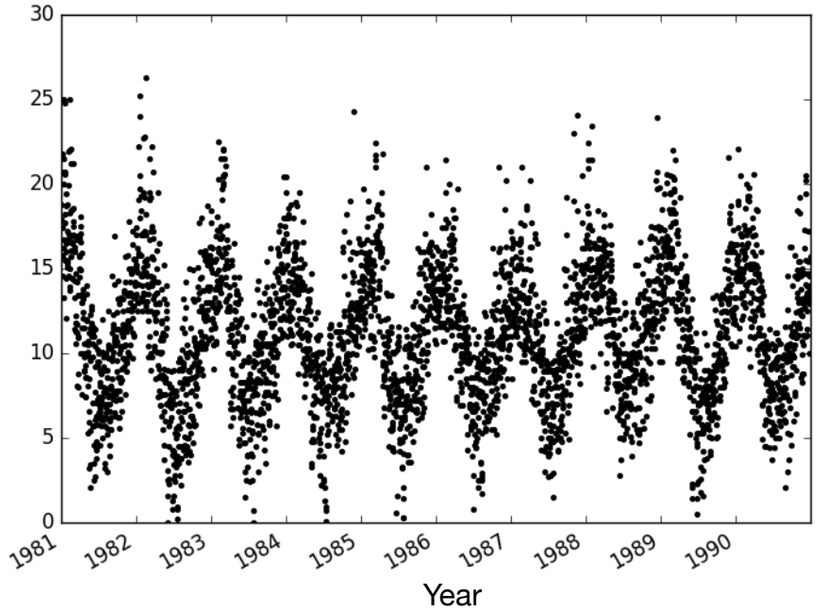
$$(\mu - 2\sigma, \mu + 2\sigma)$$

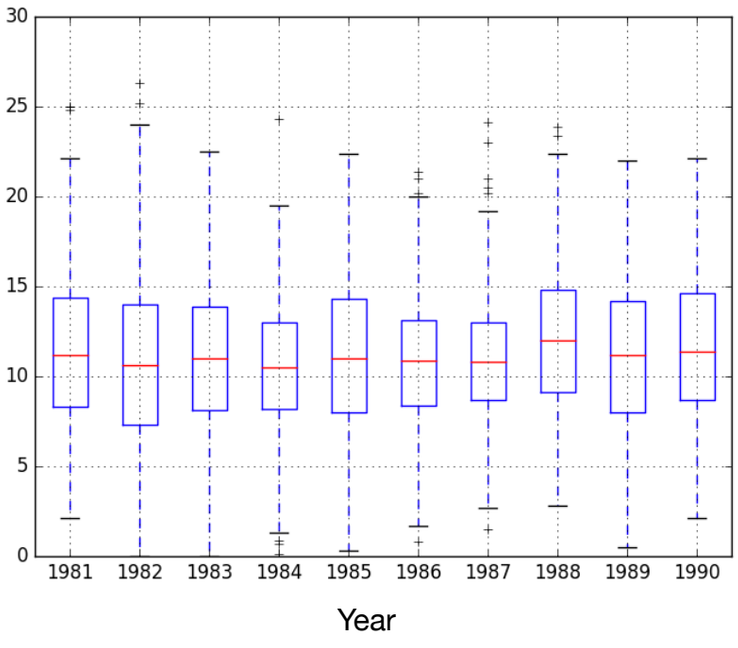
$$= (147.91, 173.95)$$

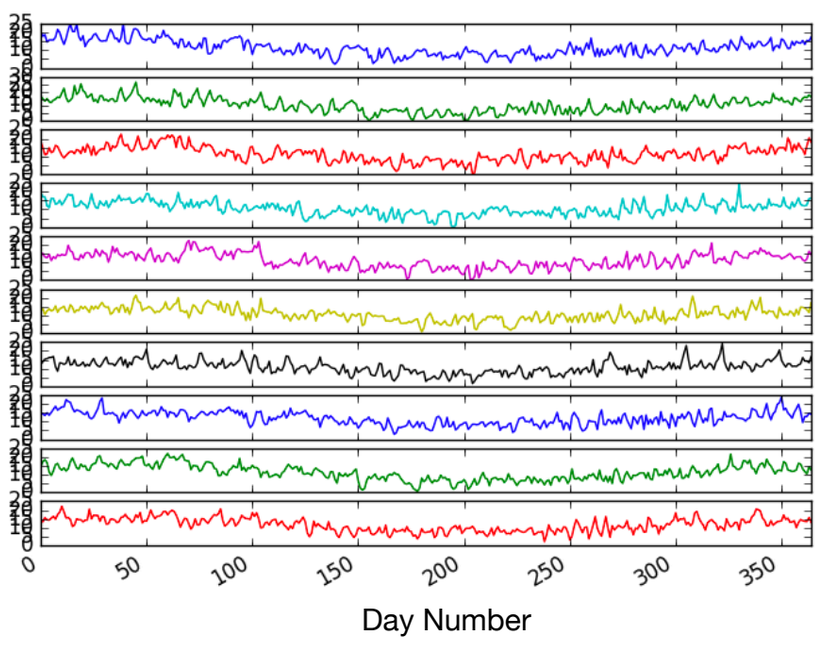
Time Series Visualisations

- While simple line plots are the most common way of visualising time series data, there are many variants and alternatives.









https://machinelearningmastery.com/time-series-data-visualization-with-python/