Week 6: Visualising Time Series

Visual Data Analytics

University of Sydney

THE UNIVERSITY OF
SYDNEY

Outline

- Lineplots
    - Time series features
    - Issues with y axis
    - Issues with x axis
- Python stuff
    - Dealing with dates
   - Plotting time series

Motivation

- Times series are common in business.
- Traditionally time series data in business were measured at low frequency (e.g. yearly, monthly, quarterly)
  - Inflation
  - Yearly sales
- Increasingly high-frequency data (daily, hourly, microsecond) are available.
  - Individual transactions data
- Business data are distinct in that they heavily influenced by calendar effects.

The line plot

- The line plot is the most common plot of a time series
- It shows a single variable on the vertical axis against time on the horizontal axis
- Using this plot we can see
    - Trend
    - Seasonal patterns
    - Calendar effects
    - Outliers
    - Volatility clustering

Yearly data

```
hle = sns.load_dataset('healthexp')
hleusa = hle[hle['Country']=='USA']
sns.lineplot(data = hleusa, x='Year', y='Life_Expectancy')
```
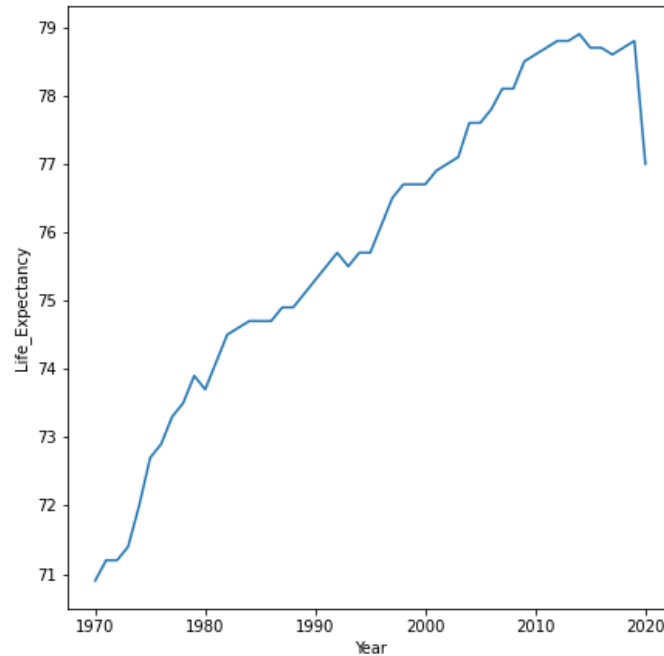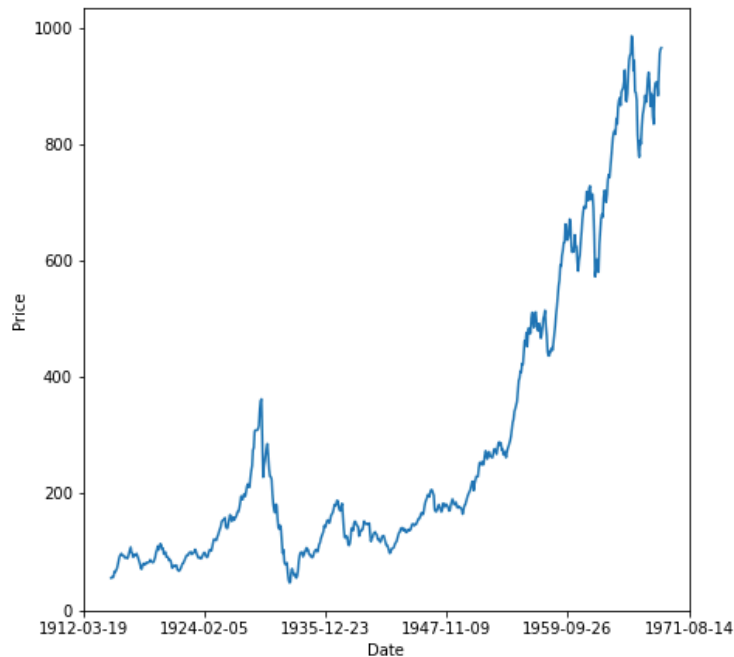
Trend

- This is an example of data with a trend.

- Life expectancy goes up over time.

- There is no seasonality (regular repeating pattern).

- There are no cycles (irregular repeating pattern).

Cycles

```python
import matplotlib.ticker as ticker
dj = sns.load_dataset('dowjones')
g = sns.lineplot(data=dj, x = 'Date', y = 'Price')
g.xaxis.set_major_locator(ticker.LinearLocator(6))
plt.show()
```

Trend and Cycle

- As well as trend, the series goes up and down.

- These are known as *cycles*.

- The cycles are irregular

  - Some are longer than others

  - The peaks and troughs are not always the same size and do not grow or shrink in a systematic way.
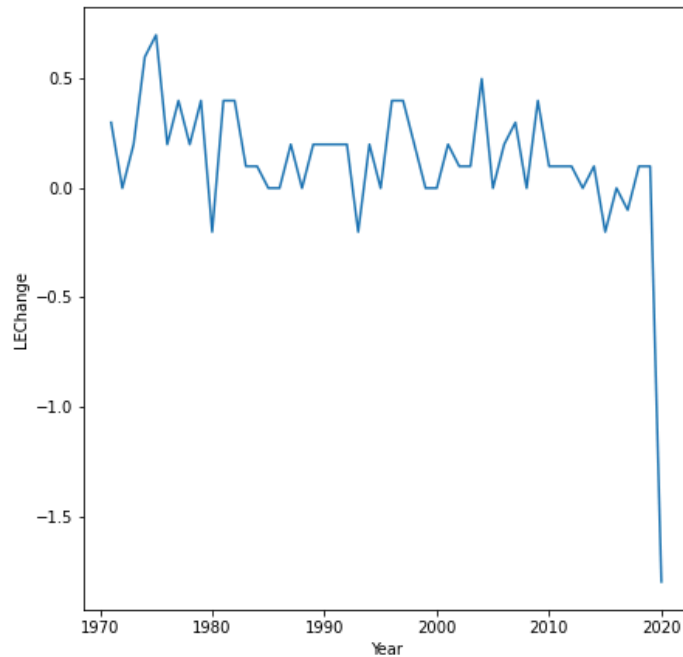
Seasonality
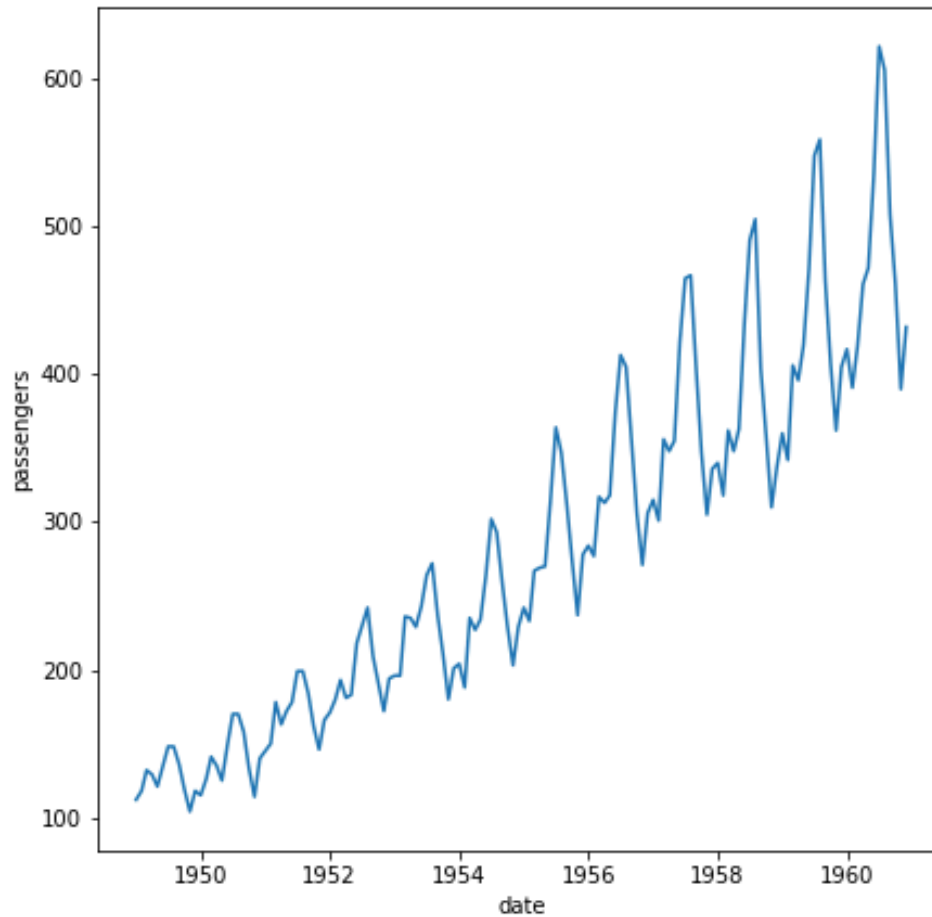
Seasonality

- The data roughly repeat every 12 periods.

- The pattern is amplified over time (which is in line with the increasing trend of the data).

- For higher freqeuncy data there may be multiple seasonalities (e.g, day of week and month of year).

Stocks v flows

- So-called *stocks* represent data measured at a single instant of time, *flows* represent changes to a stock over a period of time.
- The amount of money in my bank account is a stock, my spending during the week is a flow.
- This motivates looking at first differences, or percentage changes in data.
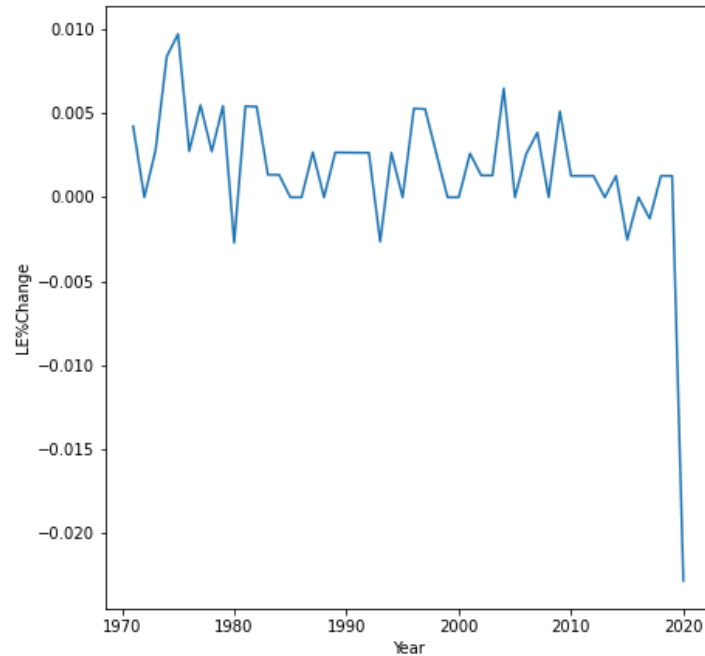
Plotting Change

```
hleusa['LEChange'] = hleusa["Life_Expectancy"].diff()
sns.lineplot(data = hleusa, x='Year', y='LEChange')
```

Percentage Change

```python
hleusa['LE%Change'] = hleusa["Life_Expectancy"].pct_change()
sns.lineplot(data = hleusa, x='Year', y='LE%Change')
```
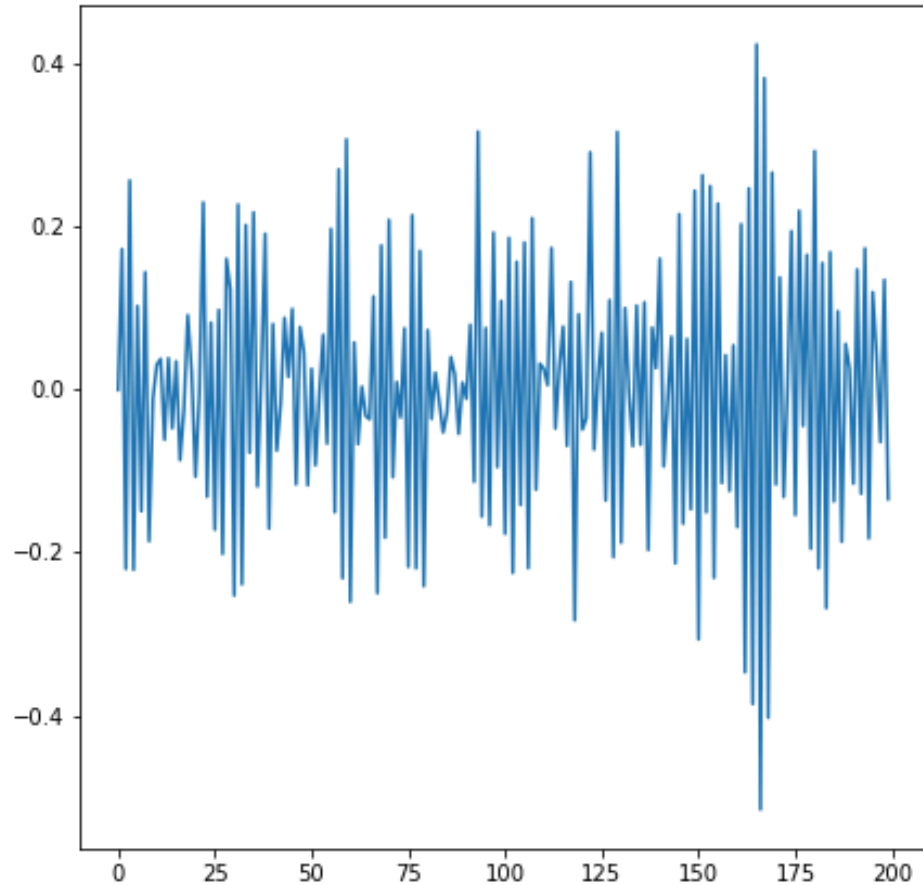
Autocorrelation

- When discussing time series, the concept of *autocorrelation* is important.
- This is the idea that a time series is correlated with its own past values.
- Line plots can indicate whether data are *positively*, correlated, *negatively* correlated or *not* correlated.
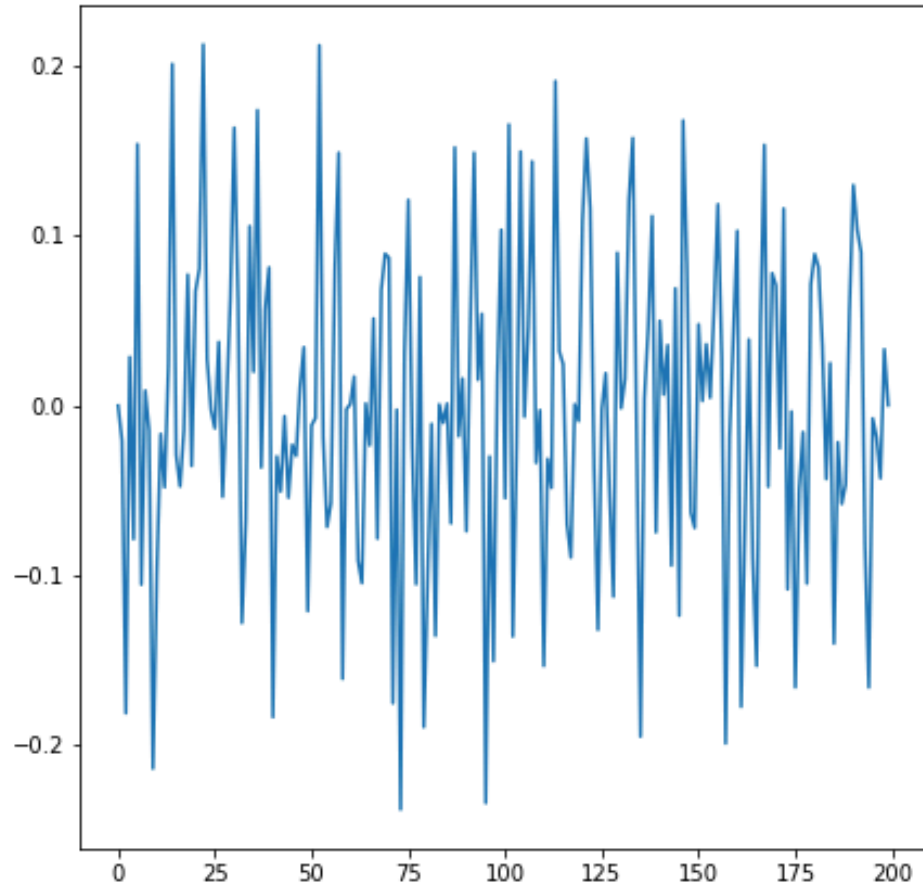- Here are some synthetic examples.

Postively correlated
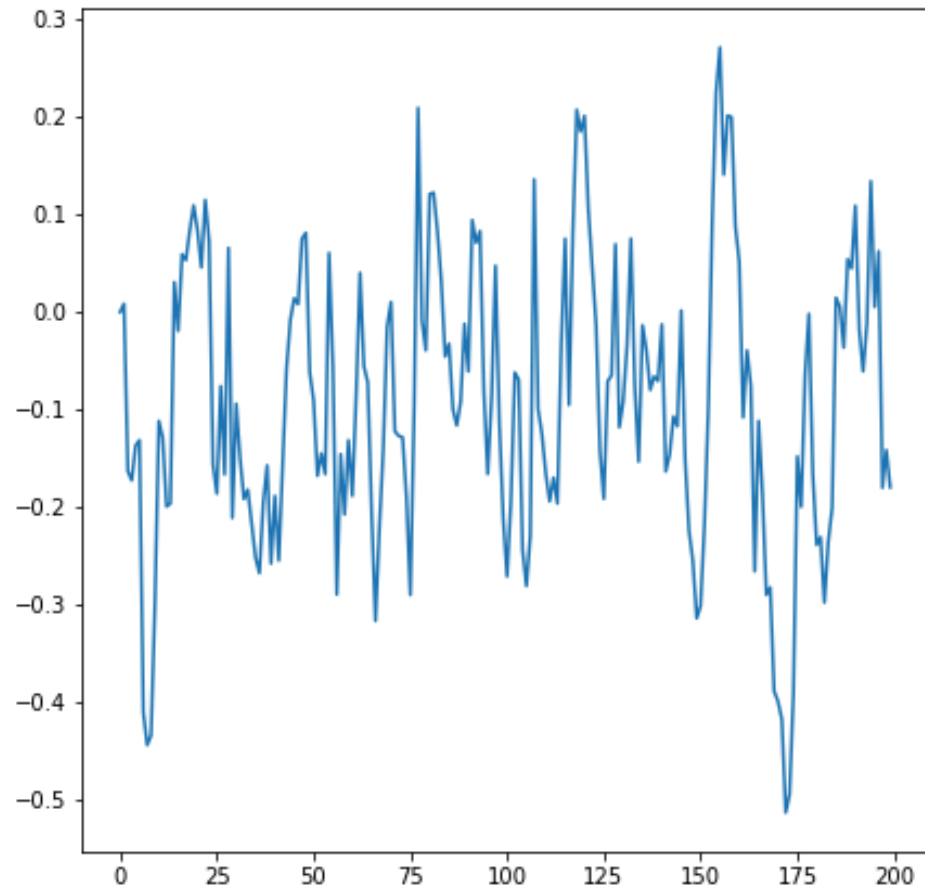
Negatively correlated

Not autocorrelated

Inerpreting autocorrelation

- For positively autocorrelated data, there will be *runs* where the series is above or below its mean.
- For negatively autocorrelated data, the series *oscillates* above and below the mean.
- For data with no autocorrelation, the series does not have these patterns.

Scatterplot

- A scatterplot of a variable against its first lag can aso indicate positive autocorrelation
- A scatterplot against other lags can show seasonality
- There are other ways to plot the autocorrelation function that are covered in other courses on time series,
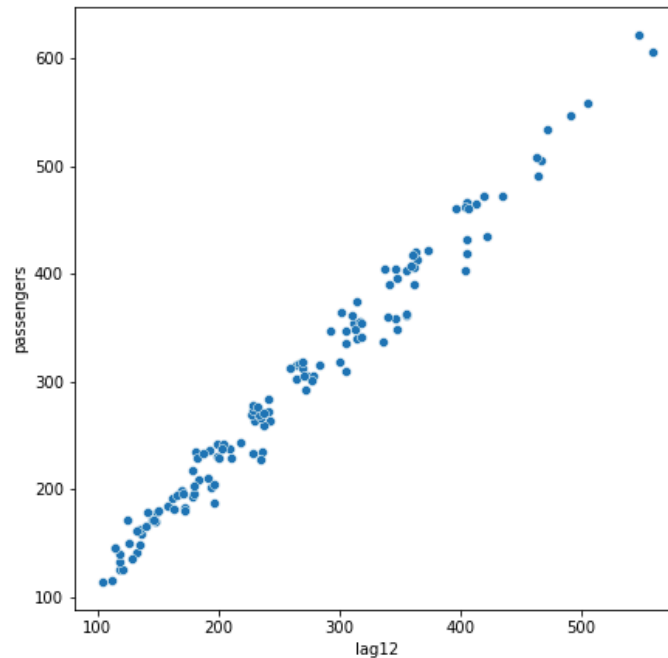
First lag

```python
flights = sns.load_dataset('flights')
flights['firstlag']=flights['passengers'].shift()
sns.scatterplot(data=flights, x = 'firstlag', y = 'passengers')
```
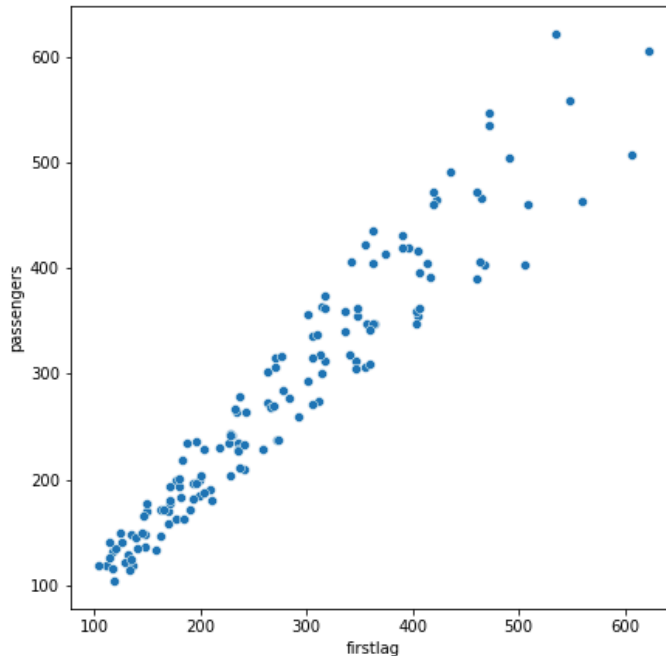
Lag twelve

```python
flights['lag12']=flights['passengers'].shift(12)
sns.scatterplot(data=flights, x = 'lag12', y = 'passengers')
```

Volatility clustering

- Sometimes there is no correlation in the mean, but there is correlation in the variance
    - Volatile periods more likely to follow volatile periods
    - Calm periods more likely to follow calm periods
- This can be seen using a line plot
- It is common with financial returns data

Volatility clustering

```
dj['Returns']=dj['Price'].pct_change()
g = sns.lineplot(data=dj, x = 'Date', y = 'Returns')
g.xaxis.set_major_locator(ticker.LinearLocator(6))
plt.show()
```
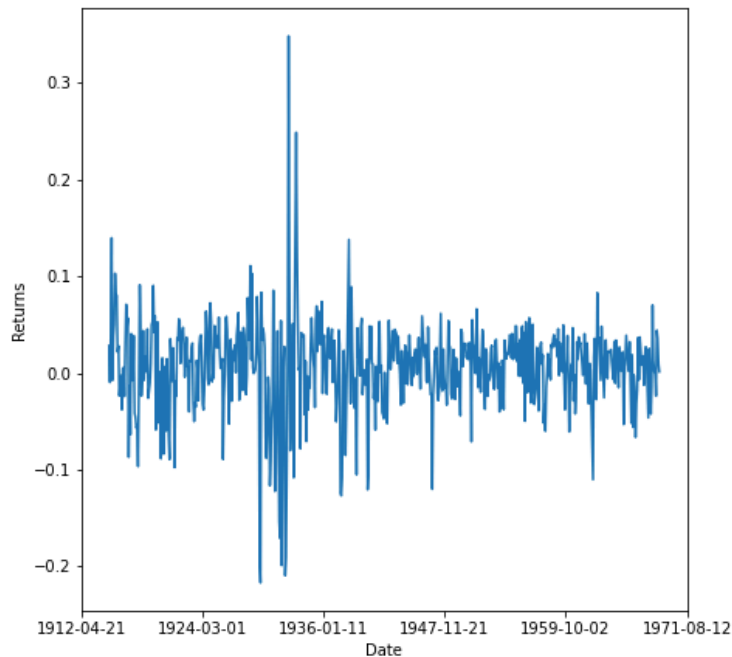
Issues with axes

Should zero be on y axis?

- Using a wide or narrow y-axis can be used to exaggerate changes in a line plot
- It is often said that zero MUST be included on the y axis
- This depends on *context*

Climate graph

Figure 2: Global Mean Temperature (1850–2006)
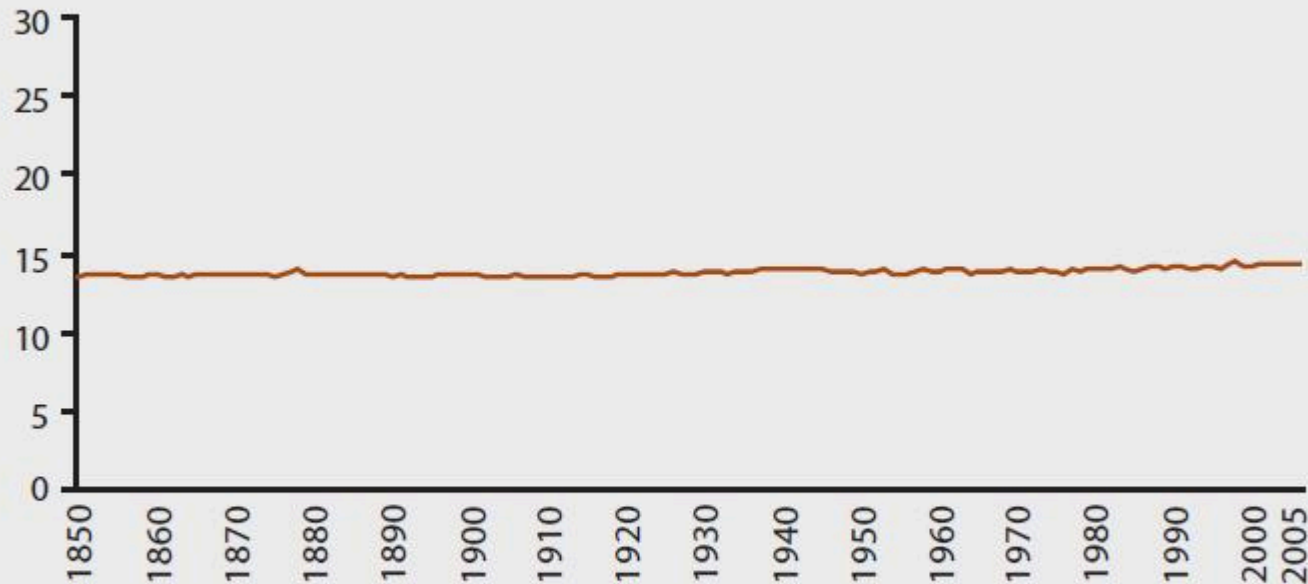
Source: Derived from the US Bureau of Meteorology Data
http://www.bom.gov.au/web01/ncc/www/cli_chg/timeseries/global_t/0112/global/latest.txt

Climate graph

Outside the range of the past 500 million years

Outside the range of the past 5 million years

Outside the range of the Holocene

Outside the range of the past 500 million years

27

Line v bar

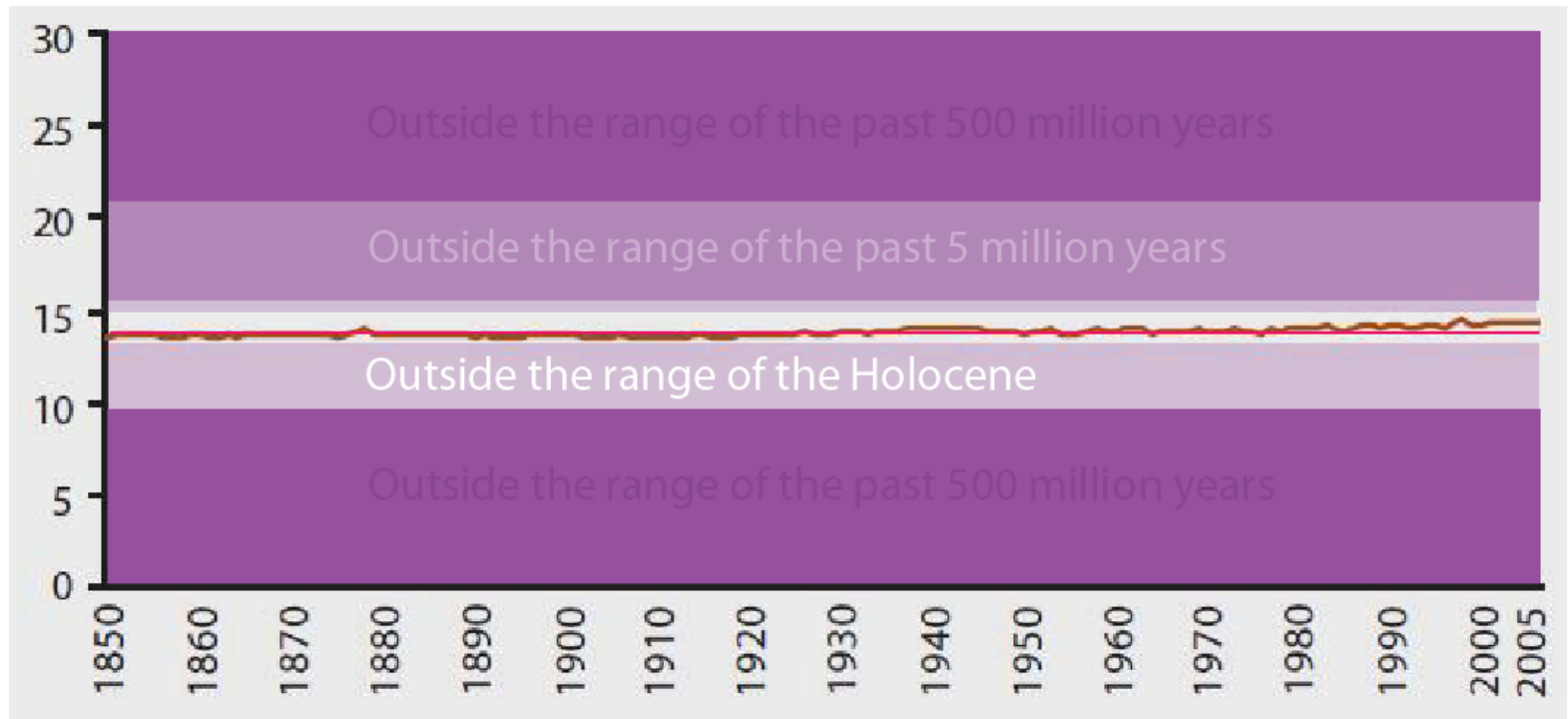- For a line plot, to decide whether to include zero, think about whether 0 is a sensible value for the y-variable to take.
- Before the Holocene, you could walk from here to Tasmania and North America was under 4km of ice.
- Note that for bar plots, it is more natural to interpret length of the bar rather than position on the y axis.
- For bar plots always include zero

Better as line plot

Americans ditch cable in droves
Cable TV subscribers, in millions

104 — 2011
103.2 — 2012
102.7 — 2013
101.7 — 2014
100.8 — 2015

Source: IBISWorld

Likely to be misinterpreted.

The x-axis

- Similarly the context of the x-axis is important.
- To understand whether a change is big or small, it is important to look at a suitable range of data.
- The following is an example with stock prices of the Commonwealth Bank of Australia (CBA).
- The following show the price of CBA shares over a five day period and a five year period.

A big change?

31

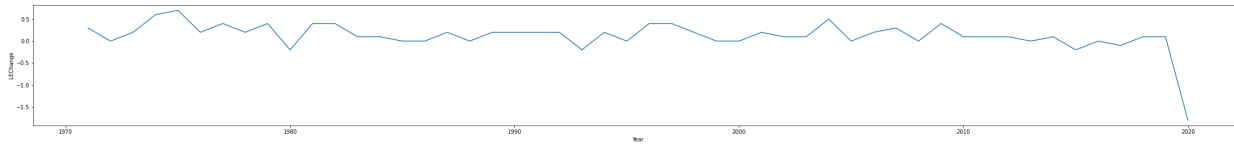# Both axes

- Another way to manipulate interpretations is through the *aspect ratio*.
- The aspect ratio is the ratio of units on the y axis relative to units on the x axis.
- By resizing a plot, features of the time series may be exaggerated (either intentionally or unintentionally).

Wide

```
fig, ax = plt.subplots(figsize = (40,4) )
sns.lineplot(data = hleusa, x='Year', y='LEChange')
```

Long

```
fig, ax = plt.subplots(figsize = (3,5) )
sns.lineplot(data = hleusa, x='Year', y='LEChange')
```
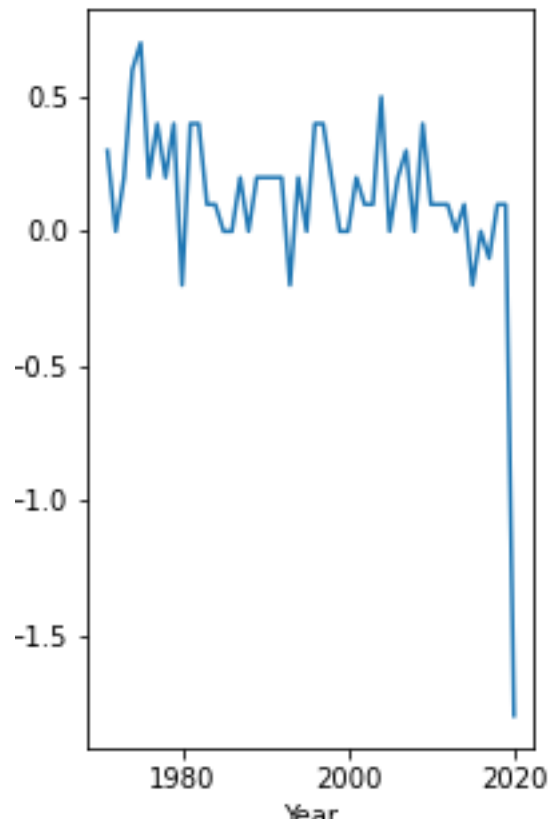
Banking to 45

- As a rough guide, consider lines making up a line plot.
- The average angle of these should be about 45 degrees
- Good software packages will do this by default.
- If you resize an image things may change.
- Always look at the axes of a line plot!

Dual y axes

Tech **Chart of the Day**

Online Hotel Revenue vs Number Of Travel Agents

Online hotel revenue
$ billion

Number of travel agents in the United States
Thousand

BUSINESS INSIDER

Source: McKinsey, Phocuswright, BLS
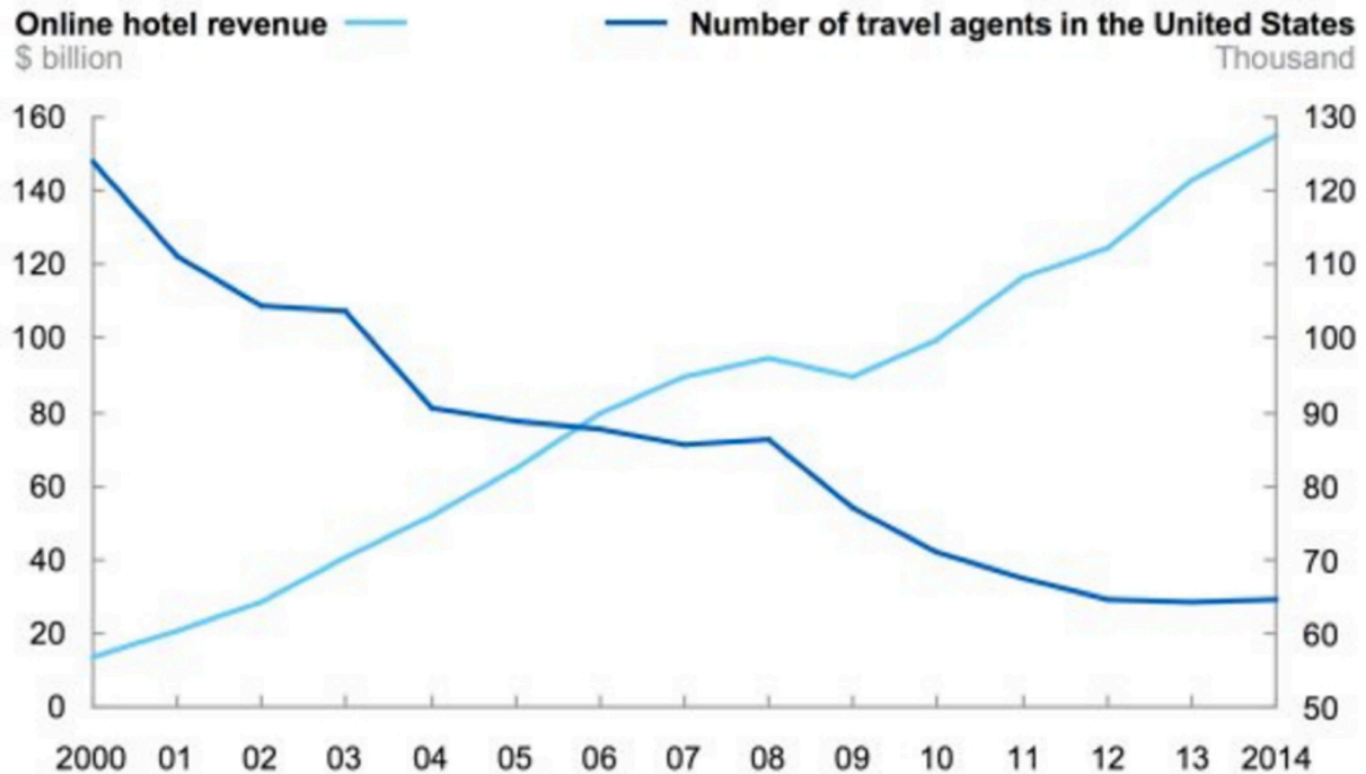
Problems

- Natural to look at and interpret 'crossing points'

- Crossing point nearly always meaningless;

  - Arbitrarily defined by changing y axis,

  - Could be different if different units are used.

- Put multiple lines on a lineplot ONLY when all variables measured in same units.

Dealing with dates

Date type in Python

The `datetime` module in Python provides a data type specifically for dealing with dates and times.

```python
import datetime
x=datetime.datetime.now()
y=datetime.datetime(2008, 3, 16)
print(x)
```

```
## 2024-08-25 15:57:28.093120
```

```python
print(y)
```

```
## 2008-03-16 00:00:00
```

Intervals

A `timedelta` is the difference between two dates

```
print(x-y)
```

```
## 6006 days, 15:57:28.093120
```

An interval can be added to a date

```
print(y)
```

```
## 2008-03-16 00:00:00
```

```
print(y+datetime.timedelta(days=10))
```

```
## 2008-03-26 00:00:00
```

Datasets

- We will look at two datasets
  - The taxis data set from seaborn
  - A dataset from the Australian Energy Market Operator of electricity demand in thirty minute intervals for five regions of Australia on the 18th-24th September 2022.

Reading in data

```python
import pandas as pd
elec = pd.read_csv('../data/electricity.csv')
elec
```

```
##       REGIONID    INTERVAL_DATETIME  OPERATIONAL_DEMAND
## 0        NSW1  2022/09/18 00:00:00                7568
## 1        QLD1  2022/09/18 00:00:00                5575
## 2         SA1  2022/09/18 00:00:00                1578
## 3        TAS1  2022/09/18 00:00:00                1071
## 4        VIC1  2022/09/18 00:00:00                5349
## ...       ...                  ...                 ...
## 1675     NSW1  2022/09/24 23:30:00                7561
## 1676     QLD1  2022/09/24 23:30:00                5694
## 1677      SA1  2022/09/24 23:30:00                1428
## 1678     TAS1  2022/09/24 23:30:00                1184
## 1679     VIC1  2022/09/24 23:30:00                5137
##
## [1680 rows x 3 columns]
```

Converting to date

Dates may not be read in as dates

```
elec.dtypes
```

```
## REGIONID              object
## INTERVAL_DATETIME     object
## OPERATIONAL_DEMAND     int64
## dtype: object
```

Can convert using `to_datetime`

```
elec['INTERVAL_DATETIME'] = pd.to_datetime(elec['INTERVAL_DATETIME'])
elec.dtypes
```

```
## REGIONID                    object
## INTERVAL_DATETIME     datetime64[ns]
## OPERATIONAL_DEMAND           int64
## dtype: object
```

Conversion

Convert dates to strings with formats here

```
datetime.datetime.strftime(y,'%a %d-%B-%Y')
```

'Sun 16-March-2008'

```
datetime.datetime.strftime(y,'%B %d, %y')
```

'March 16, 08'

And convert back
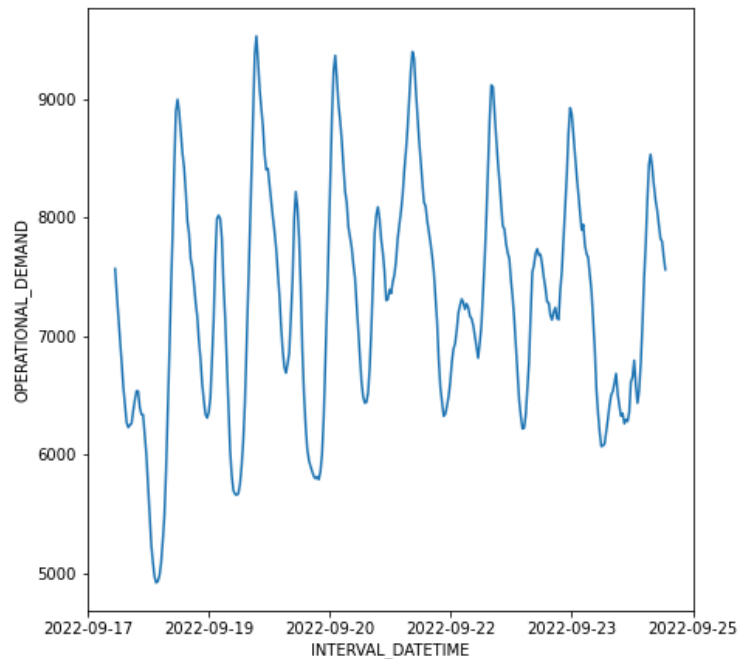
```
print(datetime.datetime.strptime('December 10, 22','%B %d, %y'))
```

2022-12-10 00:00:00

More Python plotting

NSW electricity demand

```python
fig, ax = plt.subplots()
g = sns.lineplot(data = elec[elec['REGIONID']=='NSW1'],x='INTERVAL_DATE
g.xaxis.set_major_locator(ticker.LinearLocator(6))
plt.show()
```
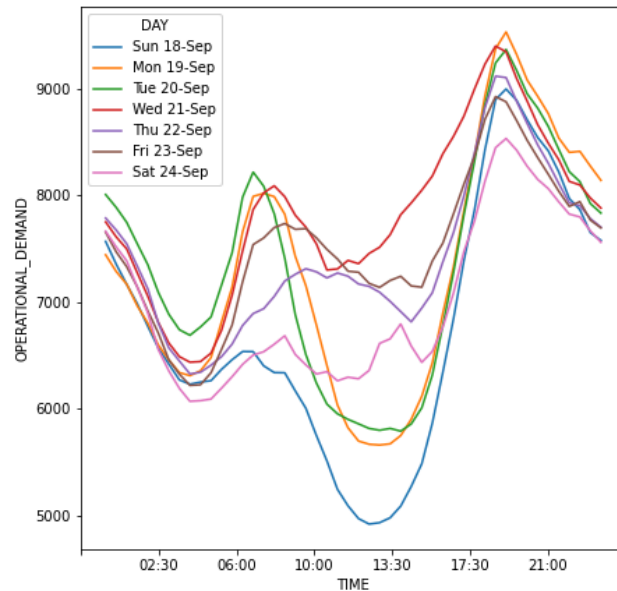
What do we see?

- Some seasonality
  - Reflects patterns of usage throughout the day
- No upwards or downwards trend
  - Make sense since data is only measured over a week
- Can we visualise differently?

Seasonal plot

```
elec['TIME']=elec['INTERVAL_DATETIME'].dt.strftime('%H:%M')
elec['DAY']=elec['INTERVAL_DATETIME'].dt.strftime('%a %d-%b')
g = sns.lineplot(data = elec[elec['REGIONID']=='NSW1'],x='TIME', y = 'O
g.xaxis.set_major_locator(ticker.LinearLocator(8))
plt.show()
```

What do we see?

- Two peaks
  - Small one in morning
  - Larger one in evening
- Why no early peak on Sunday?
- Why is there a small afternoon peak on the Saturday?

An example with taxi data

- The taxi data contains event data
- We can construct a daily series of number of taxi trips
- This requires some data preparation and conversion between datetimes and dates

Data preparation

```
taxisdat = sns.load_dataset('taxis')
taxisdat['timestamp'] = pd.to_datetime(taxisdat['pickup'])
taxisdat['date'] = taxisdat['timestamp'].dt.strftime('%Y/%m/%d')
taxists = taxisdat.groupby('date').size().reset_index(name='trips')
taxists['date'] = pd.to_datetime(taxists['date'])
taxists
```

```
##           date  trips
## 0   2019-02-28      1
## 1   2019-03-01    241
## 2   2019-03-02    198
## 3   2019-03-03    169
## 4   2019-03-04    171
## 5   2019-03-05    228
## 6   2019-03-06    257
## 7   2019-03-07    218
## 8   2019-03-08    235
## 9   2019-03-09    204
## 10  2019-03-10    185
## 11  2019-03-11    209
```

Lineplot

```
g = sns.lineplot(data = taxists, x='date', y='trips')
g.xaxis.set_major_locator(ticker.LinearLocator(6))
plt.show()
```
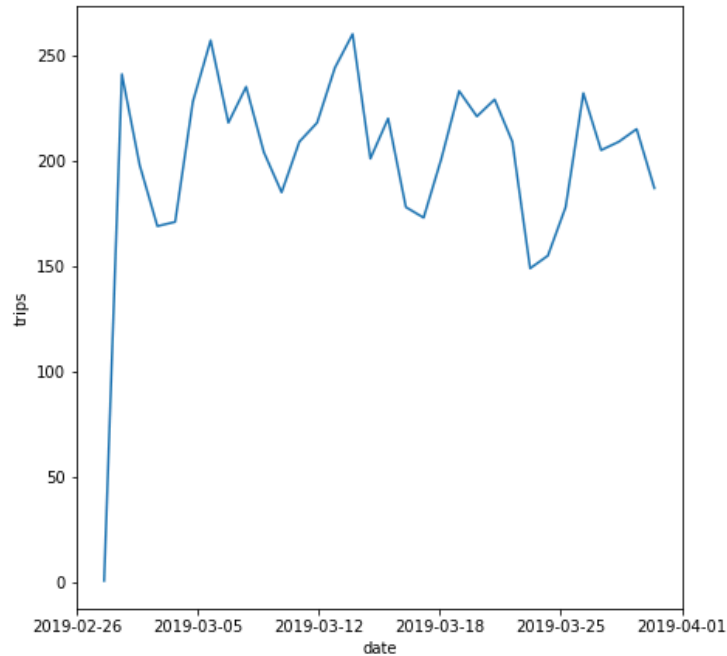
What do you see?

- The very first observation is an outlier

- There is also a weekly pattern

- Don't forget that time series data is like all other data. We can use plots *other* than line plots!

An example

```
taxisdat['datehour'] = pd.to_datetime(taxisdat['pickup']).dt.strftime('
taxish = taxisdat.groupby('datehour').agg({'fare':'mean','distance':'me
taxish['hour'] = taxish.index.str[-2:].astype(int)
taxish
```

```
##                    fare   distance   hour
## datehour
## 2019/02/28 23    5.000000   0.900000     23
## 2019/03/01 00   12.625000   3.286250      0
## 2019/03/01 01    5.000000   1.000000      1
## 2019/03/01 02    9.833333   2.480000      2
## 2019/03/01 04    5.833333   1.250000      4
## ...                 ...        ...       ...
## 2019/03/31 19    8.900000   1.691000     19
## 2019/03/31 20   11.250000   2.758333     20
## 2019/03/31 21   17.000000   4.748750     21
## 2019/03/31 22   10.625000   2.422500     22
## 2019/03/31 23   24.250000   7.640000     23
##
```
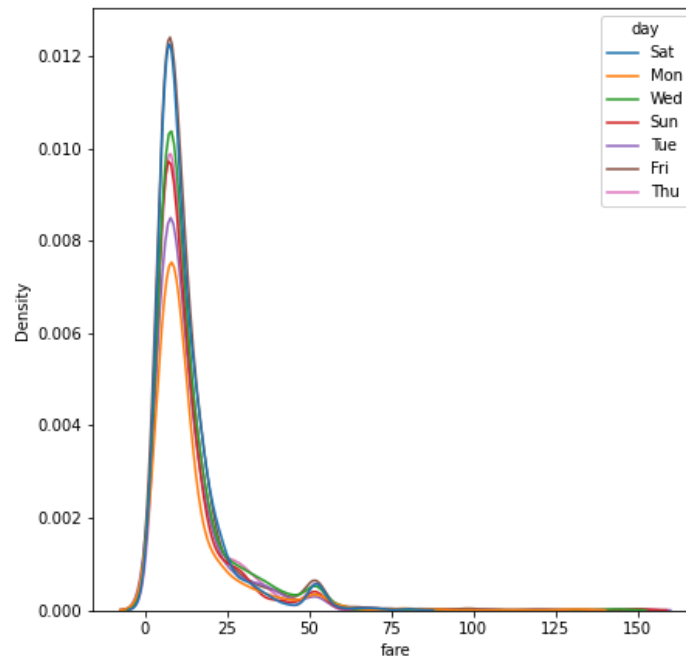
Plot

```
sns.scatterplot(data = taxish, x = 'distance', y = 'fare', hue = 'hour'
```
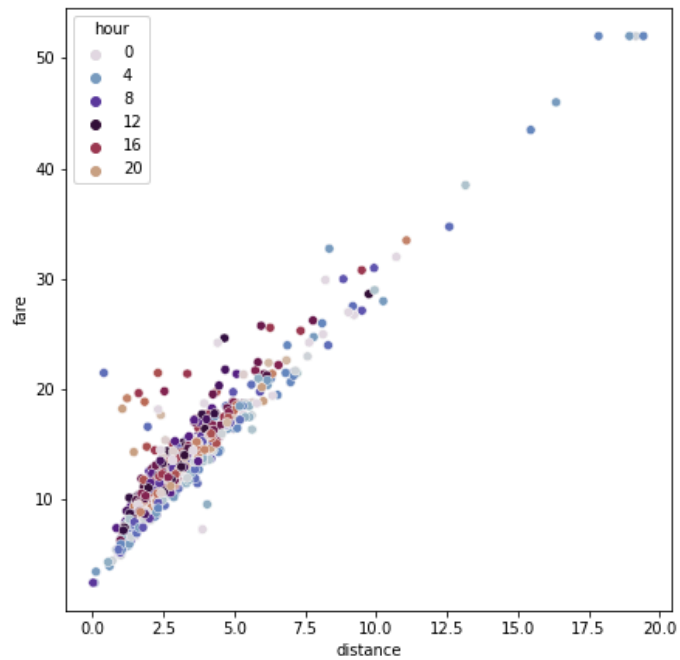
Another example

```
taxisdat['day'] = pd.to_datetime(taxisdat['pickup']).dt.strftime('%a')
sns.kdeplot(data = taxisdat, x='fare', hue='day')
```

Wrap-up

Conclusions

- The most common plot for time series is the lineplot
    - Can understand time series patterns such as trends, cycles and seasonality.
    - Can find outliers
    - Can understand autocorrelation properties.
- Important to manipulate dates
    - Can get more creative with visualisations.

Questions