Time Series Analysis From the Ground Up

Bruno Gonçalves

www.data4sci.com/newsletter

https://github.com/DataForScience/Timeseries_long

astropy

BIOPYTHON

DIPY

NIPY

SunPy

SymPy

NetworkX

SM StatsModels
Statistics in Python

scikit-image
image processing in python

scikit learn

matplotlib

PyMC

Bokeh

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

xarray

NumPy

jupyter

SciPy

IP[y]:
IPython

Cython

DASK

python™

Numba

astropy

BIOPYTHON

DIPY

NIPY

SunPy

SymPy

NetworkX

SM StatsModels
Statistics in Python

scikit-image
image processing in python

scikit learn

matplotlib

PyMC

Bokeh

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

xarray

SciPy

NumPy

jupyter

IP[y]:
IPython

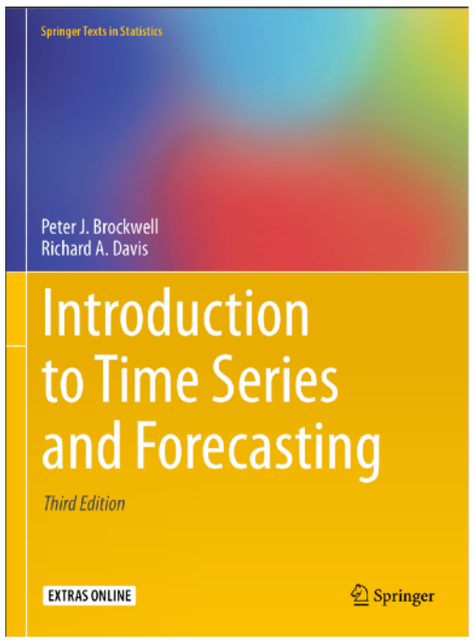
Cython

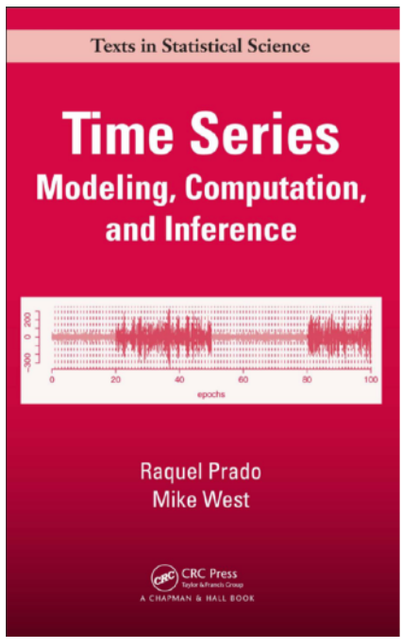
DASK

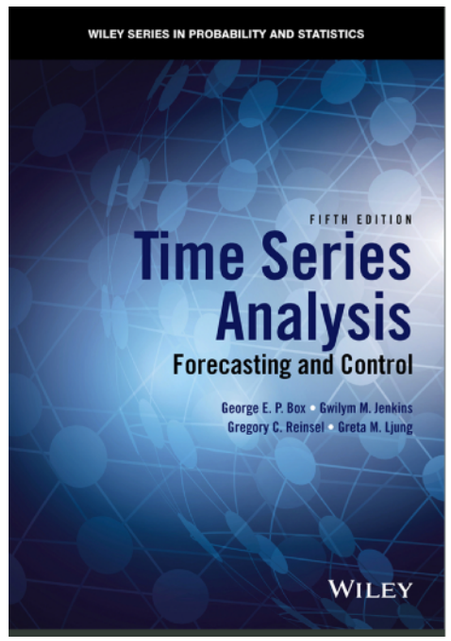
python

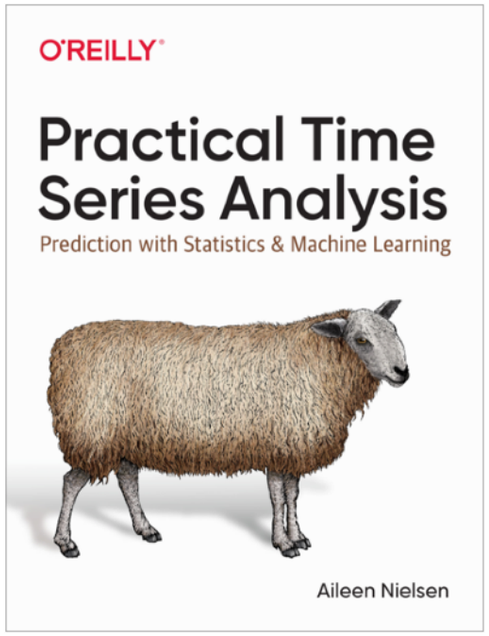
Numba

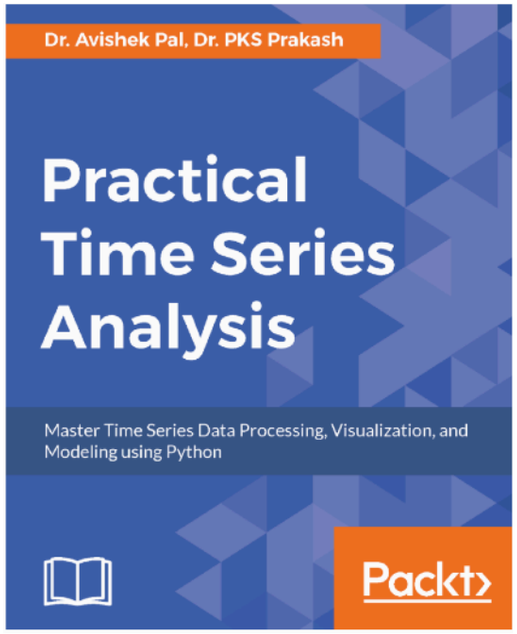
References


Springer Texts in Statistics
Peter J. Brockwell
Richard A. Davis
Introduction to Time Series and Forecasting
Third Edition
EXTRAS ONLINE
Springer


Texts in Statistical Science
Time Series
Modeling, Computation, and Inference
Raquel Prado
Mike West
CRC Press
A CHAPMAN & HALL BOOK


WILEY SERIES IN PROBABILITY AND STATISTICS
FIFTH EDITION
Time Series Analysis
Forecasting and Control
George E. P. Box • Gwilym M. Jenkins
Gregory C. Reinsel • Greta M. Ljung
WILEY

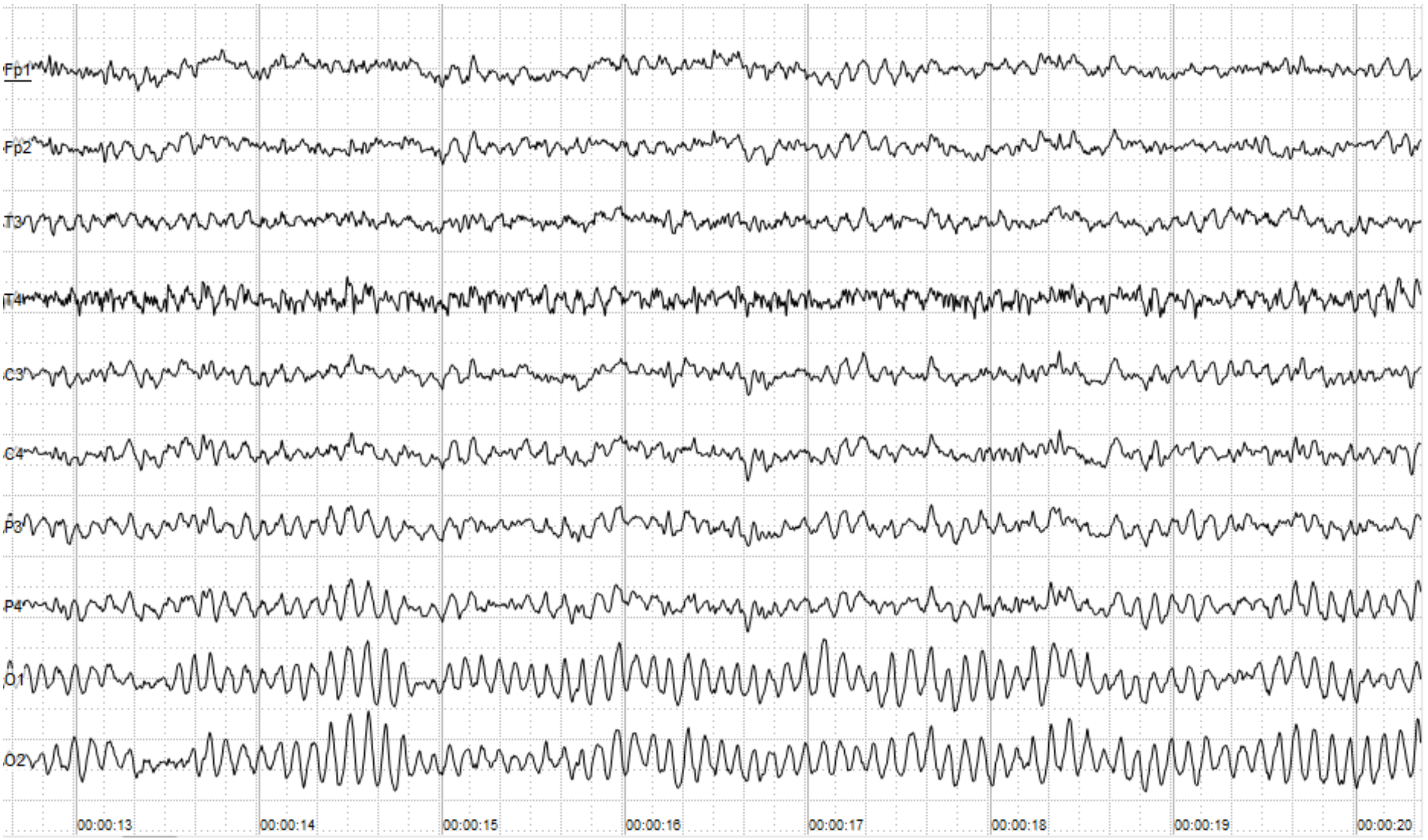

O'REILLY
Practical Time Series Analysis
Prediction with Statistics & Machine Learning
Aileen Nielsen


Dr. Avishek Pal, Dr. PKS Prakash
Practical Time Series Analysis
Master Time Series Data Processing, Visualization, and Modeling using Python
Packt›

Time series examples

Medicine - EEG

https://en.wikipedia.org/wiki/Electroencephalography

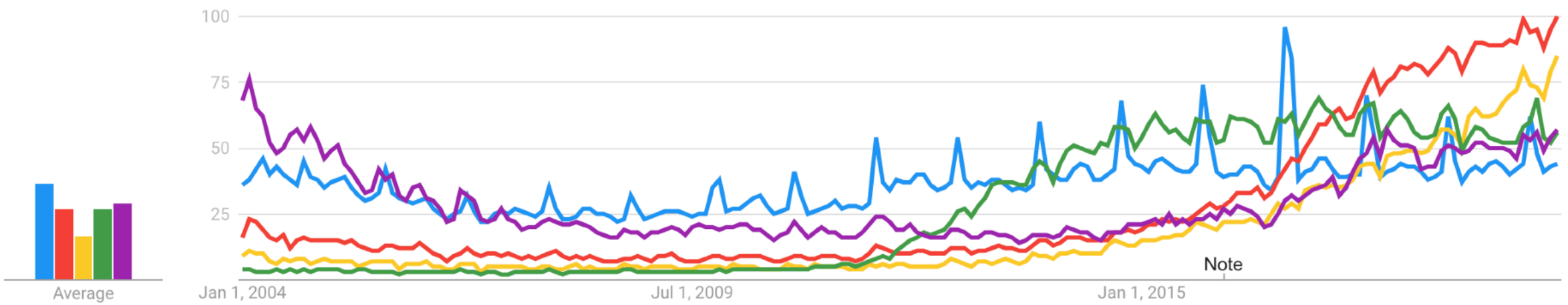


Public Interest - Search Trends

Time Series	Machine Learning	Data Science	Big Data	Artificial Intelligence
Search term	Search term	Search term	Search term	Search term

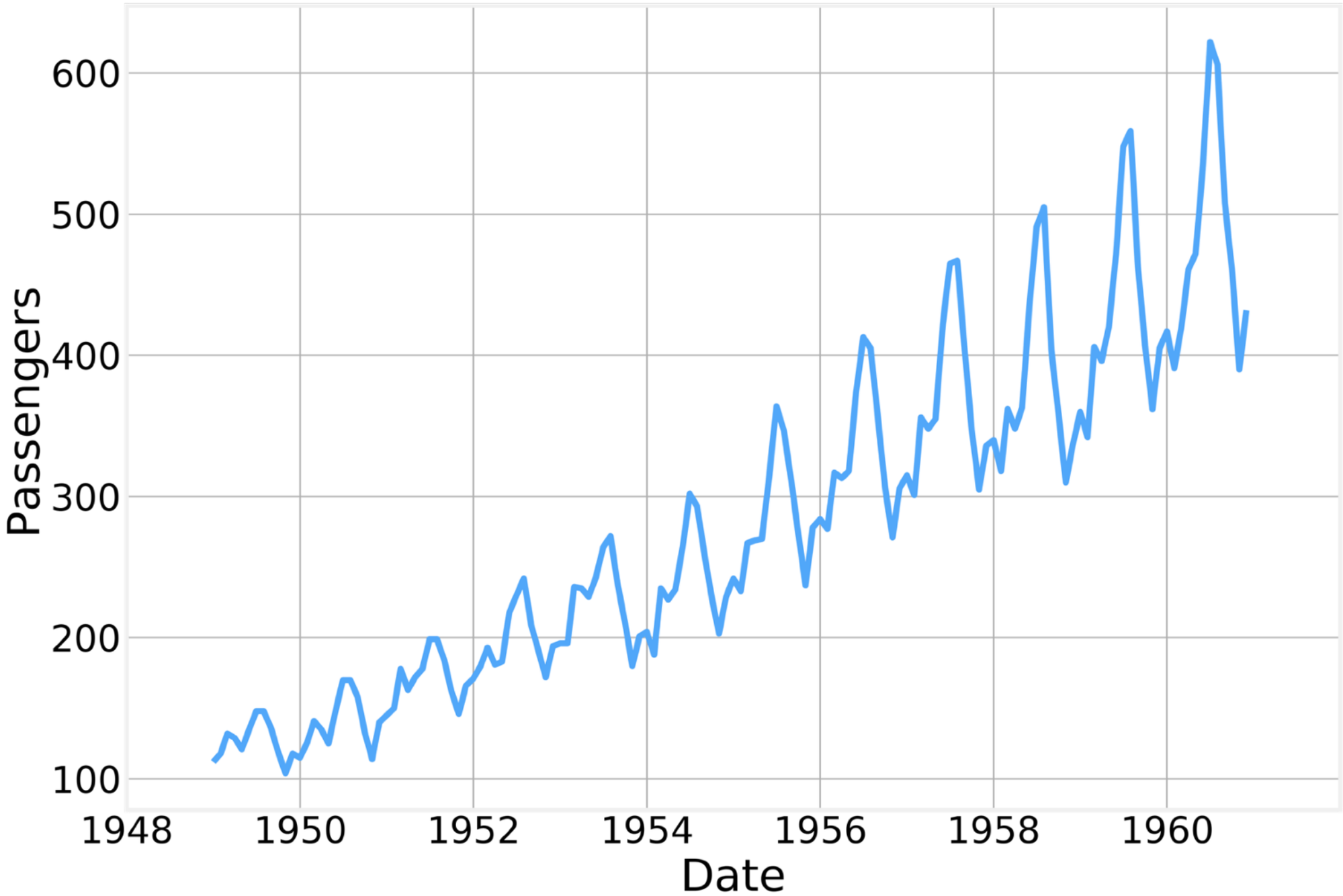
Worldwide ▾ 2004 - present ▾ All categories ▾ Web Search ▾

Interest over time



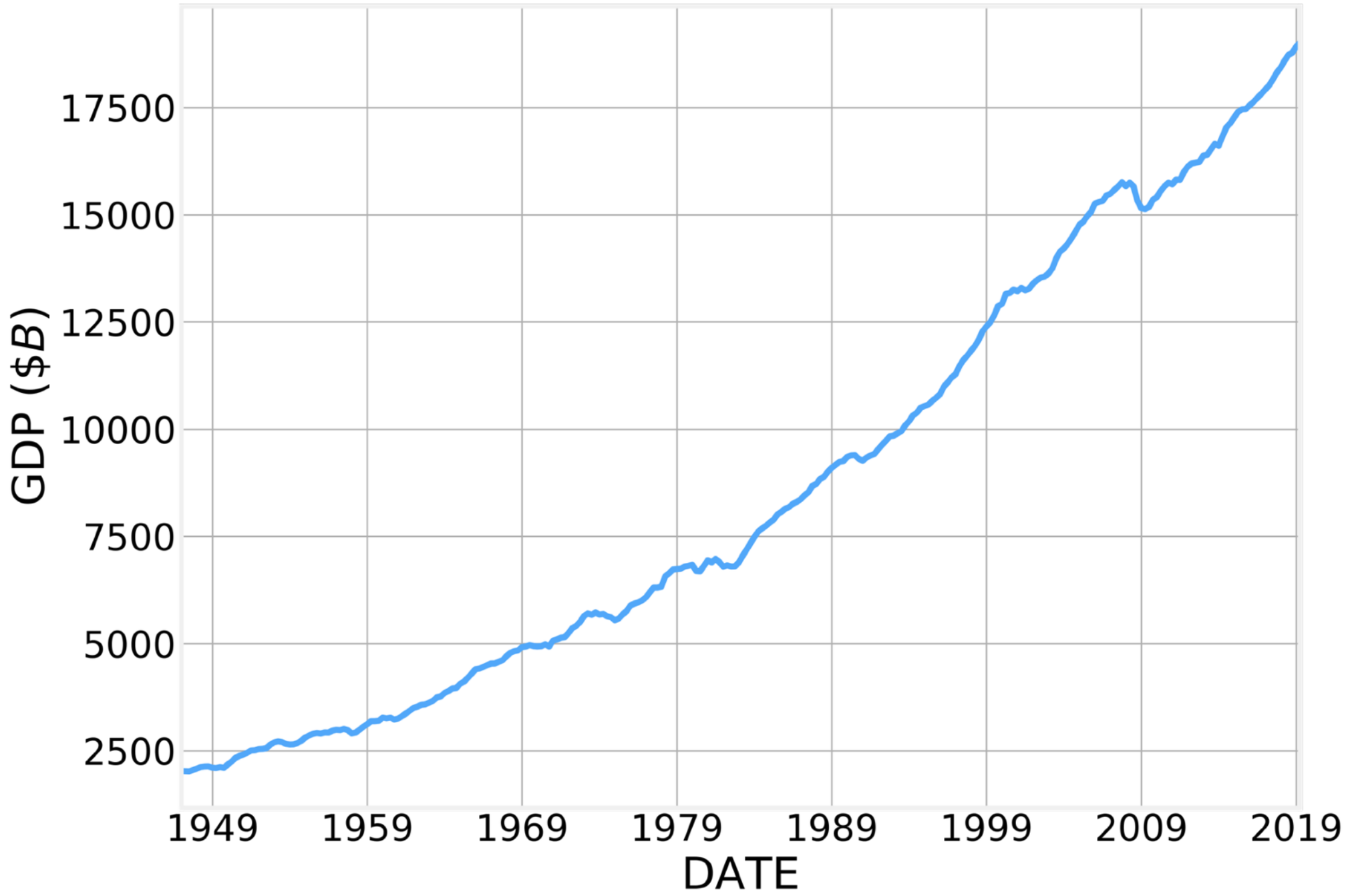
Logistics - Airline Passengers

https://www.kaggle.com/chirag19/air-passengers



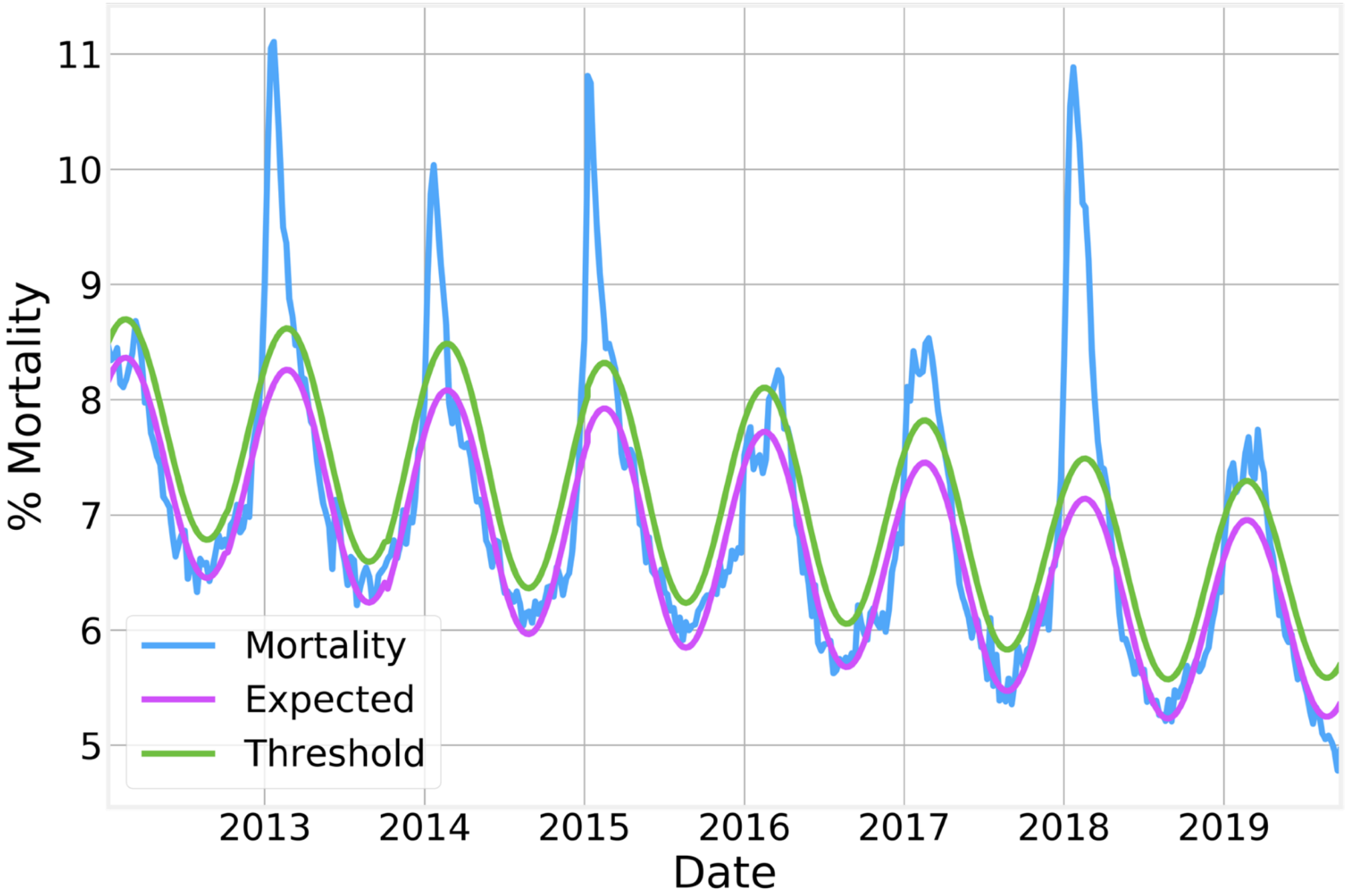
Economics - GDP

https://fred.stlouisfed.org/series/GDPC1



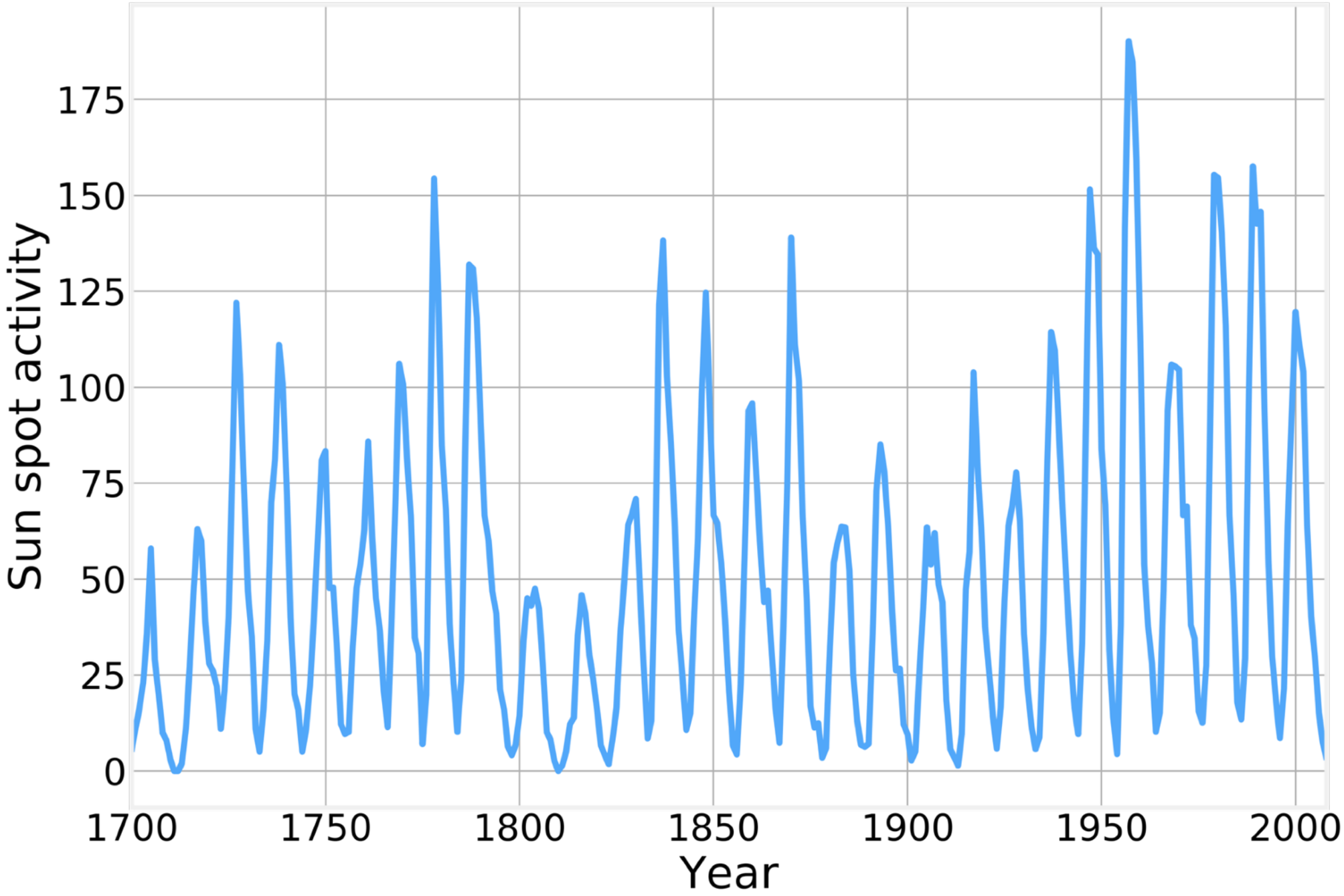
Epidemiology - Influenza

www.cdc.gov/flu/weekly/



Astronomy - Sunspot activity

http://www.sidc.be/silso/datafiles



Stock Market - DJIA

https://fred.stlouisfed.org/series/DJIA

Pandas Review

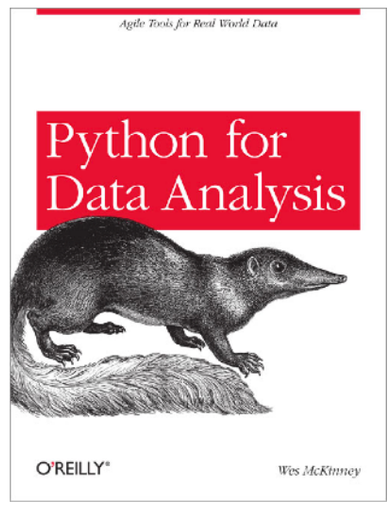
pandas

pandas.pydata.org

- Created by Wes McKinney in 2008 while working for AQR Capital Management, currently working at Two Sigma and Ursa Labs
- Pandas is specifically designed to handle time series and data frames
- High performance, flexible tool for quantitative analysis on financial data
- The functionality is built on top numpy and matplotlib (see next lecture)

- The fundamental data structures are Series (1-dimensional) and DataFrame (2-dimensional).
- Each column of a DataFrame can have a different dtype but all the elements in a column (or Series) must be of the same dtype
- Conventionally imported as

```
import pandas as pd
```



Importing and Exporting Data

- pandas has powerful methods to read and write data from multiple sources
  - pd.read_csv() - Read a comma-separated values file
    - sep=' ' - Define the separator to use. ',' is the default
    - header=0 - Row number to use as column names
  - pd.read_excel() - Read an Excel file
    - sheet_name - The sheet name to load
  - pd.read_html() - Read tabular data from a URL (or local html file)
  - pd.read_pickle() - Read a Pickle file
- Each of these functions accepts a large number of options and parameters controlling its behavior. Use help(<function name>) to explore further.
- Each read_*() function has a complementary to_*() function to write out a DataFrame to disk. The to_*() functions are members of the DataFrame object

Time series

- pandas was originally developed to handle financial data
- Temporal sequences (time series) are a common type of data encountered in financial applications, so, naturally, pandas has good support for the most common time series operations.
- pd.to_datetime - converts a Series or value into a date time timestamp.
    - Format is inferred by looking at the entire Series data
    - Format can also be manually specified using specific arguments:
        - format - detailed string specifying the full format
        - dayfirst=True - parse 10/11/12 as Nov 10, 2012
        - yearfirst=True - parse 10/11/12 as Nov 12, 2010
- pd.to_timedelta - converts a Series into an absolute difference in time
- Dates can also be parse automatically at read time by passing a list of the columns containing dates to the parse_dates argument of pd.read_csv

Time series

```
>>> data = pd.read_csv('data/AAPL.csv')
>>> data['Date'] = pd.to_datetime(data['Date'].astype('str'))
>>> data.dtypes
    Date          datetime64[ns]
    Open                 float64
    High                 float64
    Low                  float64
    Close                float64
    Adj Close            float64
    Volume               float64
    dtype: object
>>> data = pd.read_csv('data/AAPL.csv', parse_dates=['Date'])
>>> data.dtypes
    Date          datetime64[ns]
    Open                 float64
    High                 float64
    Low                  float64
    Close                float64
    Adj Close            float64
    Volume               float64
    dtype: object
```

Time series

- Sequences of dates/times can be created using pd.date_range(), similar to np.arange()
  - start/end - specify the limits
  - freq - specify frequency (step)
    - B - business day frequency
    - D - calendar day frequency
    - W - weekly frequency
    - M - month end frequency
    - Q - quarter end frequency
    - A - year end frequency
    - H - hourly frequency
    - T - minutely frequency
    - S - secondly frequency
- By setting the index to be a date time, we turn the DataFrame into a Timeseries.
- Pandas has several methods that are specialized to this case
- index.day/index.month/index.year - return the day, month and year of the time series index value

Time series

```
>>> data.set_index('date', inplace=True)
>>> data.index.month
Int64Index([9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
            ...
            8, 8, 9, 9, 9, 9, 9, 9, 9, 9],
           dtype='int64', name=u'date', length=252)
>>> data.index.year
Int64Index([2018, 2018, 2018, 2018, 2018, 2018, 2018, 2018, 2018, 2018,
            ...
            2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019],
           dtype='int64', name=u'date', length=252)
>>> data.index.day
Int64Index([12, 13, 14, 17, 18, 19, 20, 21, 24, 25,
            ...
            29, 30,  3,  4,  5,  6,  9, 10, 11, 12],
           dtype='int64', name=u'date', length=252)
```

DataFrame manipulations

- pd.to_datetime() is just one of several methods that allow us to manipulate and transform the format and contents of Series and DataFrames.

- other methods include:

    - map() - Map values of Series according to input correspondence.

        - func - function, dict or Series to use for mapping

    - transform() - Transform each column/row of the DataFrame using a function. Output must have the same shape as the original

        - axis = 0 - apply function to columns

        - axis = 1 - apply function to rows

    - apply() - Apply a function along an axis of the DataFrame

        - Similar to transform but without the limitation of preserving the shape

DataFrame manipulations

```
>>> df = pd.DataFrame({'A': range(3), 'B': range(1, 4)})
>>> df
   A  B
0  0  1
1  1  2
2  2  3
>>> df['A'].map(lambda x:x+1)
0    1
1    2
2    3
Name: A, dtype: int64
>>> df.transform(lambda x: x + 1)
   A  B
0  1  2
1  2  3
2  3  4
>>> df.apply(np.sum, axis=0)
A    3
B    6
dtype: int64
>>> df.apply(np.sum, axis=1)
0    1
1    3
2    5
dtype: int64
```

groupby

- Sometimes we need to calculate statistics for subsets of our data
- groupby() allows us to group data based on a dict or function
- groupby() returns a GroupBy object that supports several aggregations functions, including:
  - max()/min()/mean()/median()
  - transform()/apply()
  - sum()/cumsum()
  - prod()/cumprod()
  - quantile()

```
>>> data['year'] = data.index.year
>>> data[['Volume', 'year']].groupby('year').sum()
            VOLUME
year
1999    1368545700
2000    1967500700
....
2018    3809327400
2019    2155325200
```

- Each of these functions is applied to the contents of each group

Pivot Tables

- Pandas provides an extremely flexible pivot table implementation
- pd.pivot_table() - Creates a spreadsheet-style pivot table as a DataFrame.
- Three important arguments:
    - values - column to aggregate
    - index - Keys to group by on the pivot table index. If an array is passed, it is being used as the same manner as column values.
    - columns - Keys to group by on the pivot table column.
    - aggfunc - Function to use for aggregation. Defaults to np.mean()
- index, columns and aggfunc can also be lists of keys or functions to use.

Pivot Tables

```
>>> data['month'] = data.index.month
>>> data['year'] = data.index.year
>>> pd.pivot_table(data[data['year']>2014], index='month', columns='year',
values='OPEN', aggfunc=np.mean)
year       2015       2016       2017        2018        2019
month
1      57.908000  58.716316  85.655500  112.000000  101.373809
2      58.626315  57.218000  88.327369  114.032105  104.002105
3      61.074091  59.346818  90.325651  113.820476  103.487619
4      62.199524  61.241428  86.497895  110.738572  109.405238
5      65.499499  62.860476  86.172272  110.869545  111.070455
6      67.851363  62.899545  86.418182  108.087620  109.570000
7      68.380000  63.029000  92.108000  109.610000  114.352273
8      66.343810  65.586087  92.194348  115.566521  108.542728
9      62.007619  66.682857  92.259499  115.208947  112.748572
10     62.393636  68.093809  98.246819  109.563479         NaN
11     66.773500  75.084762  99.258094  109.560000         NaN
12     66.276364  84.971906  106.499500  100.535789         NaN
```

merge/join

- merge() and join() allows us to perform database-style join operation by columns or indexes (rows)

- Some of the most important arguments are common to both methods:

    - on - Column(s) to use for joining, otherwise join on index. Can pass an array as the join key if not already contained in the calling DataFrame. Like an Excel VLOOKUP operation

    - how - Type of join to perform: {'left', 'right', 'outer', 'inner'}

- merge() is more sophisticated and flexible. It also allows us to specify:

    - left_on/right_on - Field names to join on for each DataFrame

    - left_index/right_index - Whether or not to use the left/right index as join key(s)

merge/join

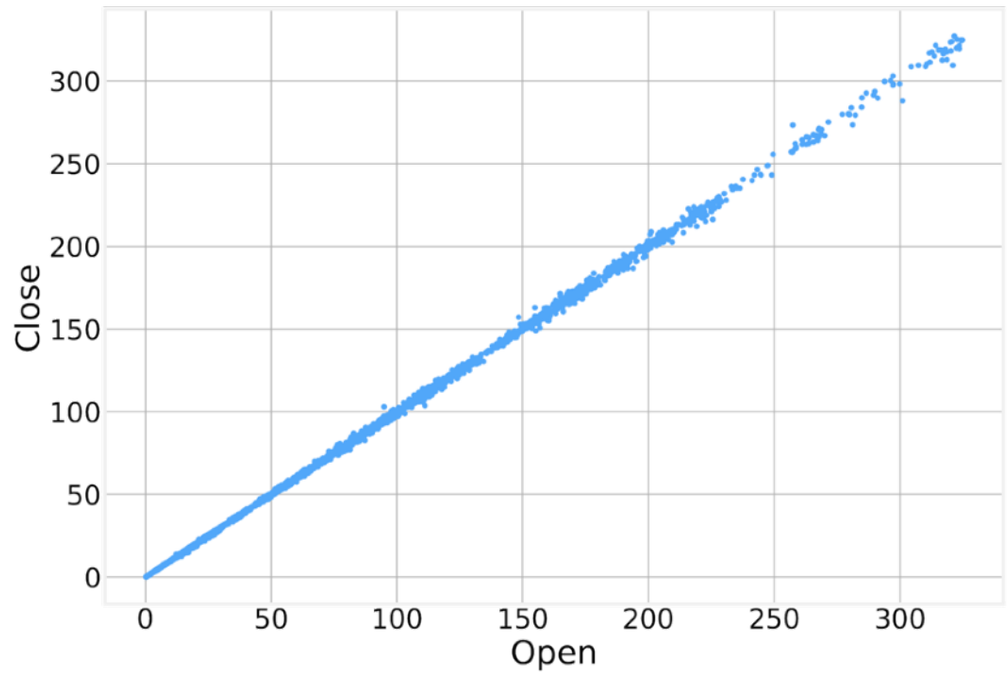
```
>>> A = pd.DataFrame({"lkey":["foo", "bar", "baz", "foo"], "value":[1,2,3,4]})
>>> B = pd.DataFrame({"rkey":["foo", "bar", "qux", "bar"], "value":[5,6,7,8]})
>>> A.merge(B, left_on="lkey", right_on="rkey", how="outer")
  lkey  value_x rkey  value_y
0  foo      1.0  foo      5.0
1  foo      4.0  foo      5.0
2  bar      2.0  bar      6.0
3  bar      2.0  bar      8.0
4  baz      3.0  NaN      NaN
5  NaN      NaN  qux      7.0
>>> A.set_index('lkey', inplace=True)
>>> B.set_index('rkey', inplace=True)
>>> A.join(B, lsuffix="_l", rsuffix="_r", how="inner")
     value_l  value_r
bar        2        6
bar        2        8
foo        1        5
foo        4        5
```

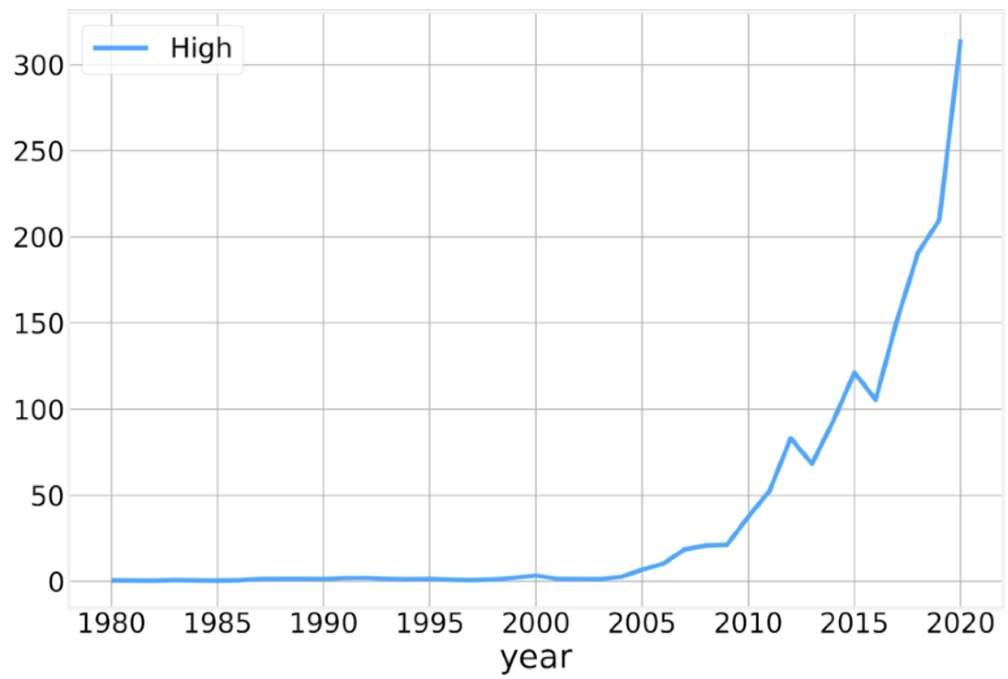
plotting

- Finally, pandas also provides a simple interface for basic plotting through the plot() function
- By specifying some basic parameters the variables plotted and even the kind of plot can be easily modified:
  - x/y -column name to use for the x/y axis
  - kind - type of plot
    - 'line' - line plot (default)
    - 'bar'/'barh' - vertical/horizontal bar plot
    - 'hist' - histogram
    - 'box' - boxplot
    - 'pie' - pie plot
    - 'scatter' - scatter plot
- The plot() function returns a matplotlib Axes object

plotting

```
>>> data.plot.scatter(x='Open', y='Close')
<matplotlib.axes._subplots.AxesSubplot object at
0x11390f190>
>>> data.groupby('year').mean().plot(y='High')
<matplotlib.axes._subplots.AxesSubplot object at
0x11bf20b50>
```





Code - Pandas Review

https://github.com/DataForScience/Timeseries_long

Data Representation

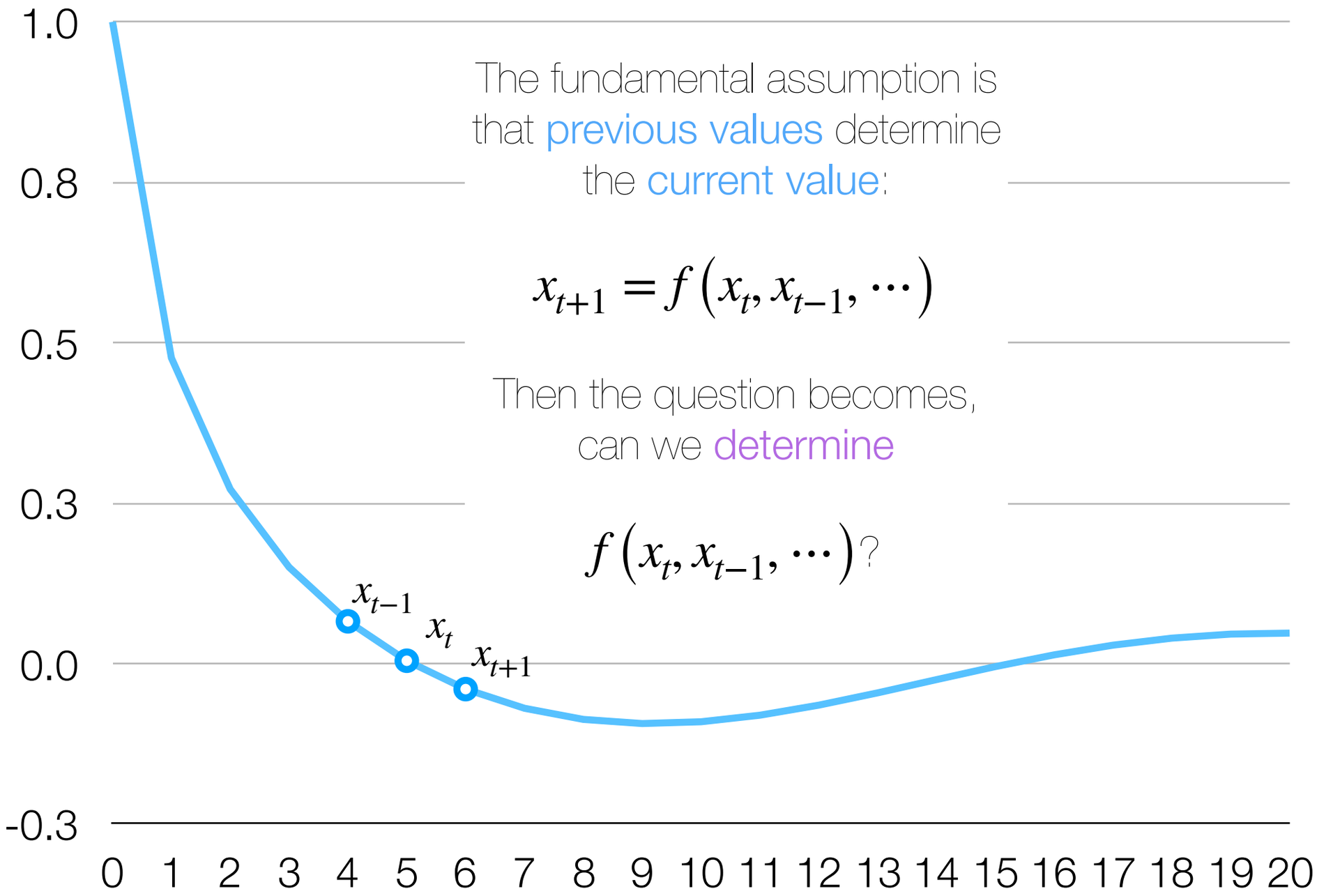
Time series

- A set of values measured sequentially in time
- Values are typically (but not always) measured at equal intervals, x_1, x_2, x_3, x_3, etc…
- Values can be:
  - continuous
  - discrete or symbolic (words).
- Associated with empirical observation of time varying phenomena:
  - Stock market prices (day, hour, minute, tick, etc…)
  - Temperatures (day, minute, second, etc…)
  - Number of patients (week, month, etc…)
  - GDP (quarter, year, etc…)
- Forecasting requires predicting future values based on past behavior

Mathematical Conventions

- The value of the time series at time t is given by x_t
- The values at a given lag l are given by x_{t-l}
- The mean of the overall signal is μ and the corresponding running value is μ_t^w
- The variance of the overall signal is σ and the corresponding running value is σ_t^w
- Running values are calculated over a window of width w

Time series analysis



The fundamental assumption is that previous values determine the current value:

$$x_{t+1} = f\left(x_t, x_{t-1}, \cdots\right)$$

Then the question becomes, can we determine

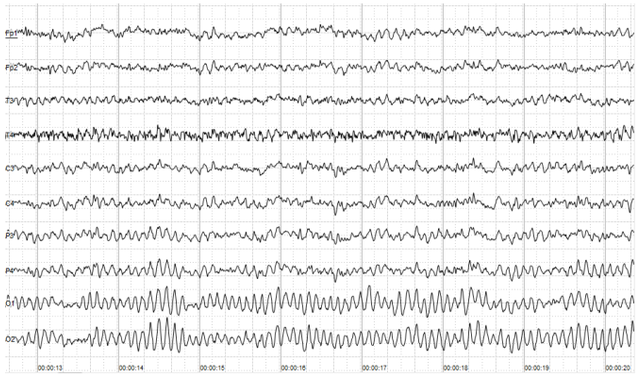
$$f\left(x_t, x_{t-1}, \cdots\right)?$$

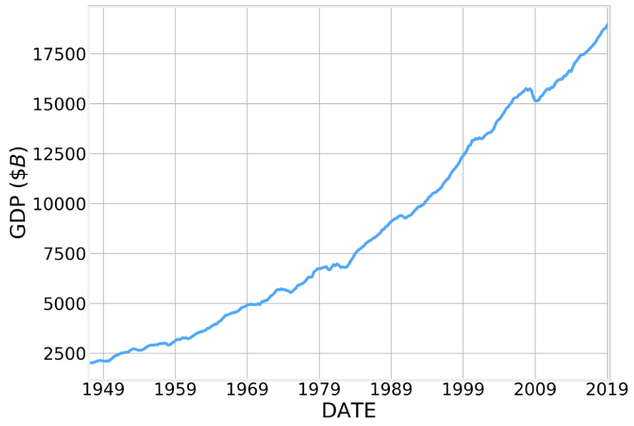
Code - Data Exploration

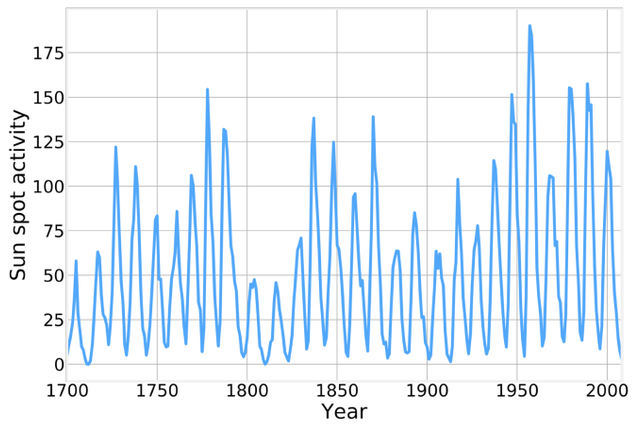
https://github.com/DataForScience/Timeseries_long

Stationarity and Trends

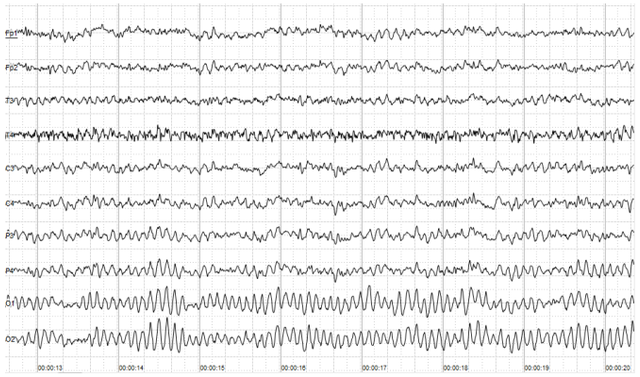
Three fundamental behaviors





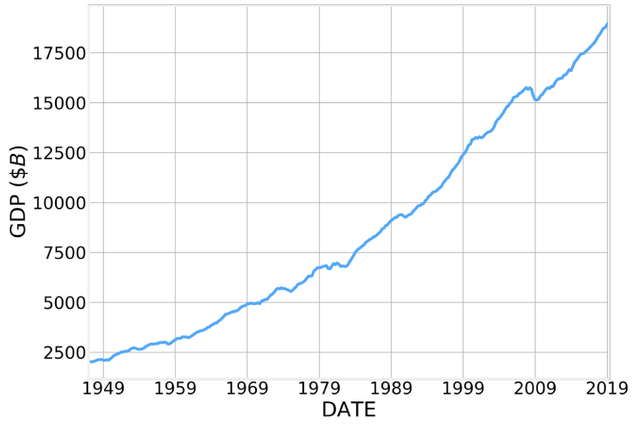


Three fundamental behaviors



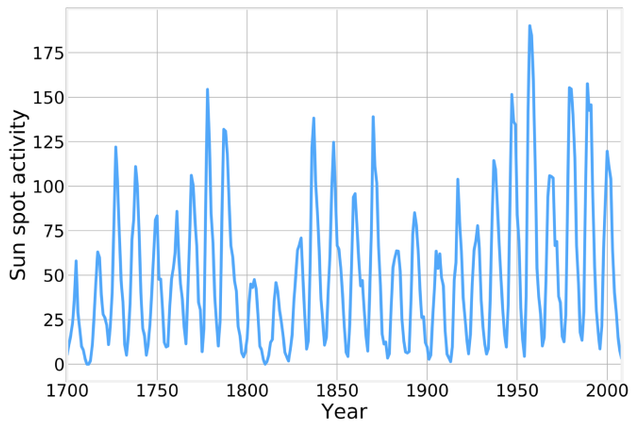
Stationarity

$$\langle x_t \rangle \approx \text{constant}$$



Trend

$$\langle x_t \rangle \approx ct$$



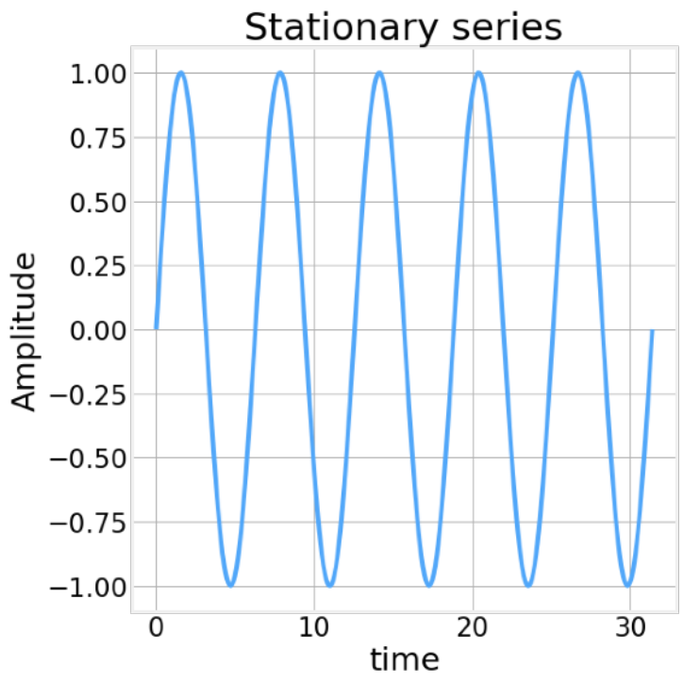
Seasonality

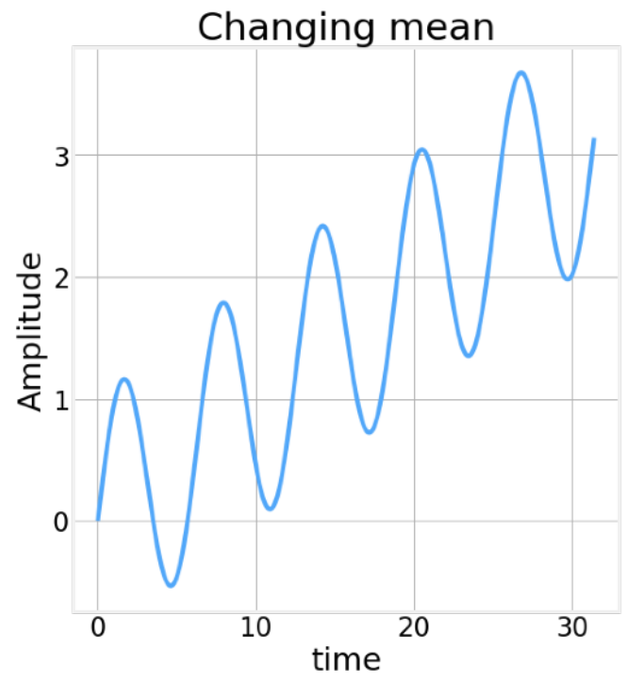
$$x_{t+T} \approx x_t$$

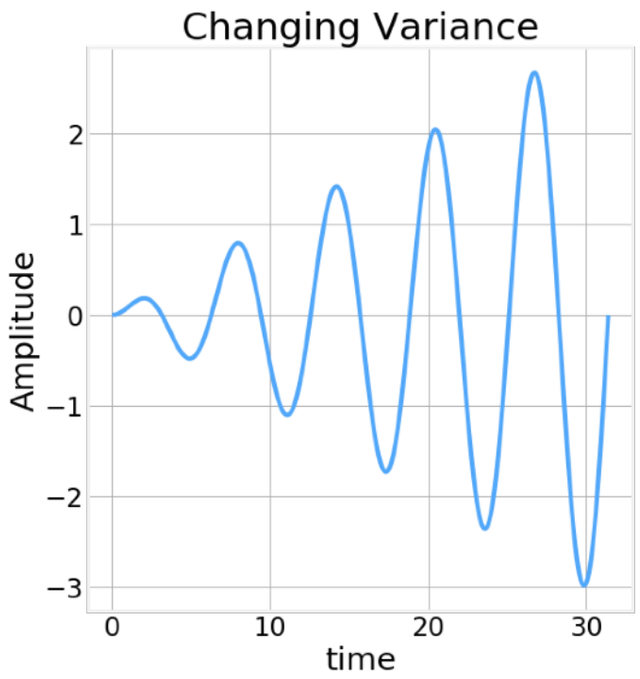
Stationarity

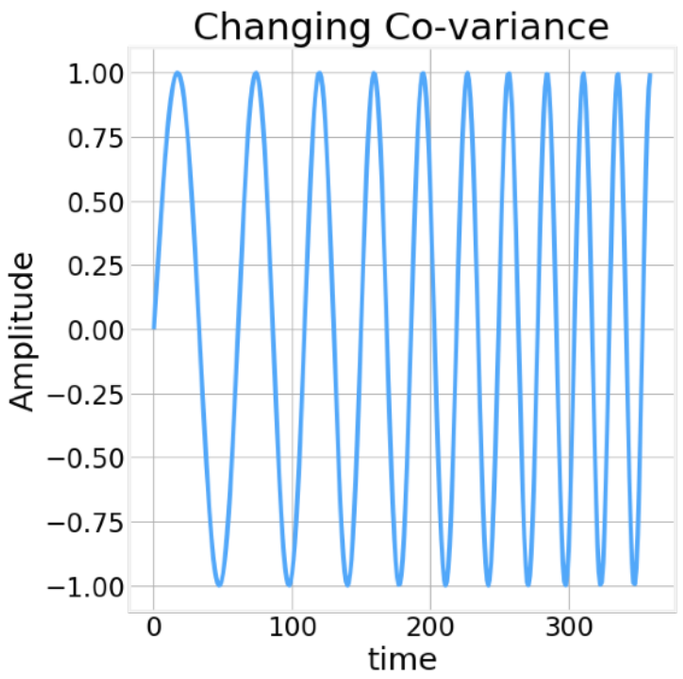
- A time series is said to be stationary if its basic statistical properties are independent of time
- In particular:
    - Mean - Average value stays constant
    - Variance - The width of the curve is bounded
    - Covariance - Correlation between points is independent of time
- Stationary processes are easier to analyze
- Many time series analysis algorithms assume the time series to be stationary
- Several rigorous tests for stationarity have been developed such as the (Augmented) Dickey-Fuller and Hurst Exponent
- Typically, the first step of any analysis is to transform the series to make it stationary

Stationarity

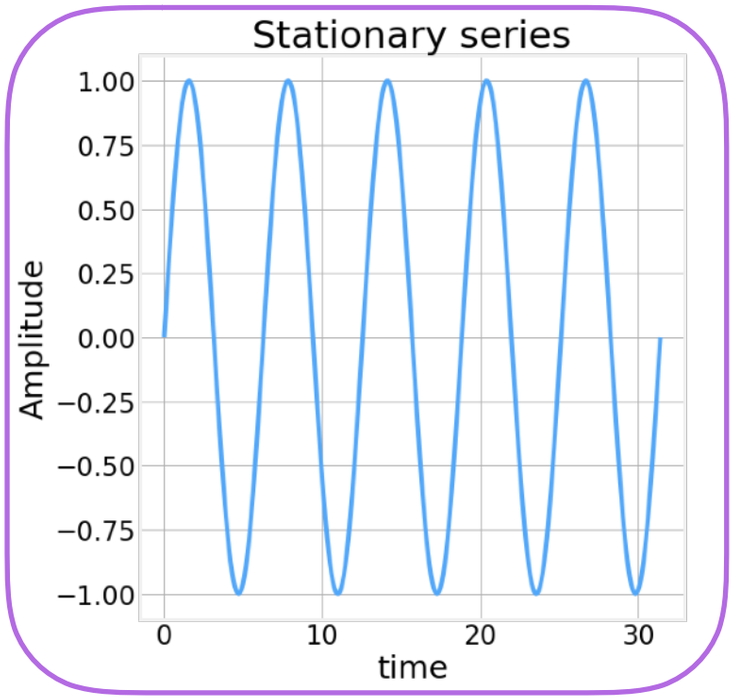


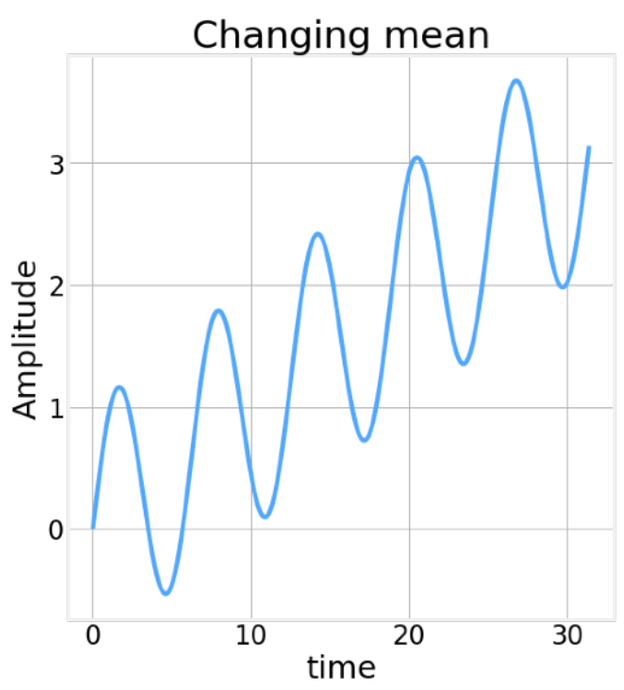


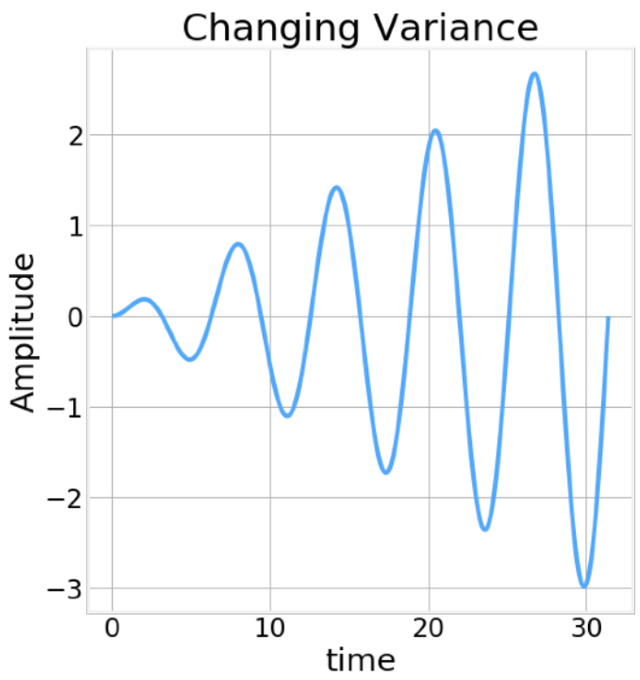


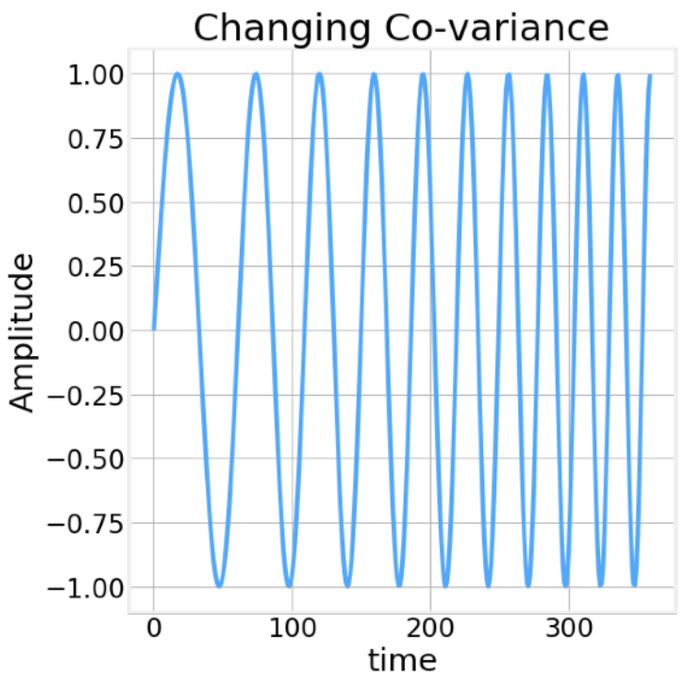


Stationarity





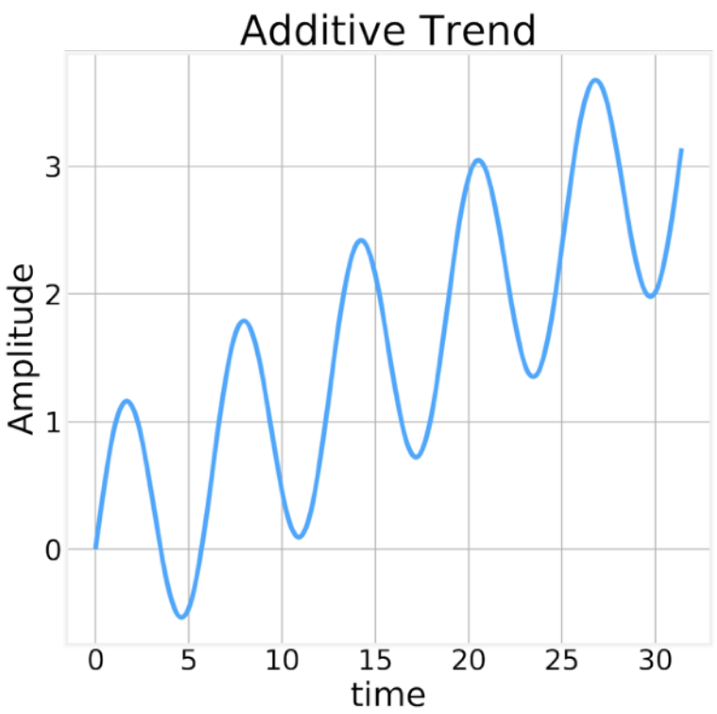


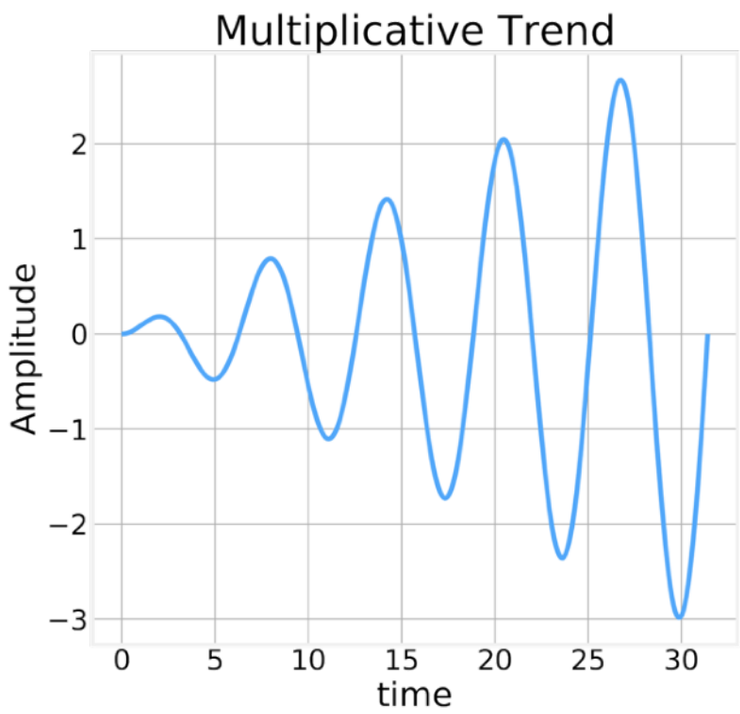


Trend

- Many time series have a clear trend or tendency:
  - Stock market indices tend to go up over time
  - Number of cases of preventable diseases tends to go down over time
  - etc
- Trends can be additive or multiplicative:

$$x = \frac{t}{10} + \sin(t)$$

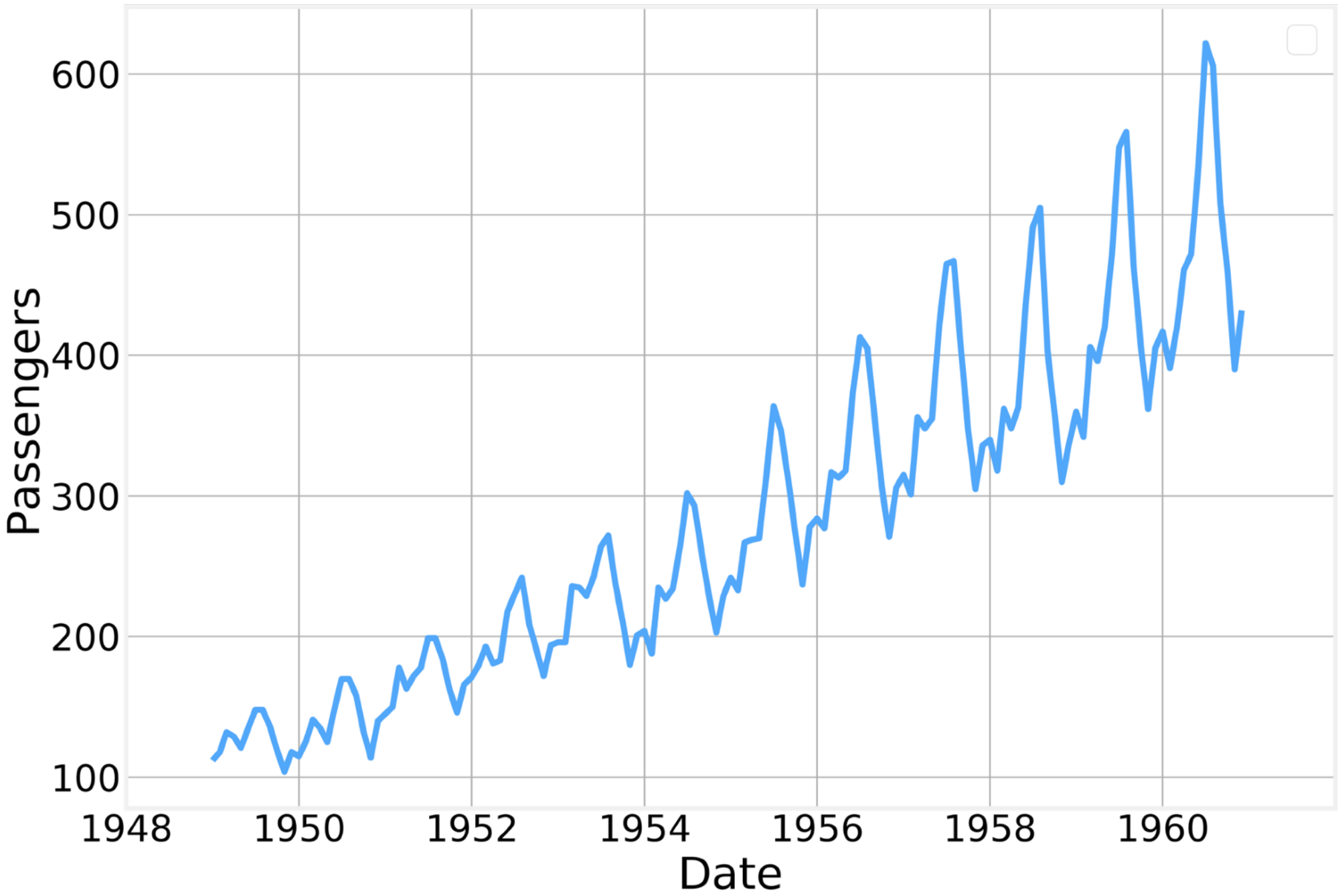




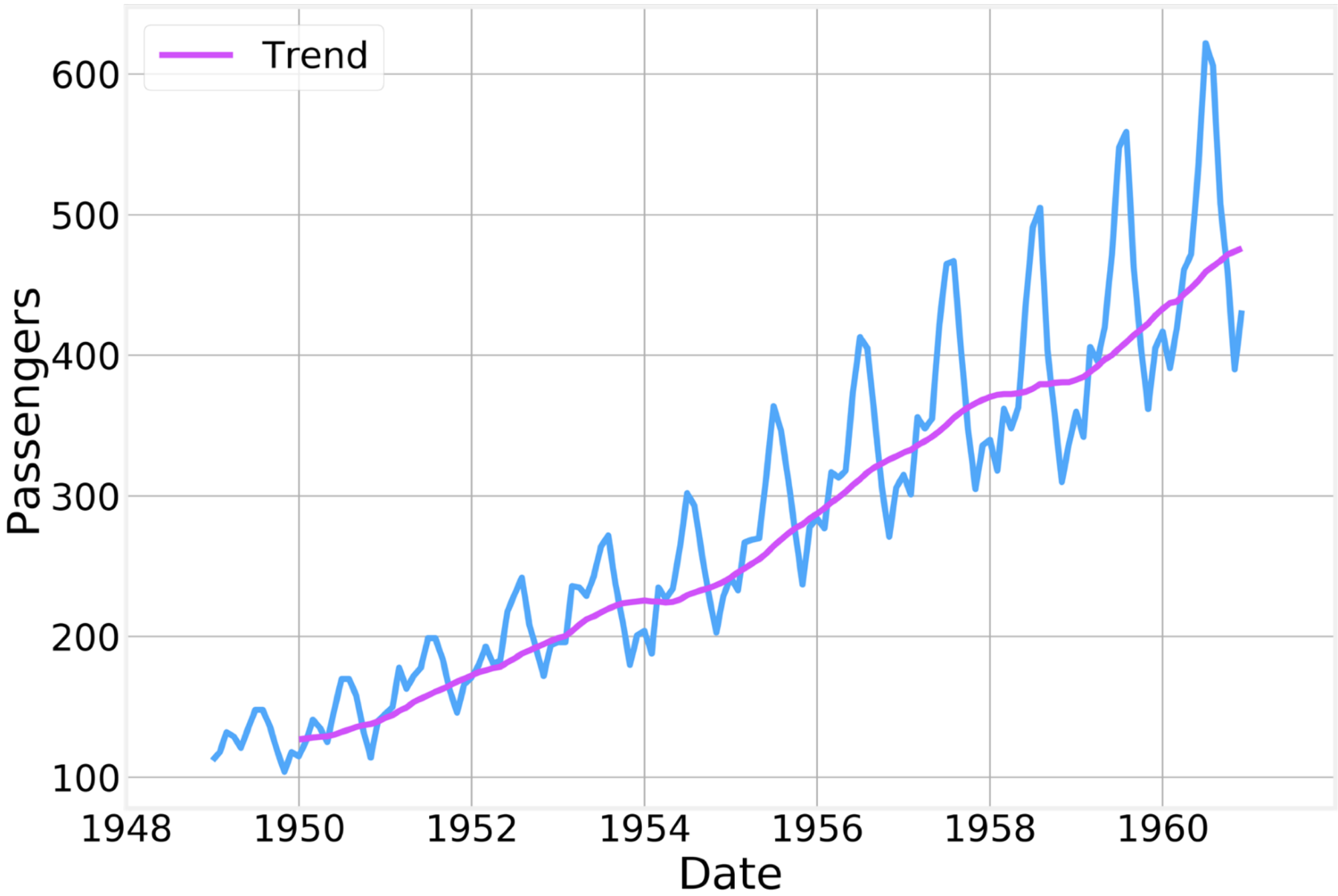
$$x = \frac{t}{10}\sin(t)$$

- Trends can be removed by subtraction or division of the correct values
- One simple way to determine the trend is to calculate a running average over the series

Trend


600
500
400
300
200
100
Passengers
1948
1950
1952
1954
1956
1958
1960
Date

Trend


Trend
600
500
400
300
200
100
Passengers
1948
1950
1952
1954
1956
1958
1960
Date

Code - Stationarity and Trends

https://github.com/DataForScience/Timeseries_long

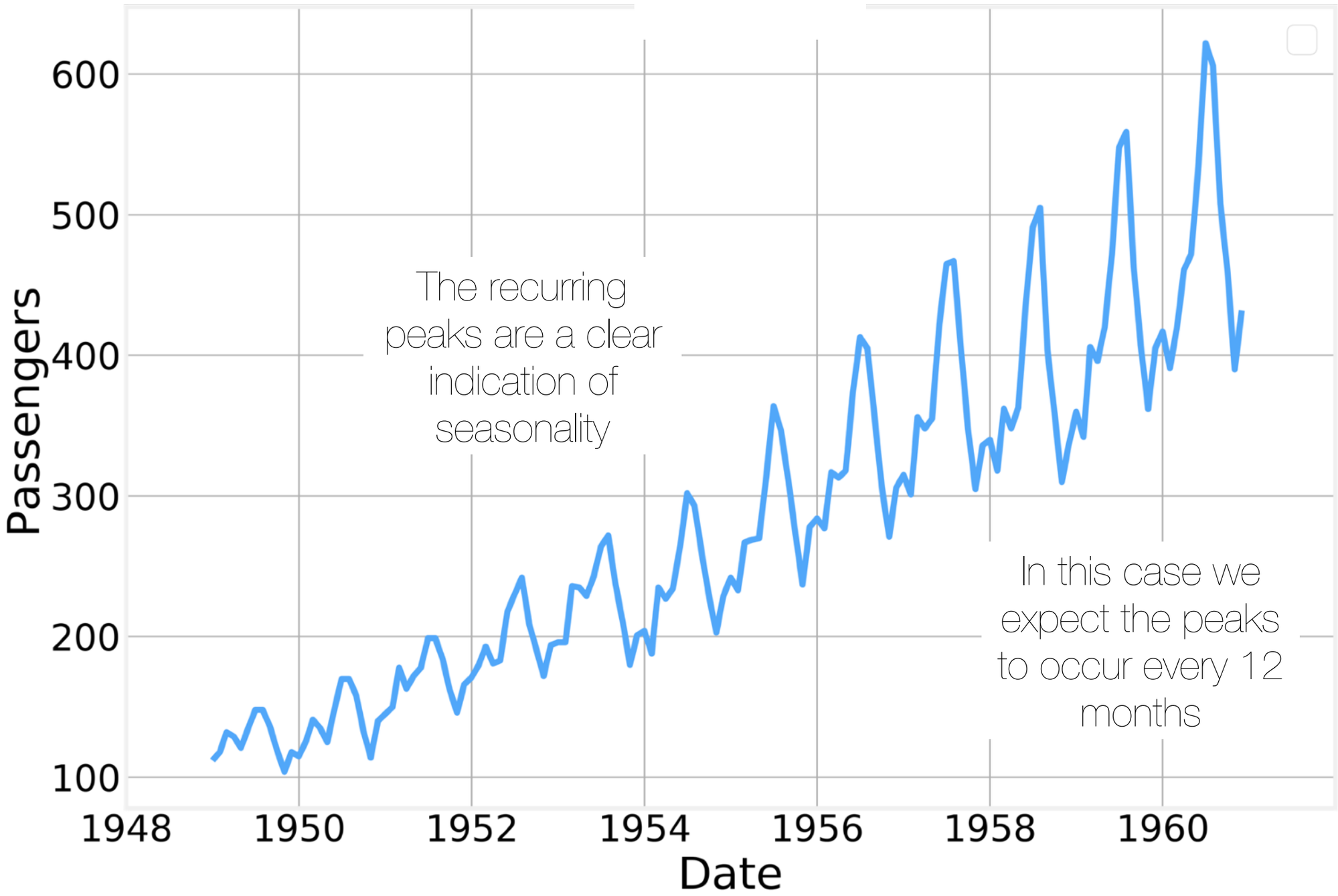
Time Series Decomposition

Seasonality

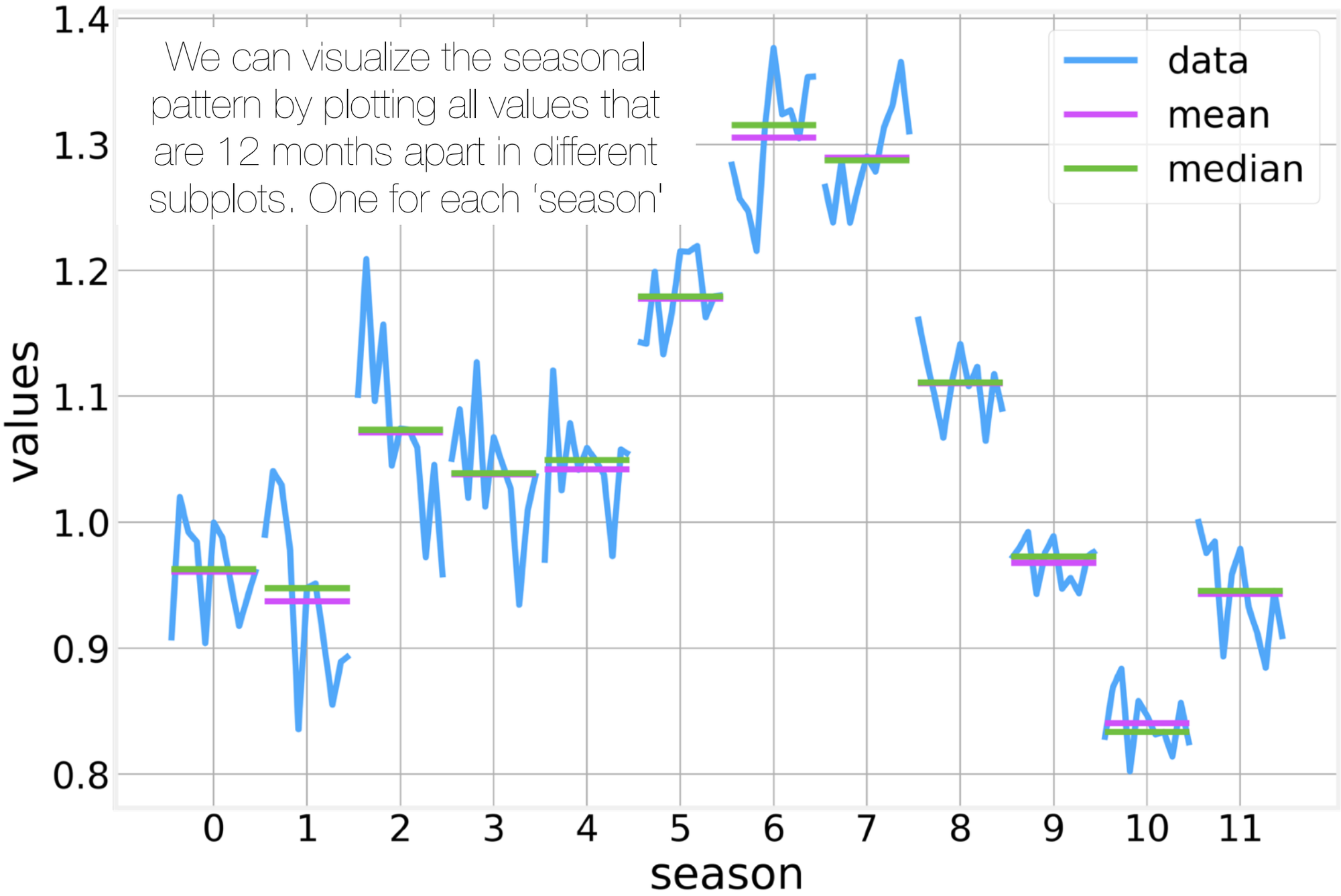
- Many of the phenomena we might be interested in varying in time in a cyclical or seasonal fashion
    - Ice-cream sales peak in the summer and drop in the winter
    - Number of cell phone calls made is larger during the day than at night
    - Many types of crime are more frequent at night than during the day
    - Visits to museums are more frequent in the weekend than in weekdays
    - The stock market grows during bull periods and shrinks during bear periods
    - etc
- Understanding the seasonality of a time series provides important information about its long term behavior and is extremely useful in predicting future values
- If the period is fixed it's called seasonality, while if the period is irregular it's called cyclical

Seasonality

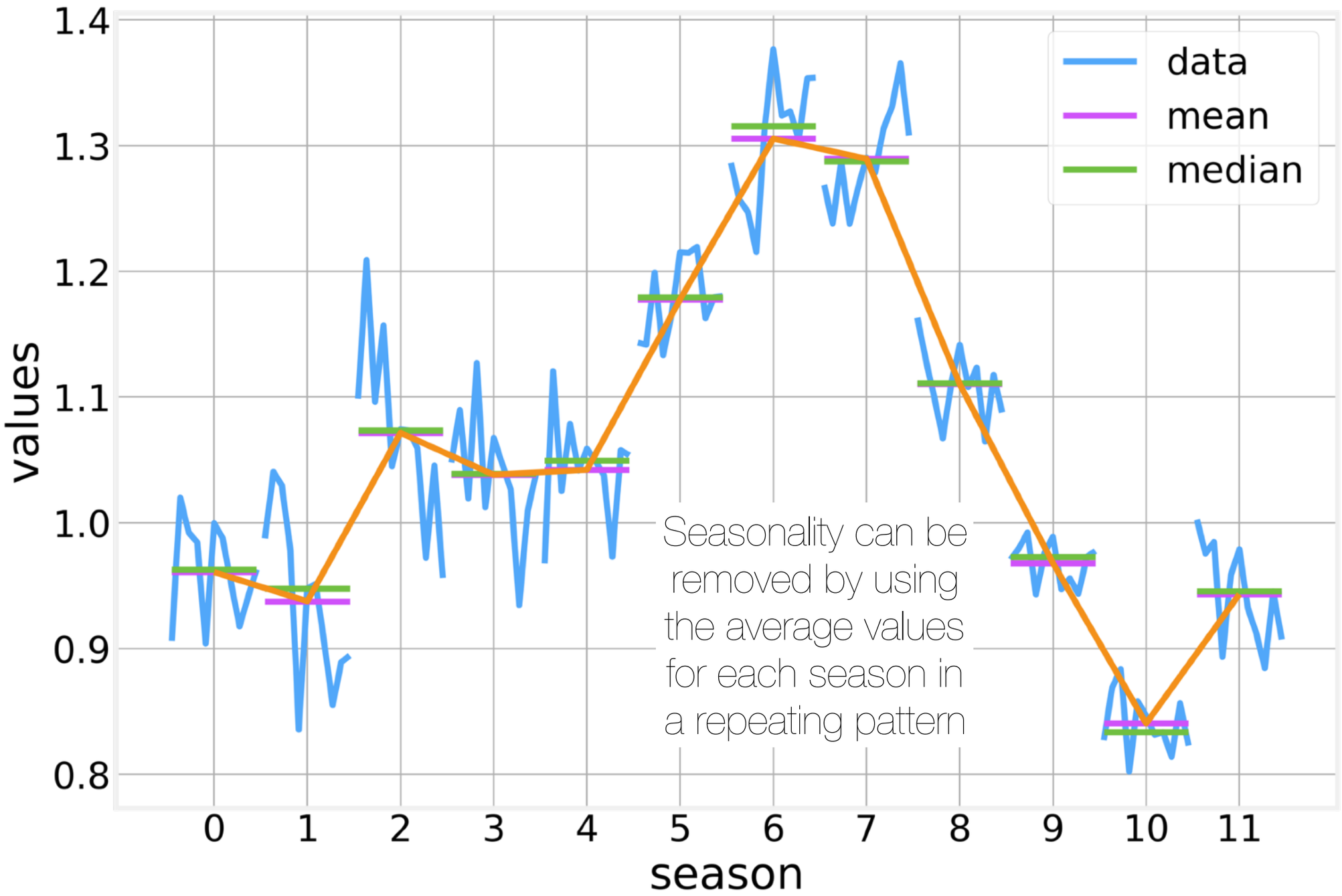
https://www.kaggle.com/chirag19/air-passengers



Seasonality



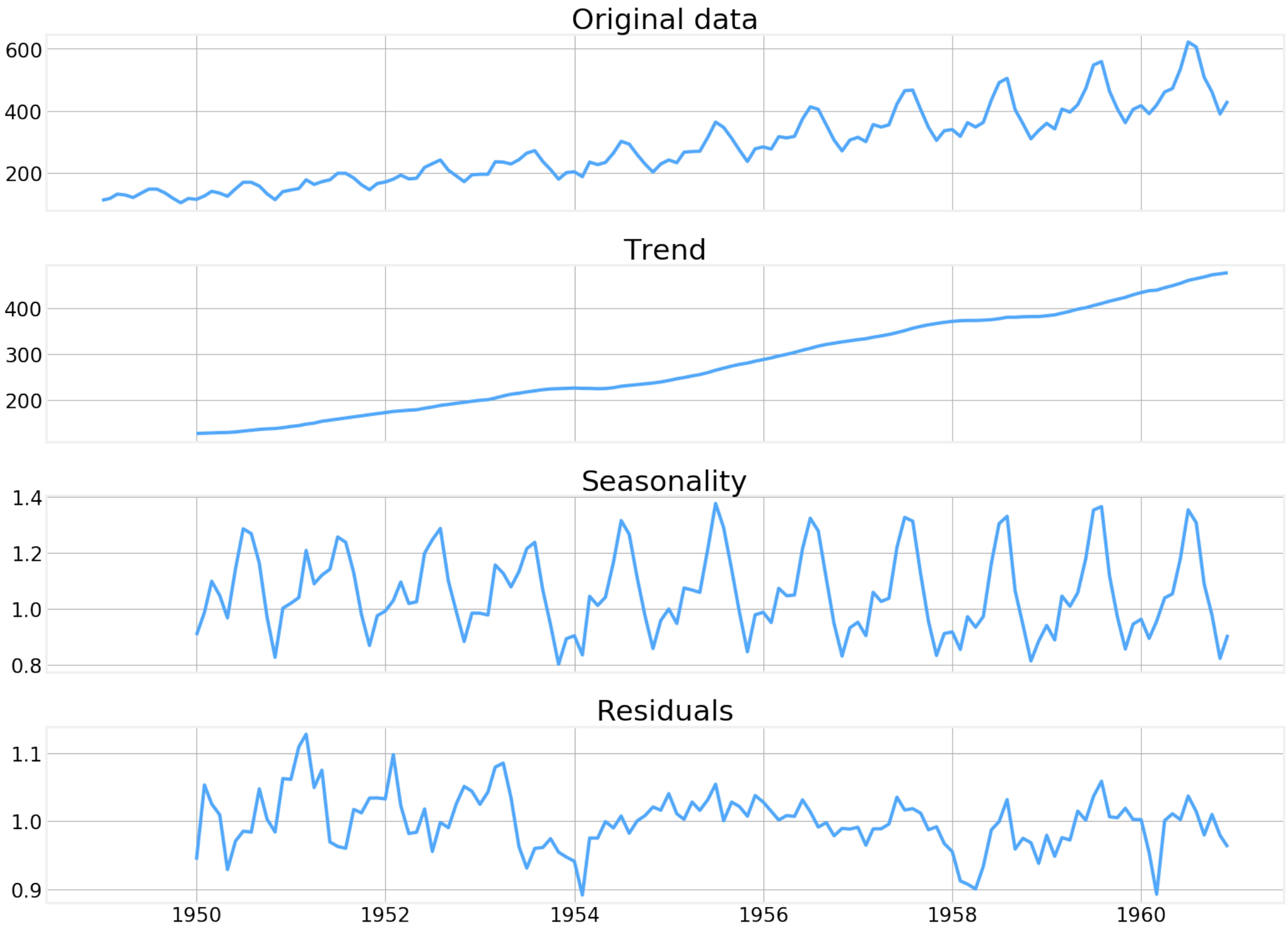
Seasonality



Time series decomposition

- A time series can be decomposed into three components:
  - Trend, T_t
  - Seasonality, S_t
  - Residuals, R_t
- Decompositions can be
  - additive - $x_t = T_t + S_t + R_t$
  - multiplicative - $x_t = T_t \cdot S_t \cdot R_t$
- The residuals are simply what is left of the original signal after we remove the trend and the seasonality
- Residuals are typically stationary

Time series decomposition



Code - Decomposition

https://github.com/DataForScience/Timeseries_long

Processing Timeseries data

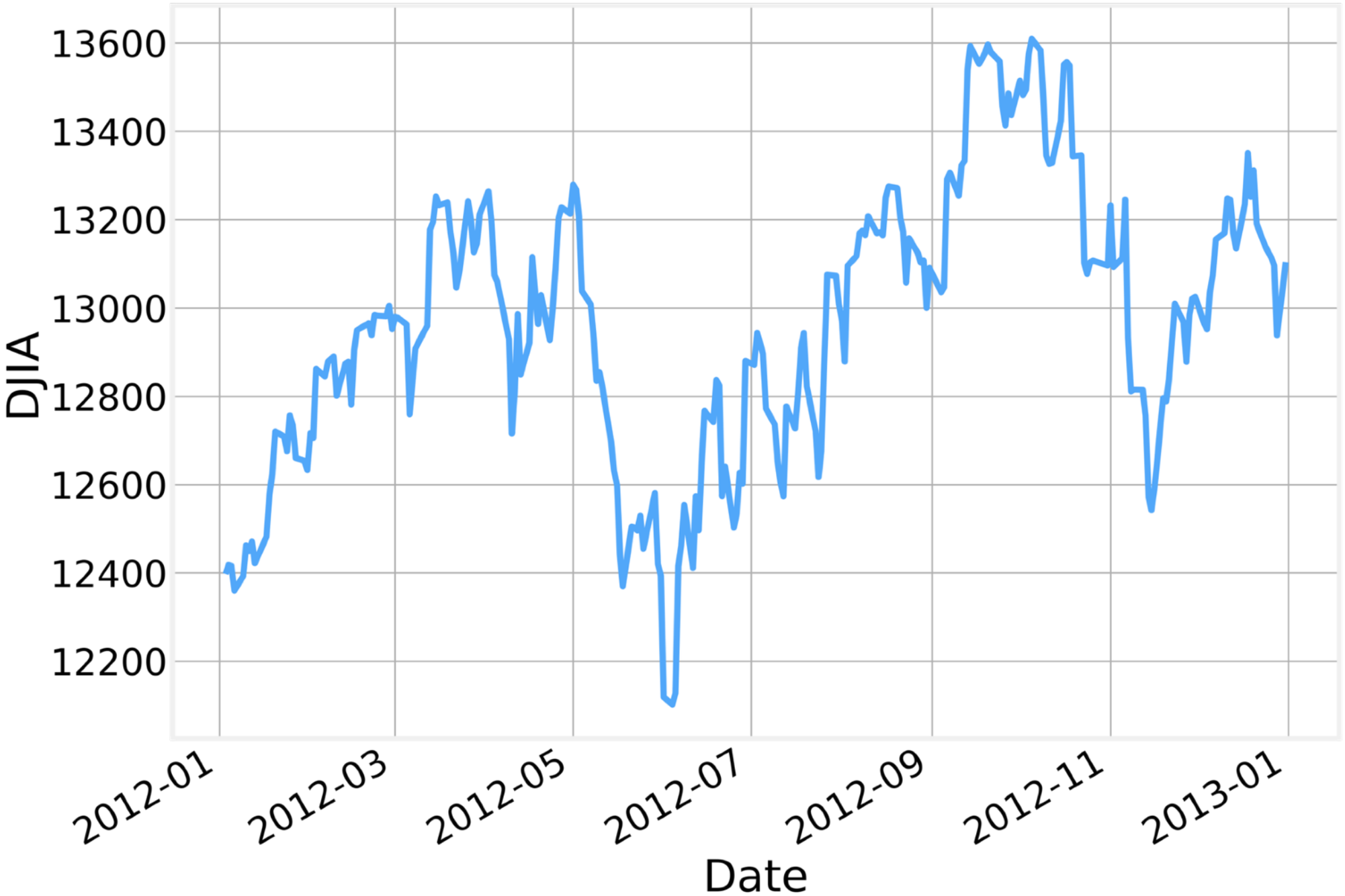
Lagged values

- While analyzing time series, we often refer to values that our time series took 1, 2, 3, etc time steps in the past
- These are known as lagged values and denoted:

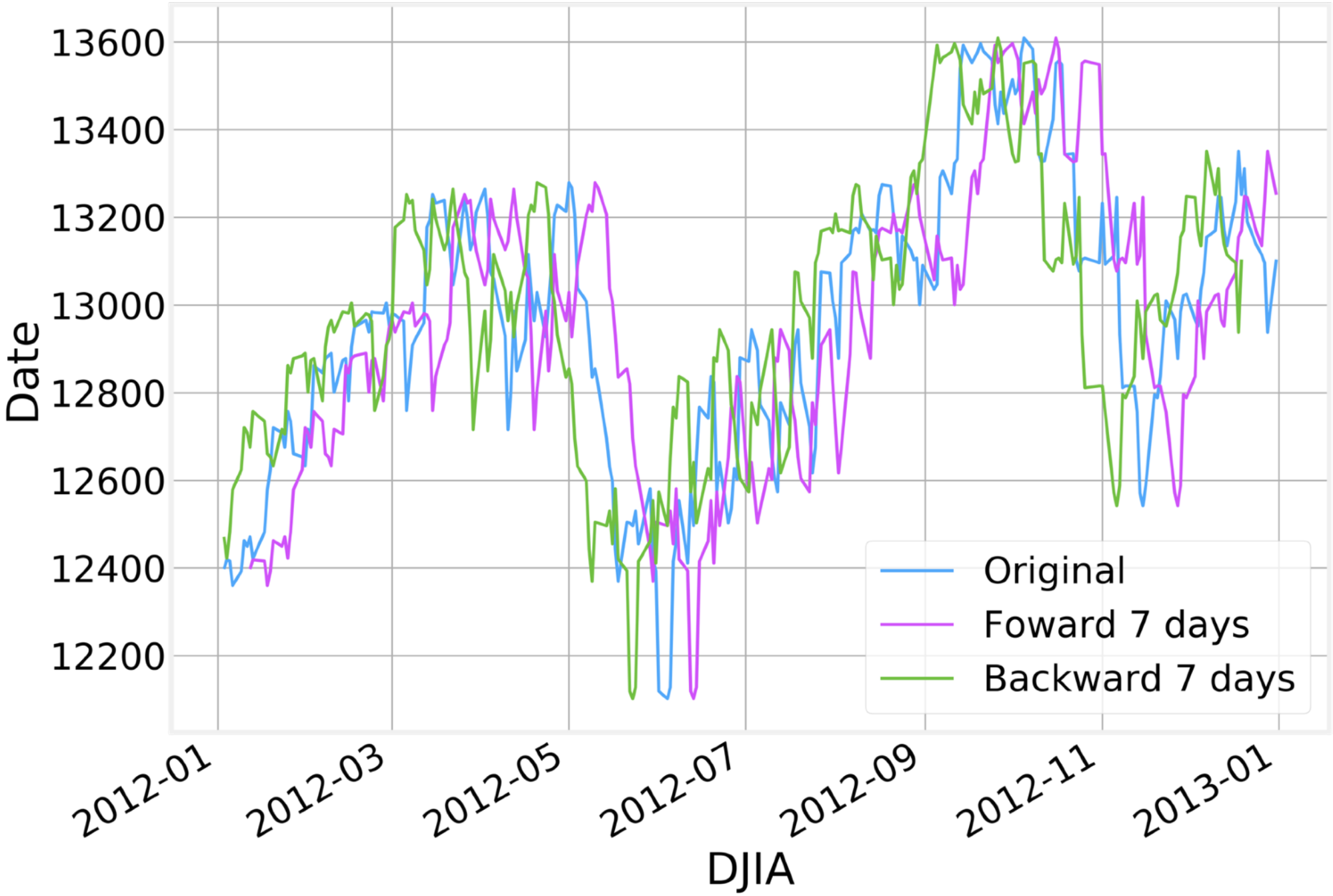
$$x_{t-l}$$

- where l is the value of the lag we are considering.

Lagged values



Lagged values



Differences

- Perhaps the most common use case for lagged values is for the calculation of differences of the form:

$$x_t - x_{t-l}$$

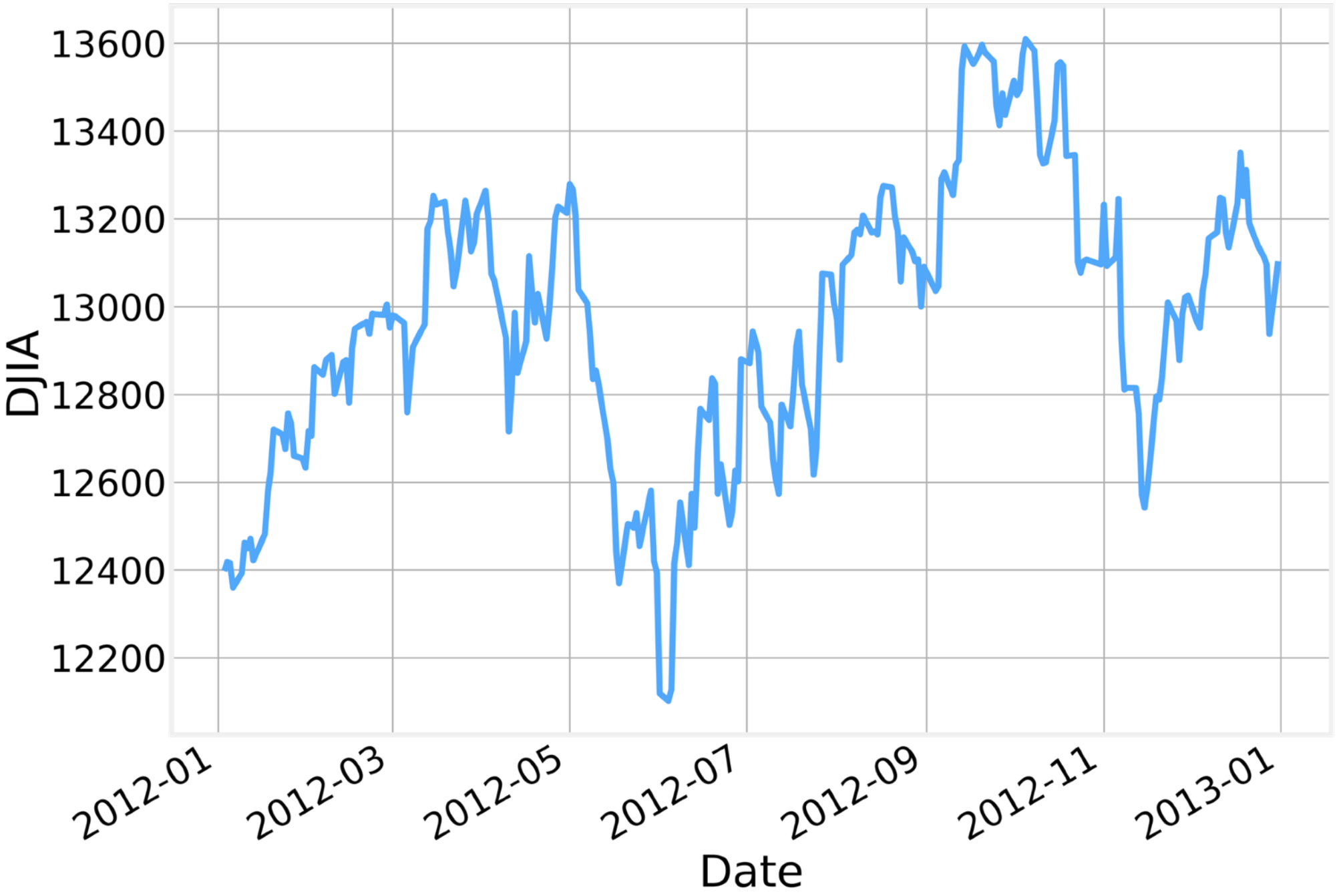
- Where $l \geq 1$ is the value of the lag we are interested in.
- Naturally, higher order differences can also be used, in which case, the difference of the difference is calculated:

$$y_t = x_t - x_{t-l}$$

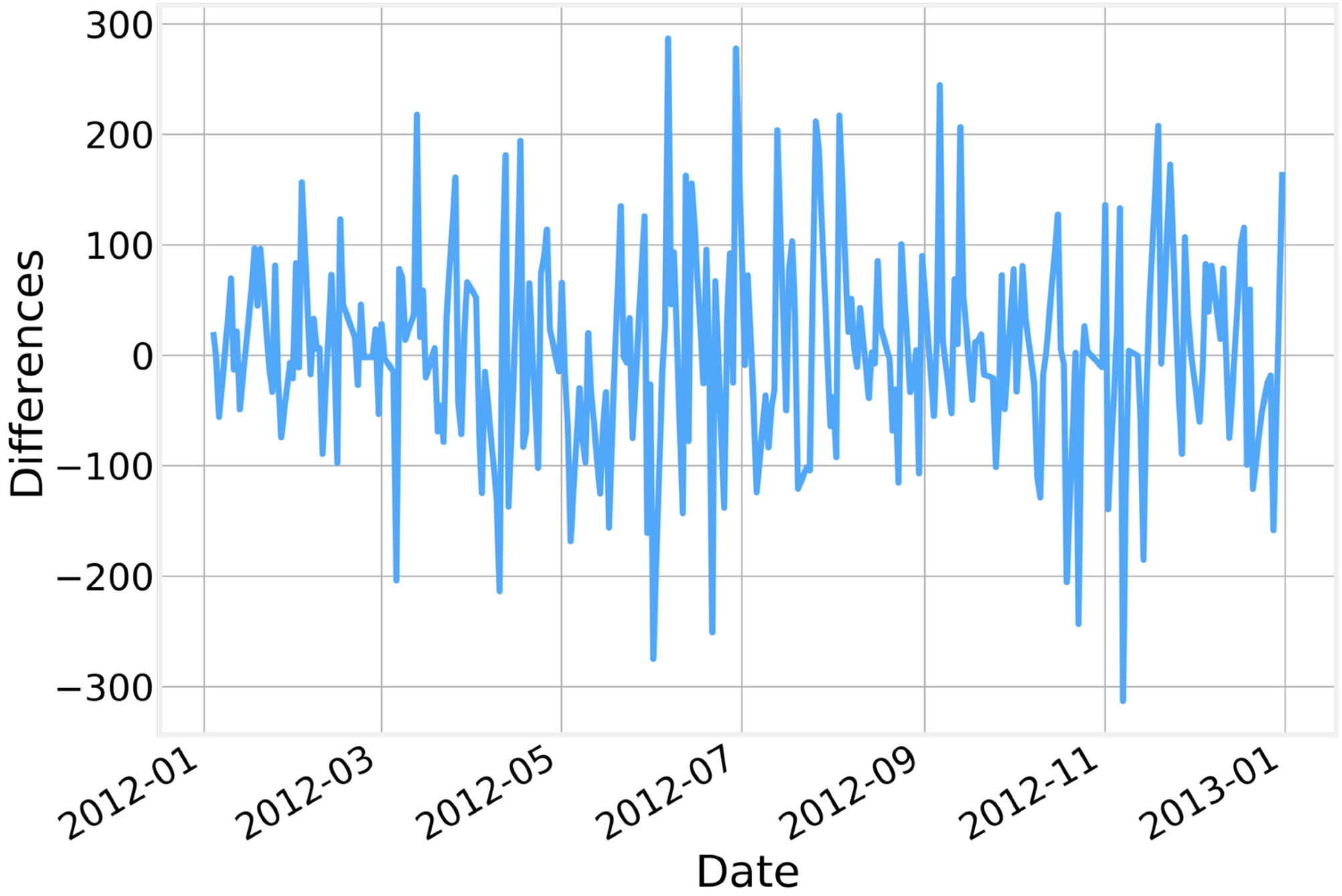
$$z_t = y_t - y_{t-l} \equiv x_t - 2x_{t-l} + x_{t-2l}$$

- This can be thought of as a discrete version of the usual derivative of a function.
- Differences are also a particularly simple way to detrend a time series

Differences

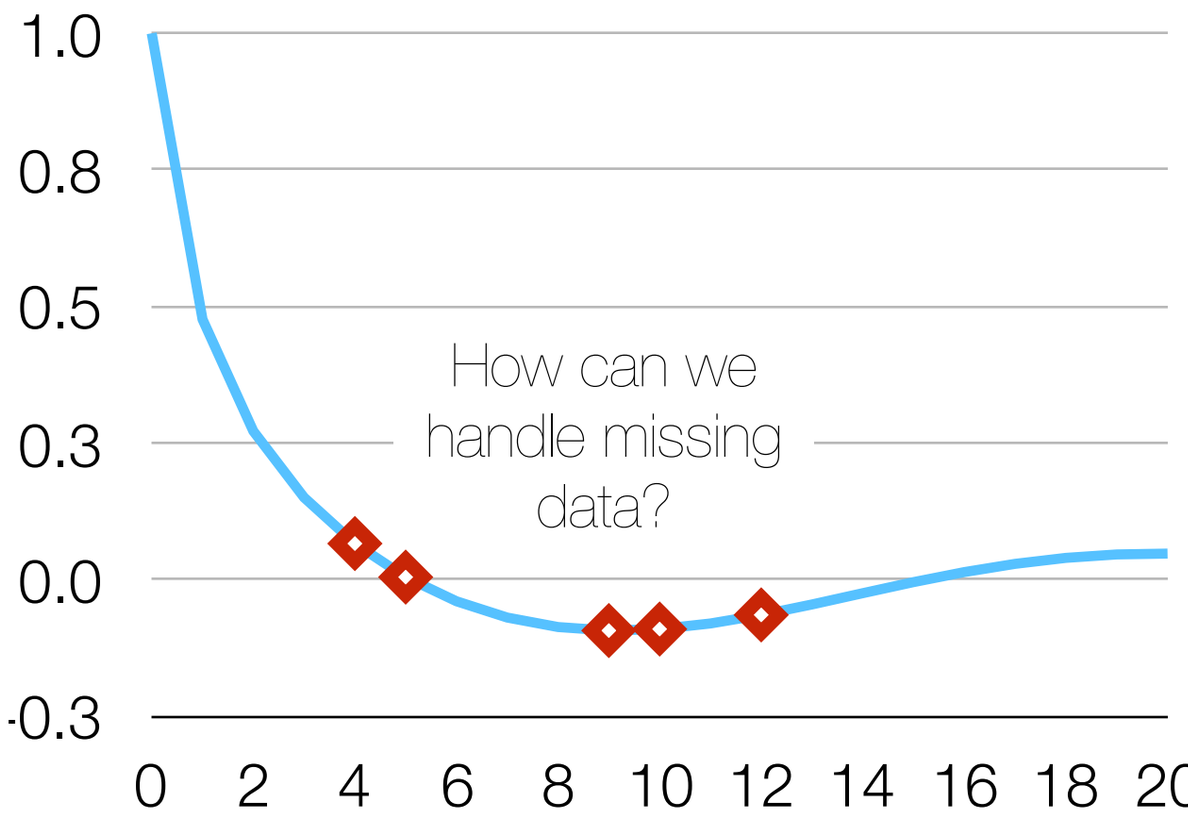


Differences

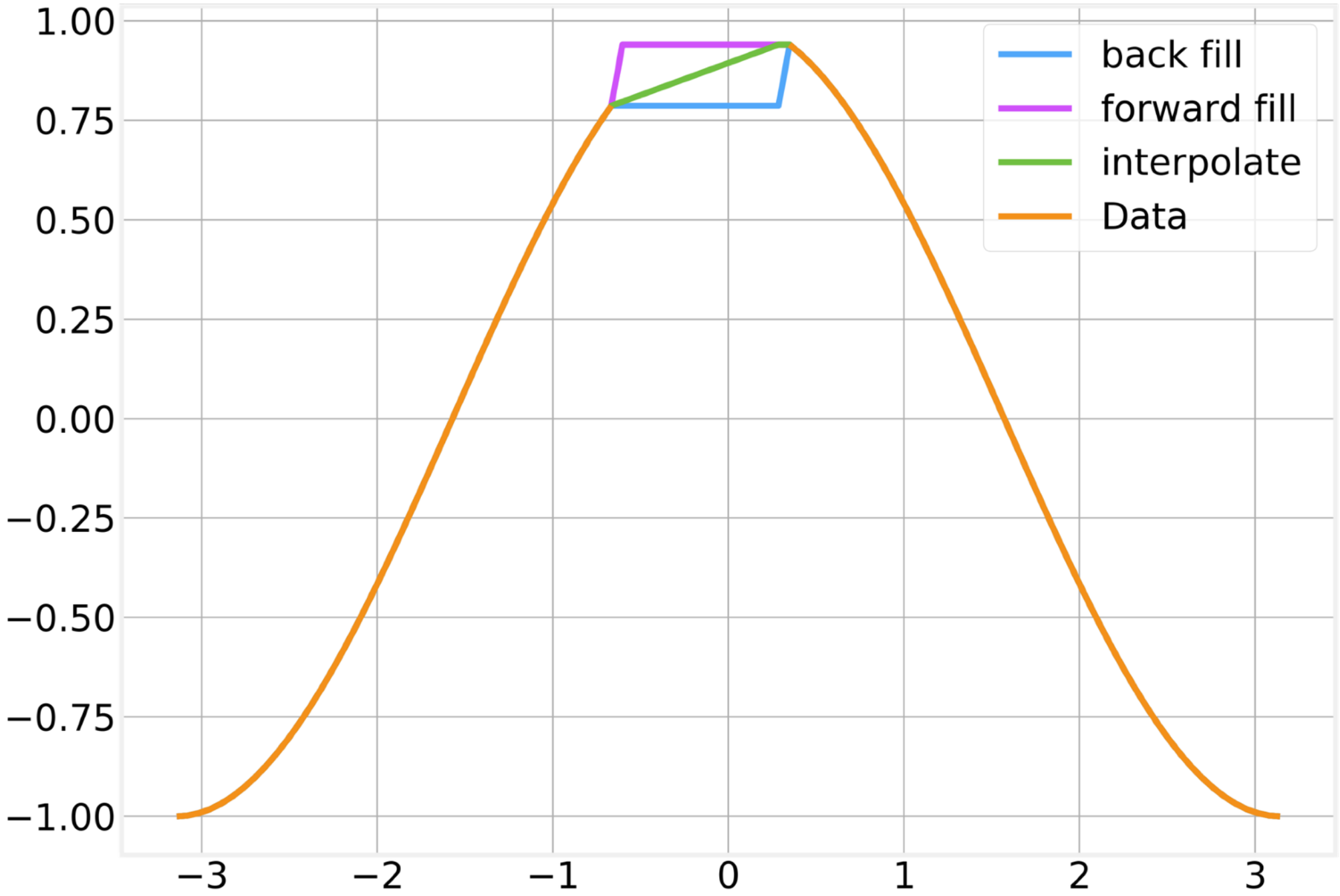


Data Imputation

- Sometimes the time series is incomplete.
- Missing data points can be due to data corruption, data collection issues, etc.
- Missing values are represented as nan
- Several techniques have been developed to handle this case:
  - back fill - keep the last previous value
  - forward fill - keep the next value
  - interpolate - add values by interpolating between the previous and the next value
  - imputation - add values based on what we expect the missing values to be



Data Imputation



Resampling

- Time series typically have an intrinsic time scale at which the data was collected: ticks, seconds, days, months, etc

- In many cases, our analysis requires that we resample the data to a different time scale

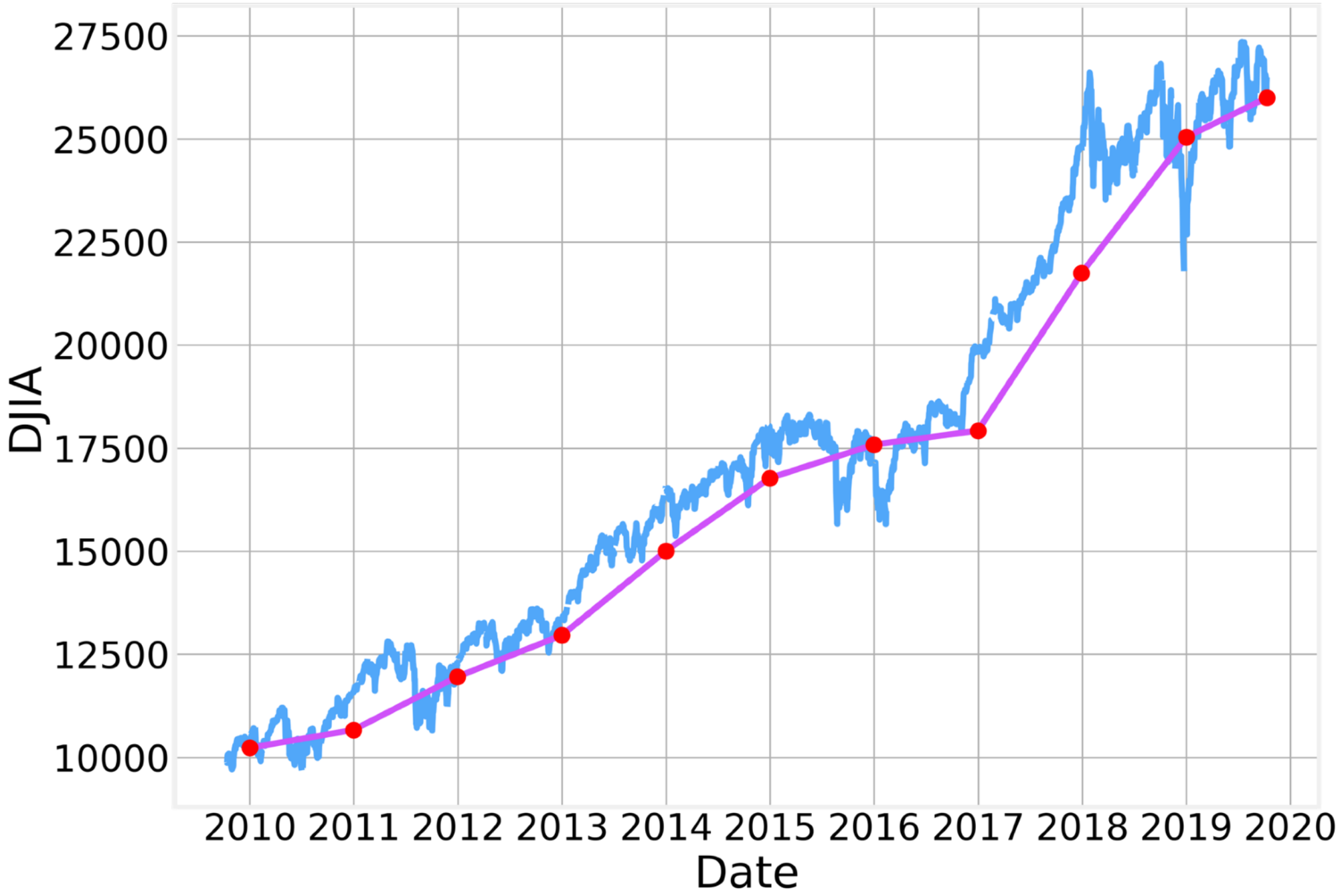
- Resampling to a longer timescale is relatively simple and similar to aggregation:

    - Transforming from daily to weekly frequency requires simply aggregating by week

- Resampling to shorter timescales requires interpolation or imputation to make up for the missing values

    - Going from weekly to daily frequency requires specifying how to allocate the values for each day of the week

Resampling


27500
25000
22500
20000
17500
15000
12500
10000
DJIA
2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020
Date

Jackknife Estimation

- Also known as 'leave one out' estimation
- Commonly used for mean and variance estimates
- The Jackknife estimate of a parameter is the average value of the parameter calculated by omitting each of the values one at a time.
- If μ_i is the mean calculated by omitting the i^{th} value, then the Jackknife estimate of the mean is given by:

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N}\mu_i$$

- And the variance of the estimate is:

$$\hat{\sigma}\left(\hat{\mu}\right) = \frac{N-1}{N}\sum_{i=1}^{N}\left(\mu_i - \hat{\mu}\right)^2$$

Bootstrapping

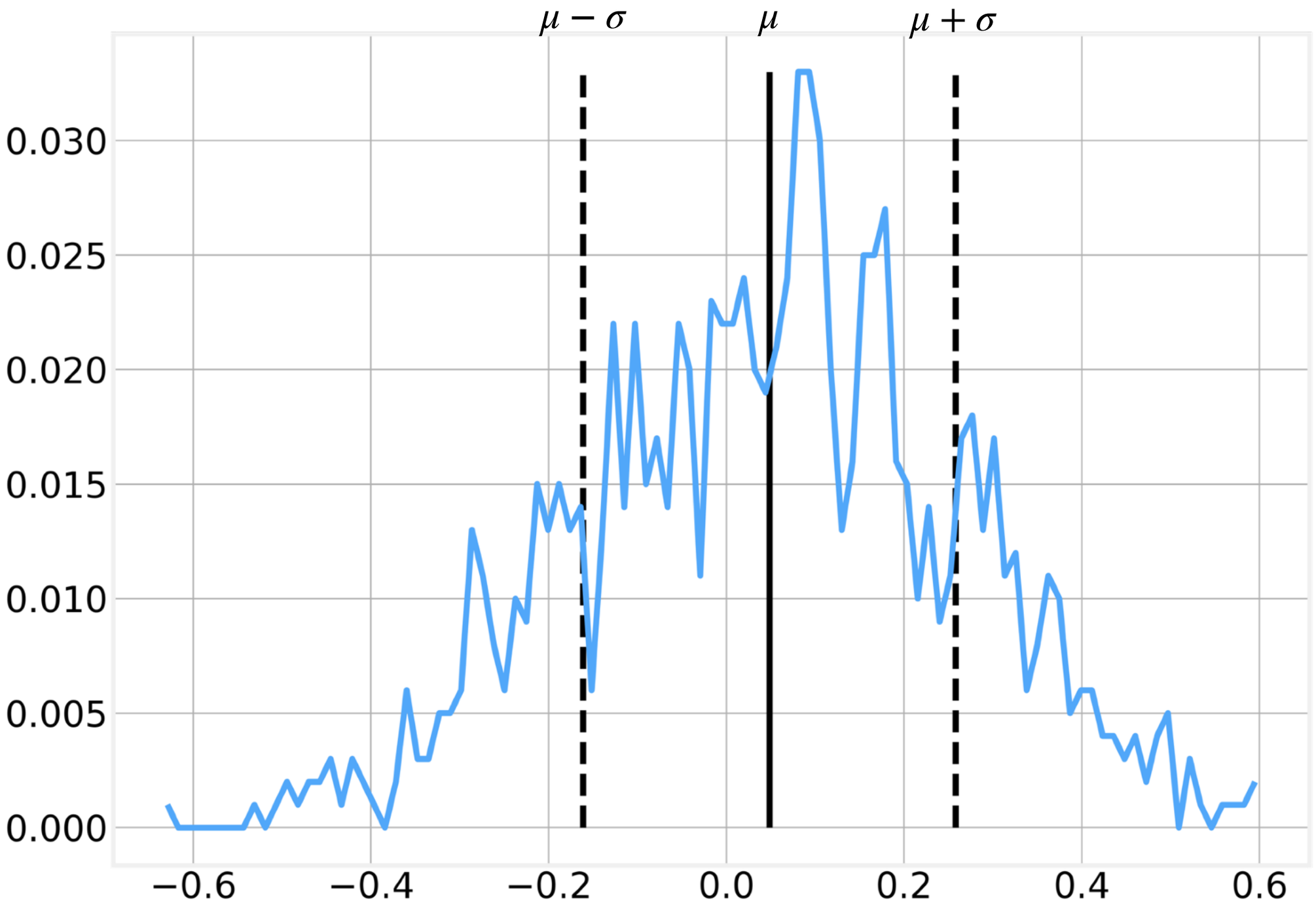
- Bootstrapping is another alternative to estimate statistical properties such as mean and variance

- Bootstrapping measures the desired property in a large number of samples (with replacement), of the observed dataset.

- Each sample has equal size to the observed dataset.

- From the entire population of samples, the empirical bootstrap distribution of the expected values can be obtained to provide information about the distribution in the total population

Bootstrapping



Code - Transformations

https://github.com/DataForScience/Timeseries_long

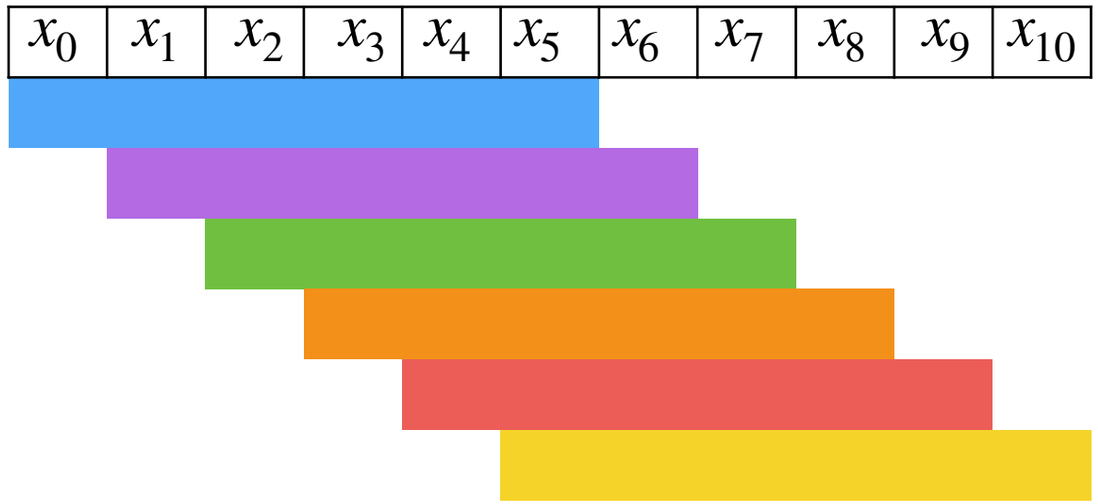
Running Values

Windowing

- When analyzing the temporal behavior of of a signal, we often need to evaluate if specific quantities are time varying or not

- A common approach is to use sliding windows of a given length to evaluate the required values

- So a sliding window of width 6 on a series of length 11 would look like:



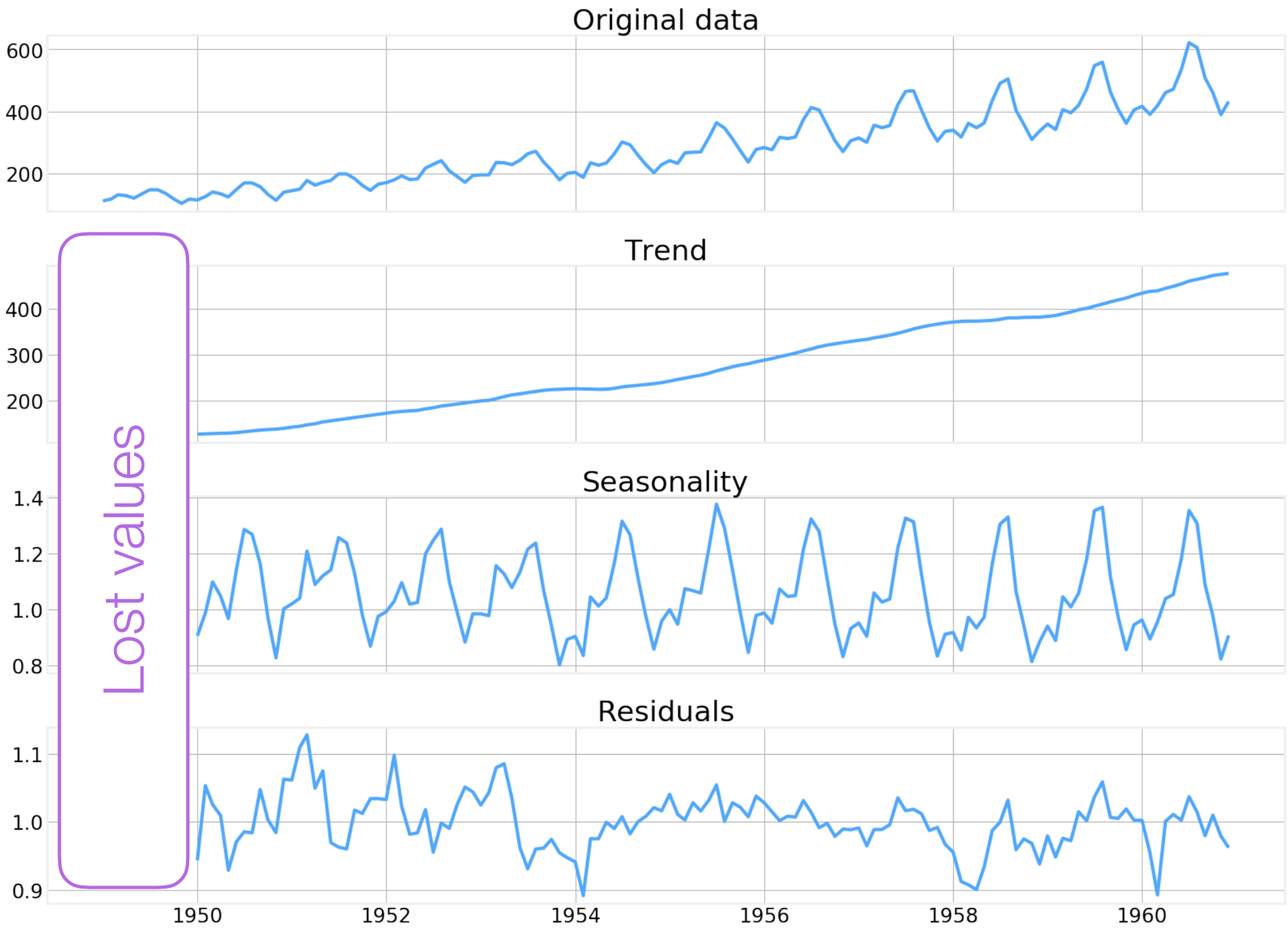
$$
=
\begin{matrix}
x_0 & x_1 & x_2 & x_3 & x_4 & x_5 \\
x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\
x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\
x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \\
x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \\
x_5 & x_6 & x_7 & x_8 & x_9 & x_{10}
\end{matrix}
$$

- and we would calculate the metric of interest within each window.

- By using sliding windows we necessary lose some data points

Windowing

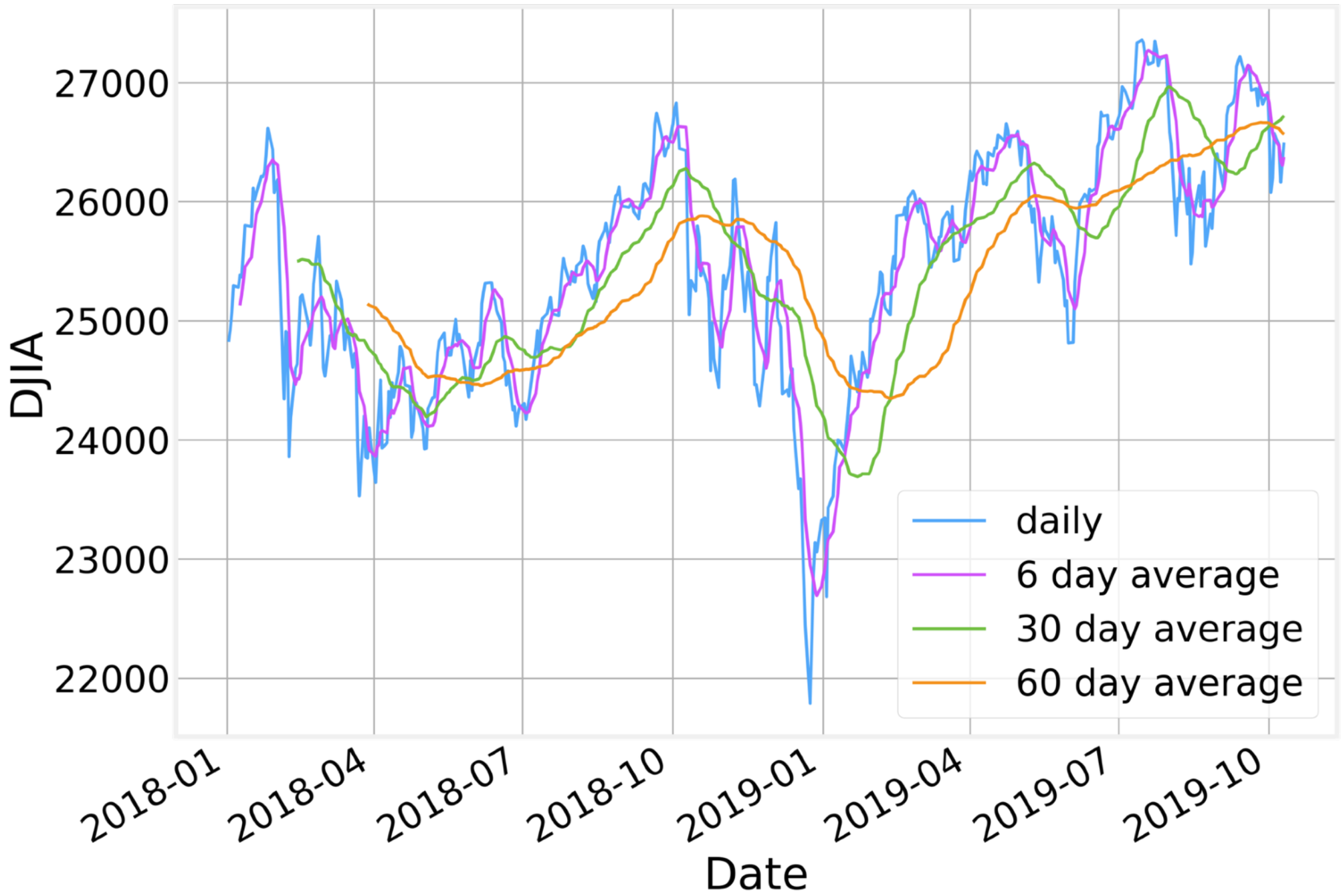
One common approach is to place all "lost values" at the beginning as it avoids "future leaking" when splitting the dataset



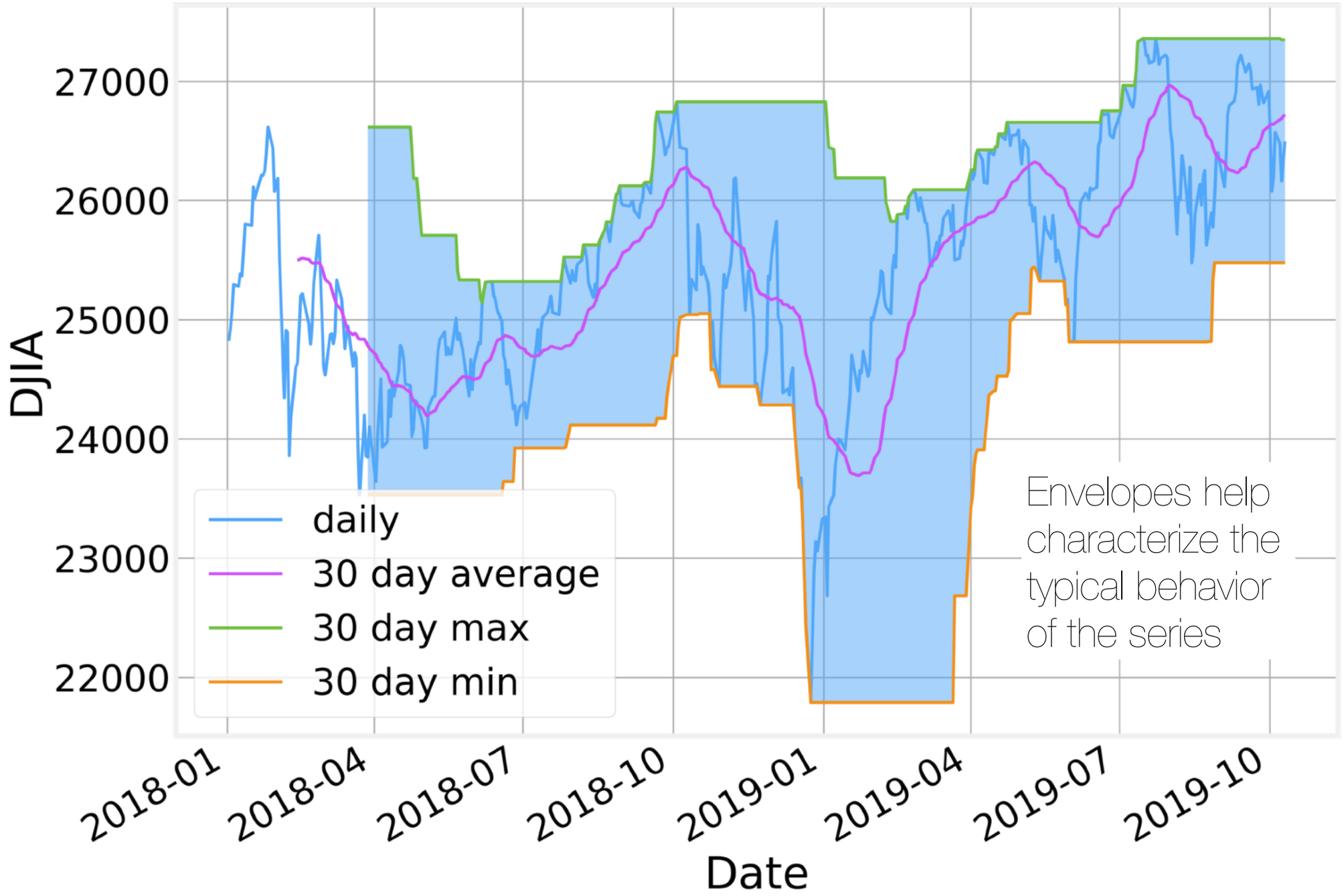
Running Values

- In the first part of the lecture, we already used running averages to detrend a time series
- Other common metrics are:
    - Variance
    - Maximum value
    - Minimum value
    - etc…
- One important detail to note is that while using windowing to calculate running values we "lose" a number of points equal to the width of the window
- Depending on the application we can choose to place the missing values in either or (or even both) extremes of the time interval

Running Values



Envelopes

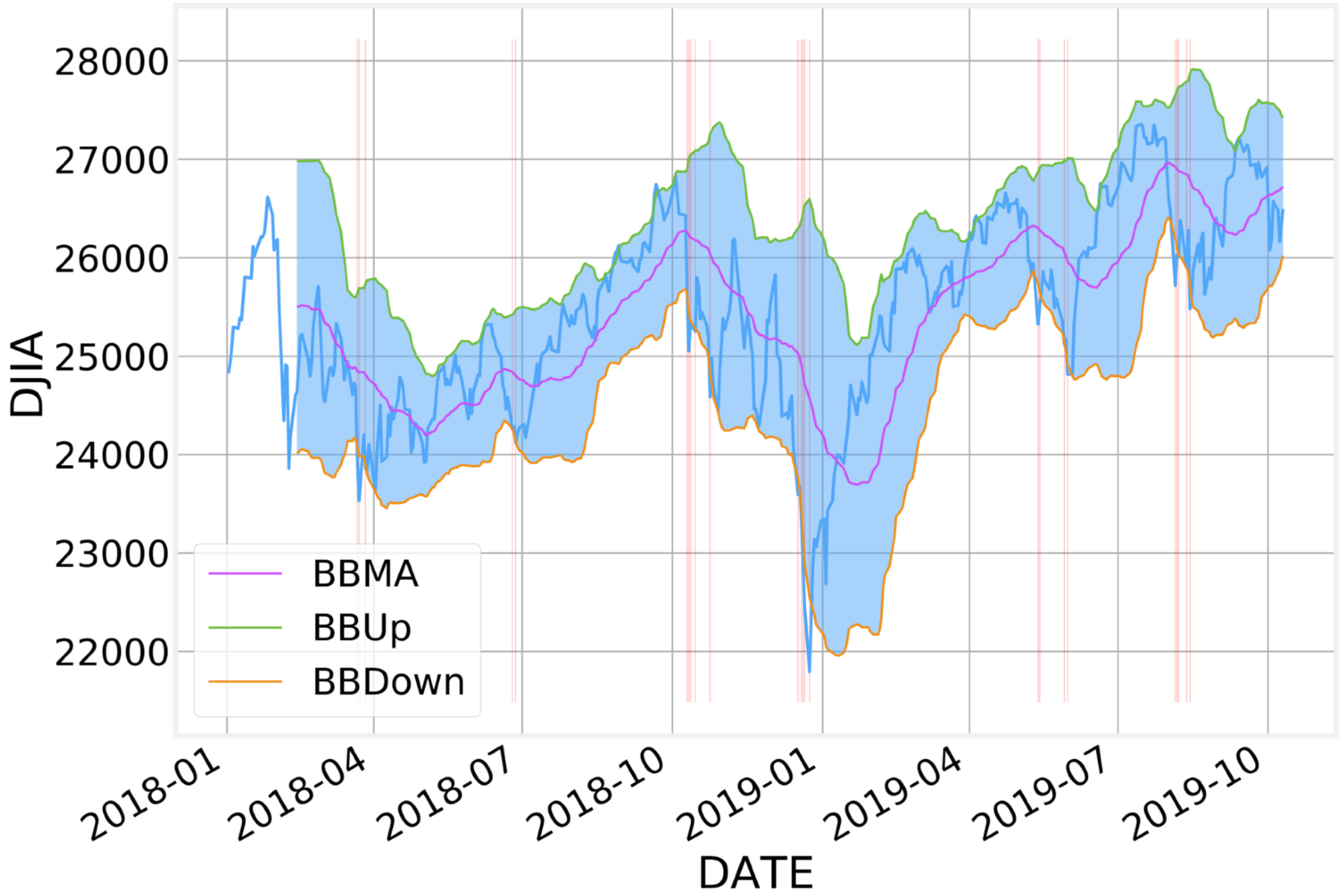

27000
26000
25000
24000
23000
22000
DJIA
daily
30 day average
30 day max
30 day min
Envelopes help characterize the typical behavior of the series
2018-01
2018-04
2018-07
2018-10
2019-01
2019-04
2019-07
2019-10
Date

Bollinger Bands

https://en.wikipedia.org/wiki/Bollinger_Bands

- A common use for application for running values is the calculation of Bollinger Bands.
- Introduced by John Bollinger in the 1980s as a complement to more traditional time series technical analysis techniques.
- Bollinger Bands are defined by two components:
  - A N period moving average, μ_N
    - The area K standard deviations above and below the moving average $\mu_N \pm K\sigma_N$
- Both μ_N and σ_N are computed on a running window of size N
- The values of N and K are application specific. For stock trading, $N = 20$ and $K = 2$ are typical values.
- Whenever the time series steps out of the Bollinger Band that's a clear indication of a change in the temporal behavior.

Bollinger Bands



Exponential Running Average

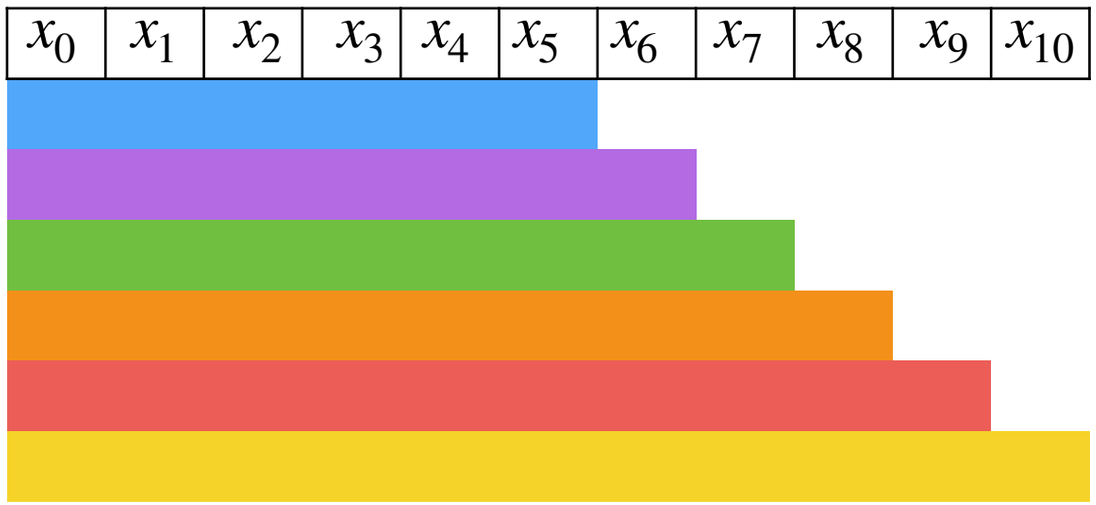
- One alternative to a simple running average is Exponential Smoothing

- The exponentially"smooth" version of a time series is given by:

$$z_t = \alpha x_t + (1 - \alpha) z_{t-1}$$

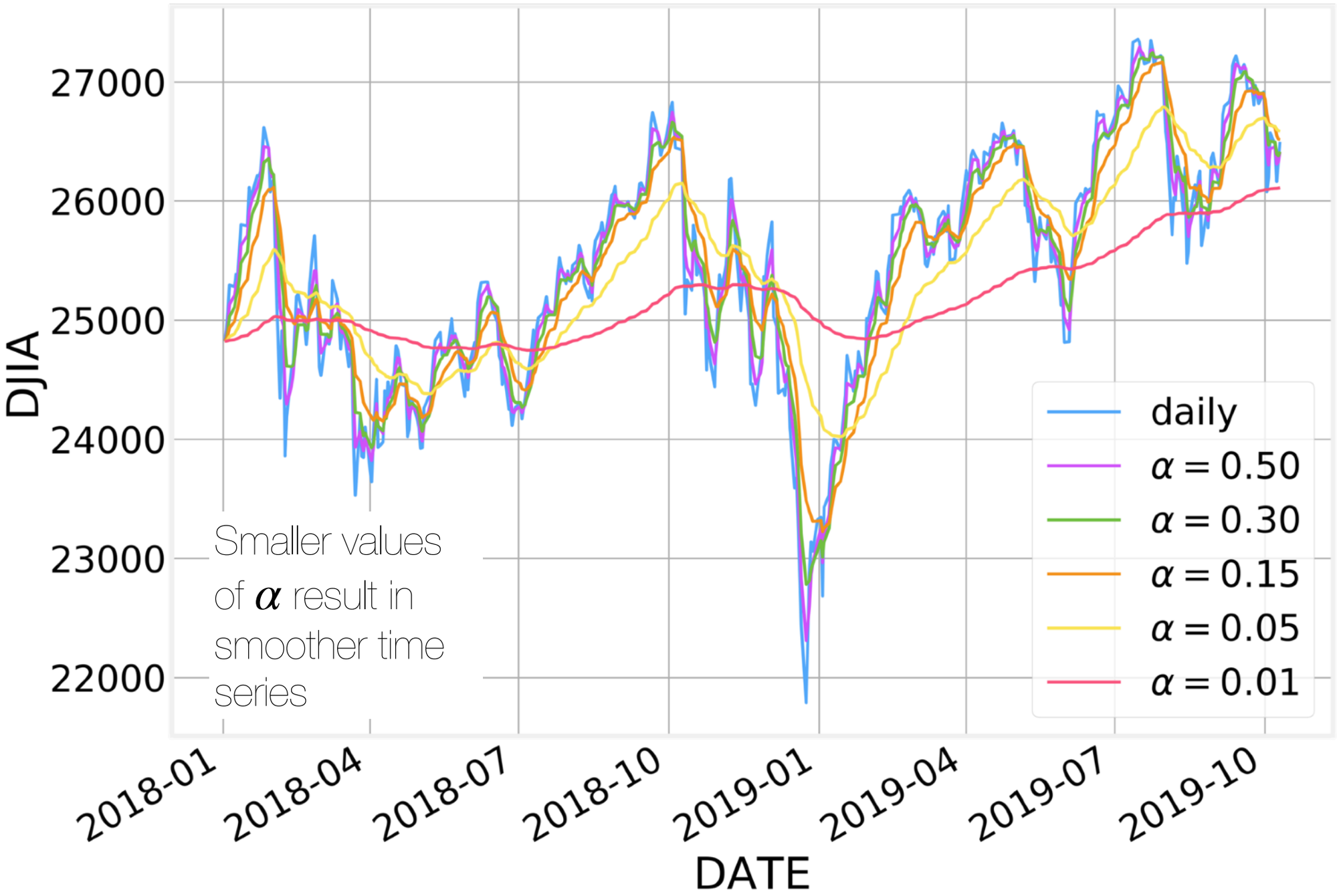
- The smaller the value of the weight α, the less influence each point has on the transformed time series.

- Each point depends implicitly on all previous points



$$\begin{pmatrix} \alpha & & & & \\ \alpha(1-\alpha)^1 & \alpha & & & \\ \alpha(1-\alpha)^2 & \alpha(1-\alpha)^1 & \alpha & & \\ \vdots & \vdots & \vdots & \ddots & \\ \alpha(1-\alpha)^{n-1} & \alpha(1-\alpha)^{n-2} & \alpha(1-\alpha)^{n-3} & & \alpha \end{pmatrix}$$

Exponential Running Average



Forecasting

- We can also use the Exponential Moving Averages as a simple forecasting tool.

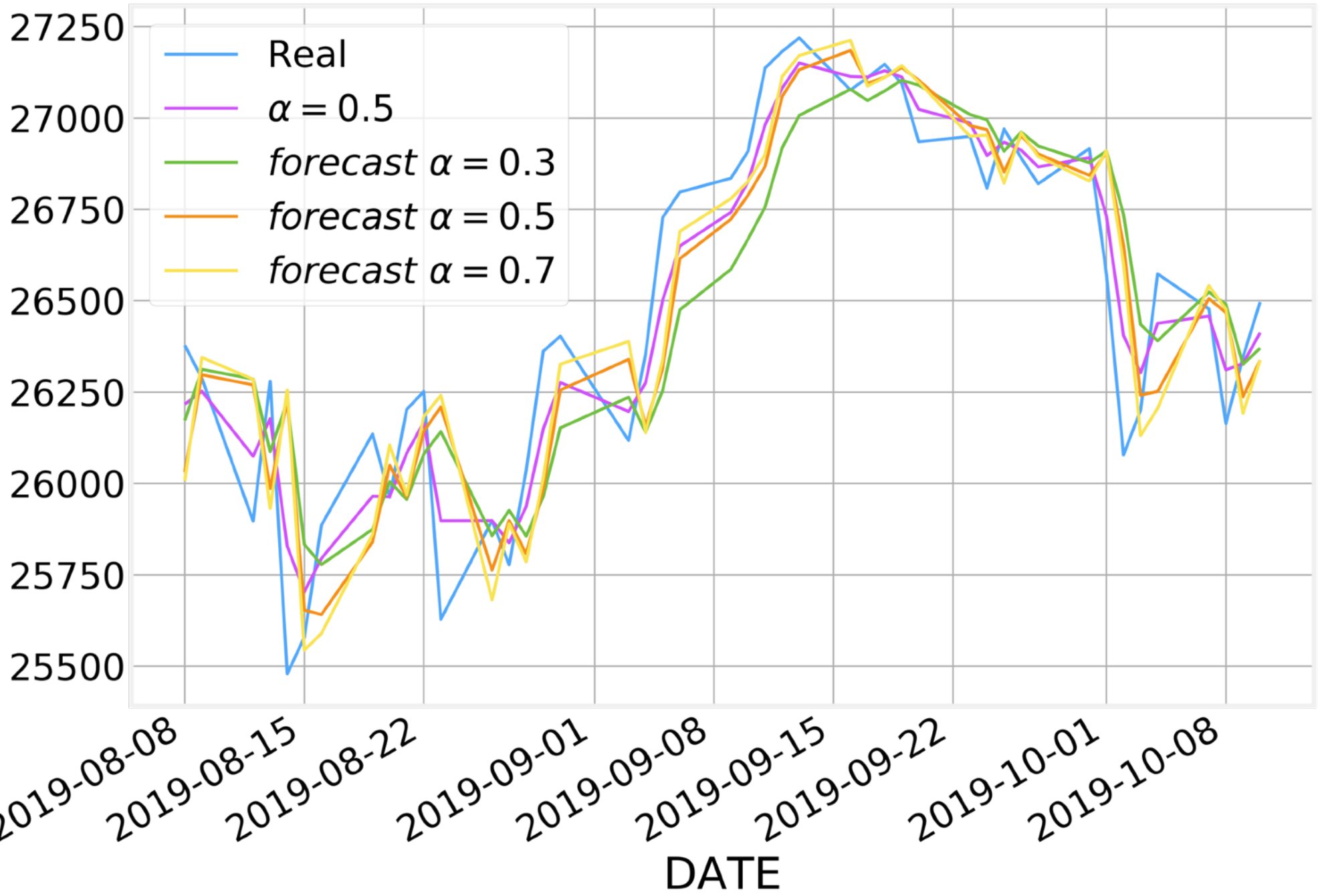
- The value at time $t+1$ is given by:

$$z_{t+1} = \alpha x_t + (1-\alpha) z_t$$

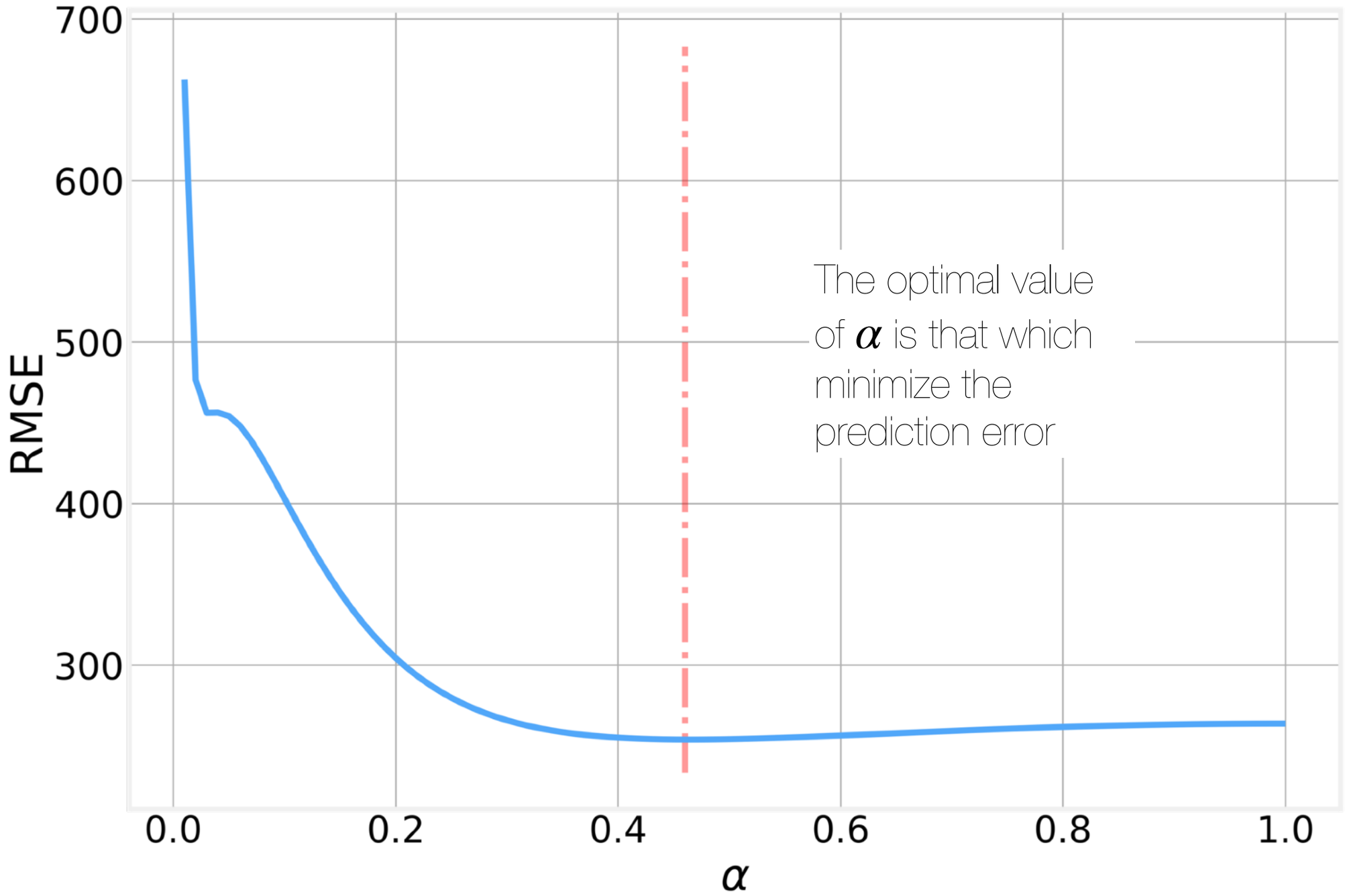
- Which we can consider to be a prediction on the value of x_{t+1}, based on the current value of z_t and some faction of our current error value $x_t - z_t$:

$$z_{t+1} = z_t + \alpha \left(x_t - z_t \right)$$

Forecasting



Forecasting



Code - Running Values

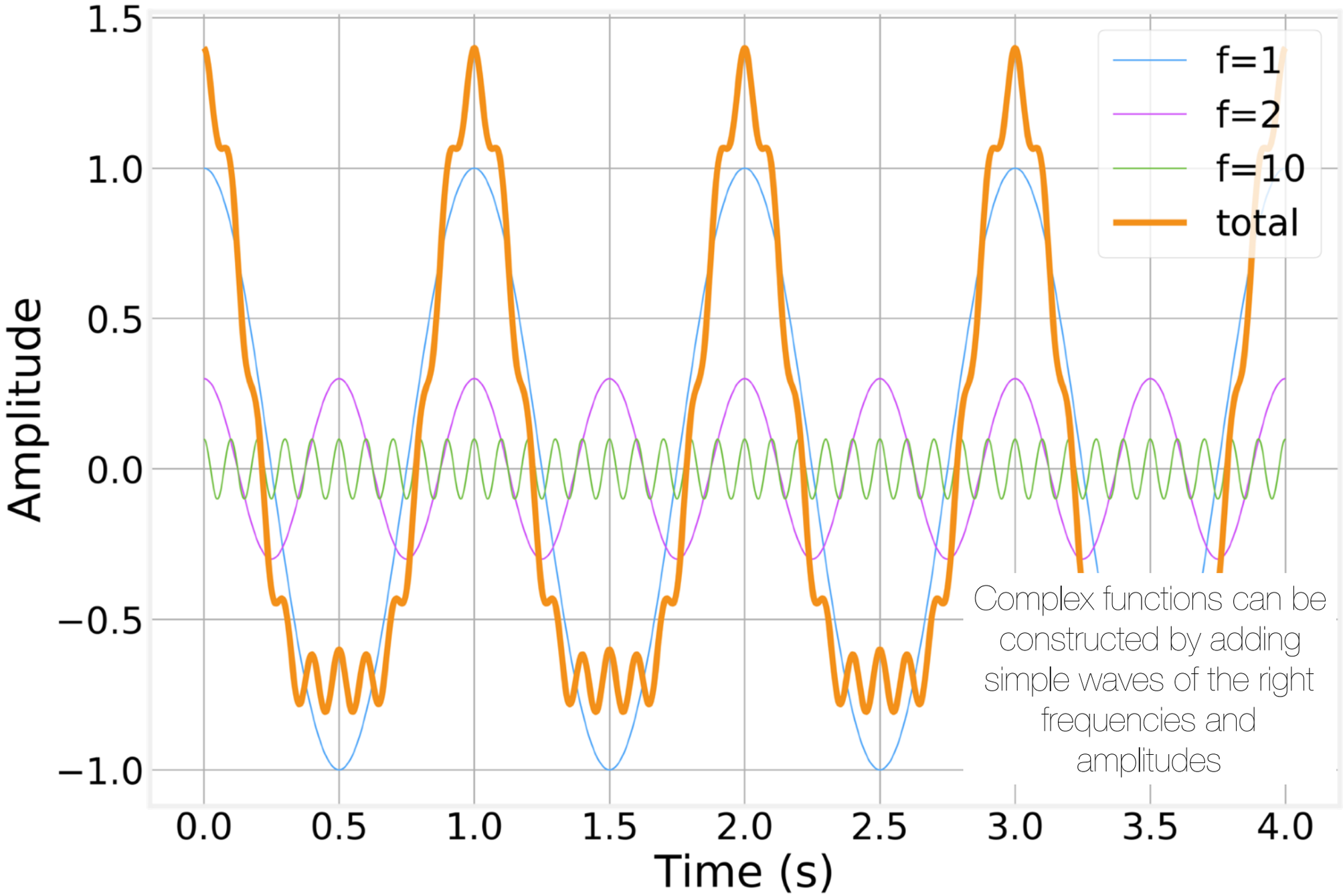
https://github.com/DataForScience/Timeseries_long

Fourier Analysis

Frequency Domain

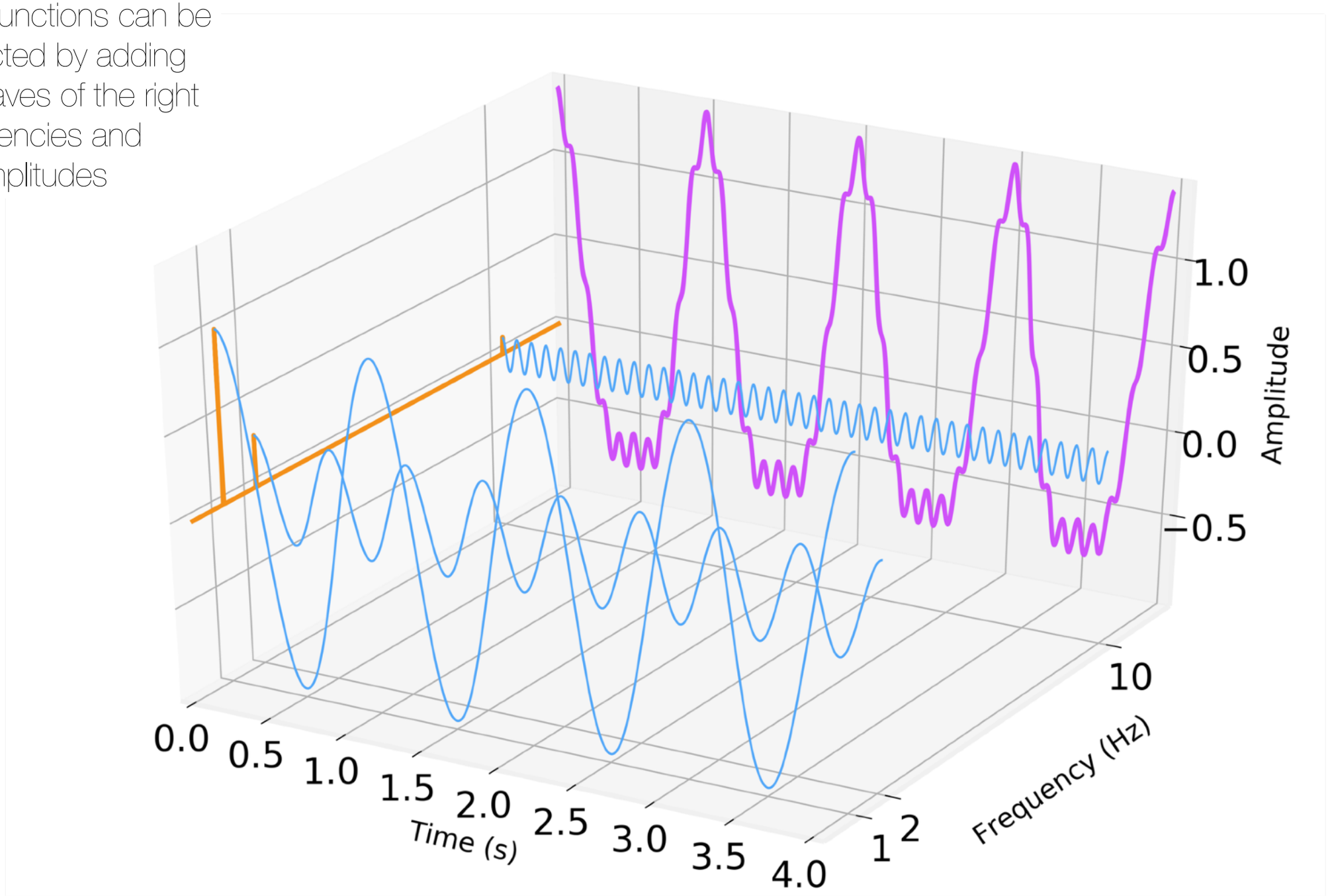
- So far we have focused on the natural (time based) representation of a time series.
- An alternative representation is based on frequencies and was first introduced by Jean Fourier in 1807
- Fourier showed that periodic functions can be decomposed as a sum of trigonometric functions
- Fourier's original result was later extended to all functions
- The Discrete Fourier Transforms provides us with an simple and convenient way to move from the time-domain to the frequency-domain and back.

Adding Frequencies



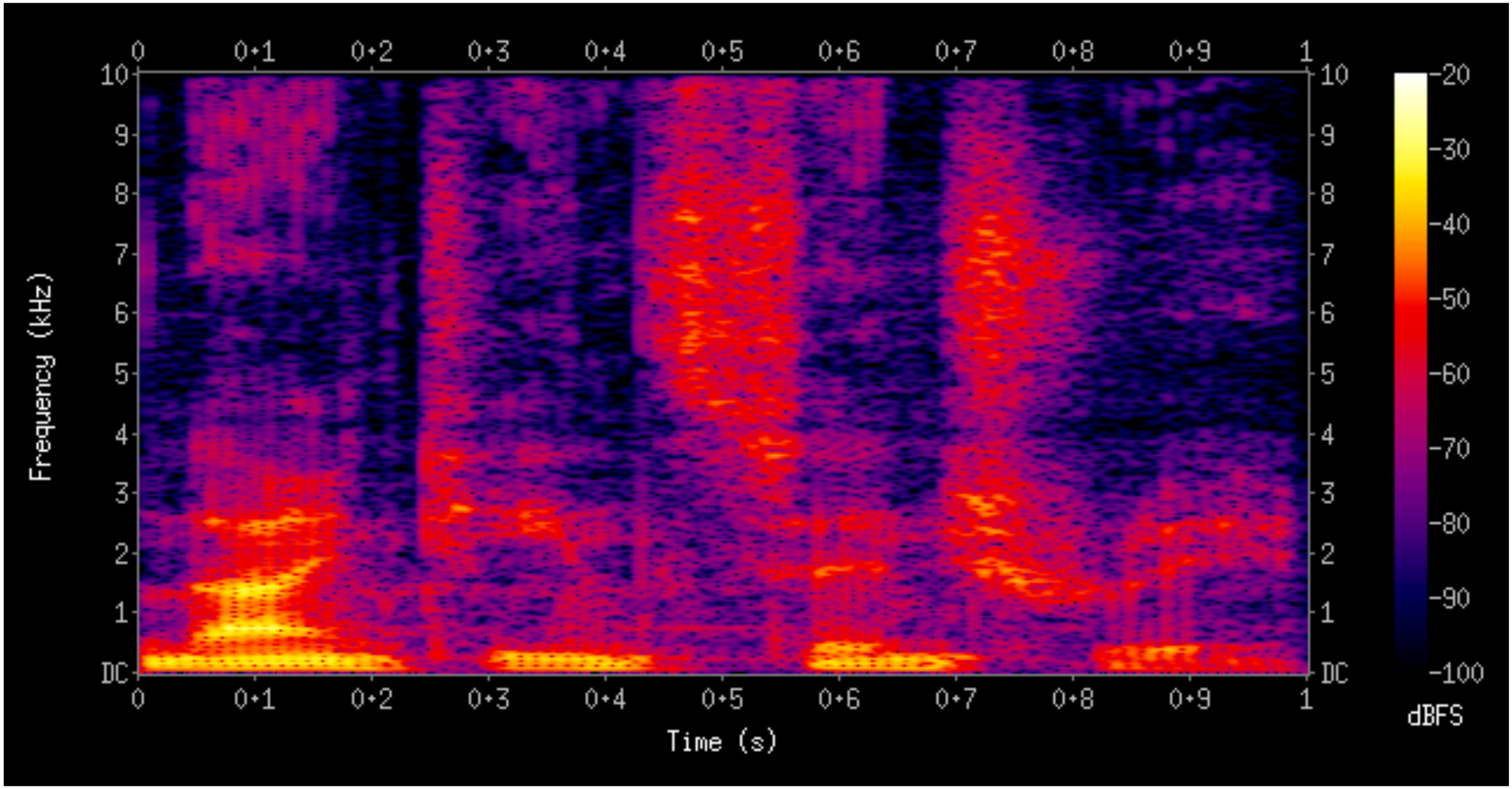
3D Visualization

Complex functions can be constructed by adding simple waves of the right frequencies and amplitudes



Spectrogram

https://en.wikipedia.org/wiki/Spectrogram



(Discrete) Fourier Transform

https://en.wikipedia.org/wiki/Discrete-time_Fourier_transform

- The DFT maps a sequence of N values x_n representing a time series $x(t)$ into N complex numbers X_k defined as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i \frac{2\pi k n}{N}}$$

- where:

$$e^{i\theta} = \cos(\theta) + i\sin(\theta)$$

- To recover the original values we use the Inverse DFT, defined as:

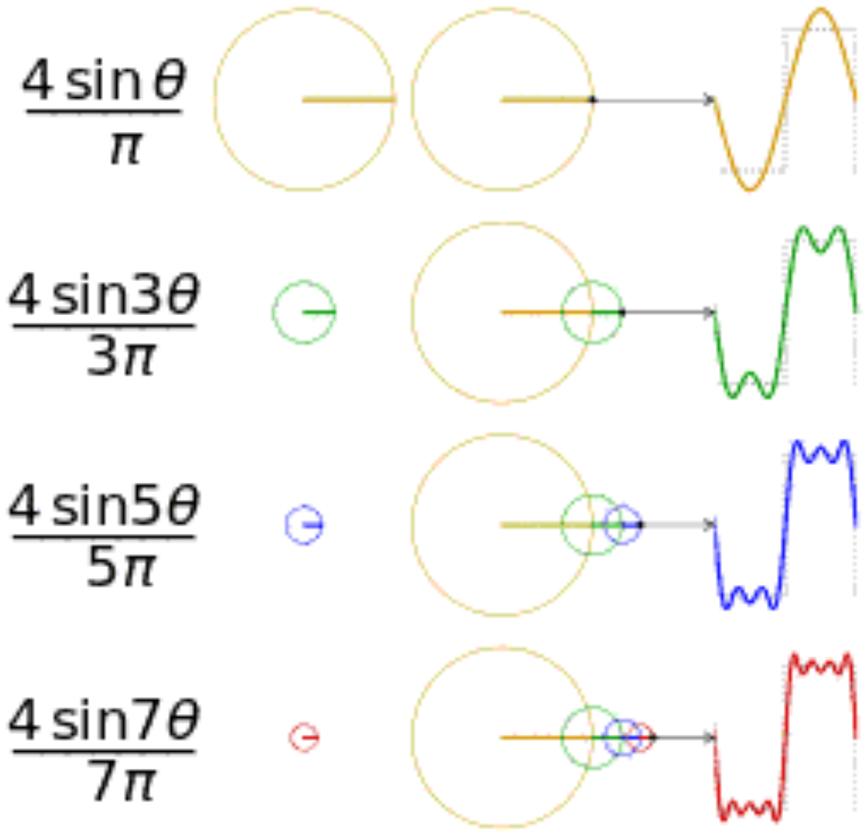
$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} x_k e^{i \frac{2\pi k n}{N}}$$

- The DFT represents the continuous series $x(t)$ as a sum of discrete frequencies:

$$\omega_n = 2\pi \frac{k}{N}$$

(Discrete) Fourier Transform

https://en.wikipedia.org/wiki/Discrete-time_Fourier_transform



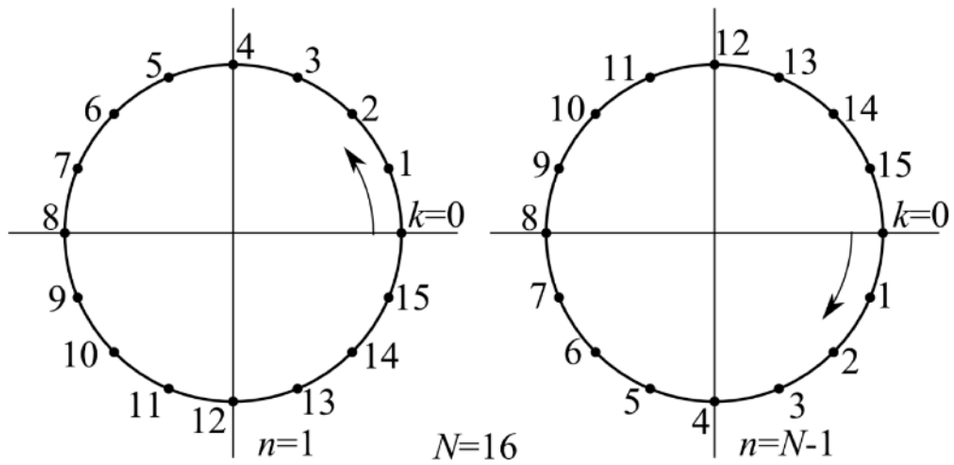
numpy

- Provides practical implementation of the Fast Fourier Transform an efficient algorithm to compute the DFT and IDFT. See https://en.wikipedia.org/wiki/Fast_Fourier_transform

- numpy.fft.fft()/numpy.fft.ifft() - DFT and IFT

- numpy.fft.fftfreq() - return the list of frequencies

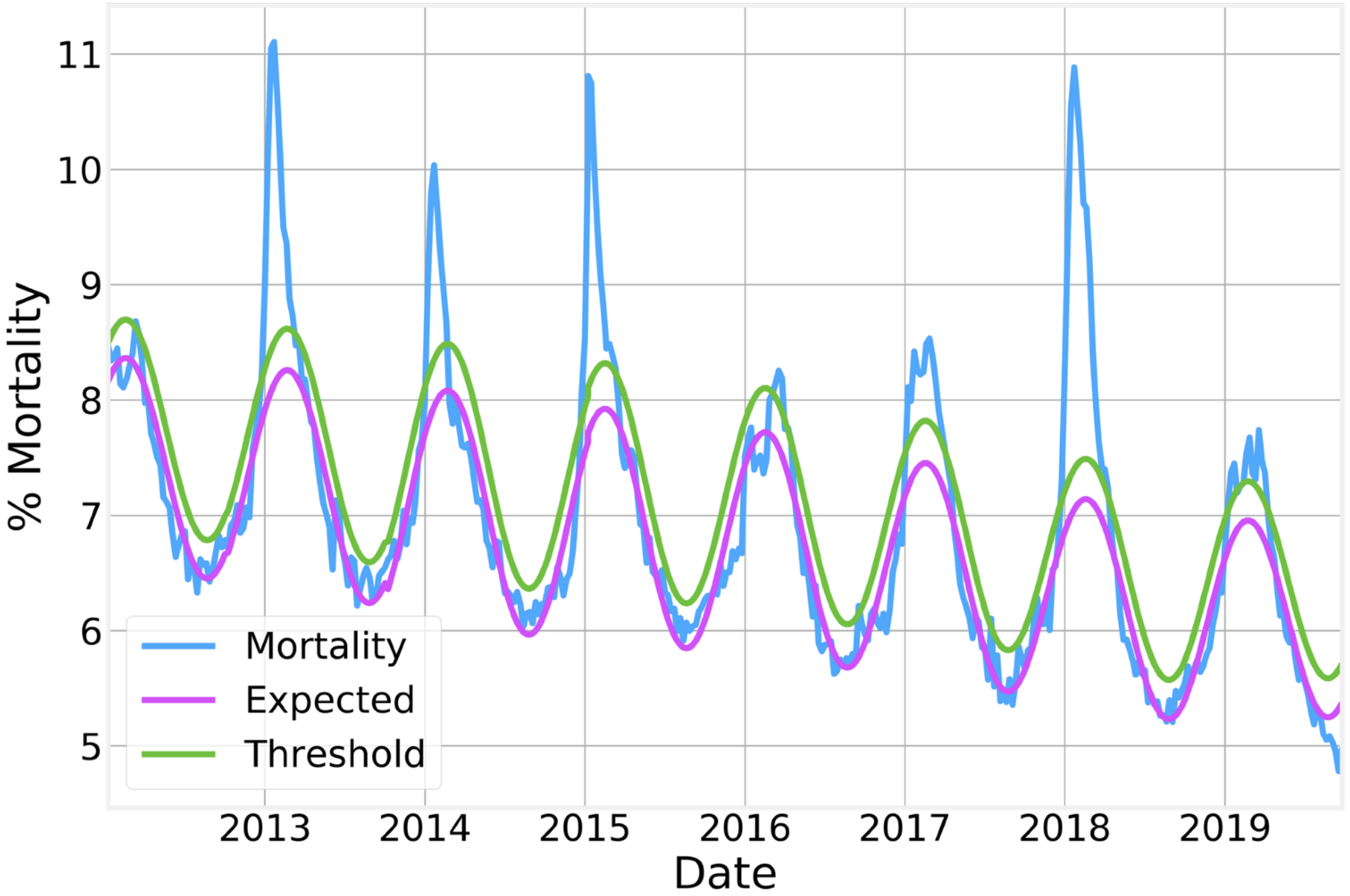
- numpy.fft.fftshift()/numpy.fft.ifftshift() - Shift the zero-frequency component to the center of the spectrum and back.



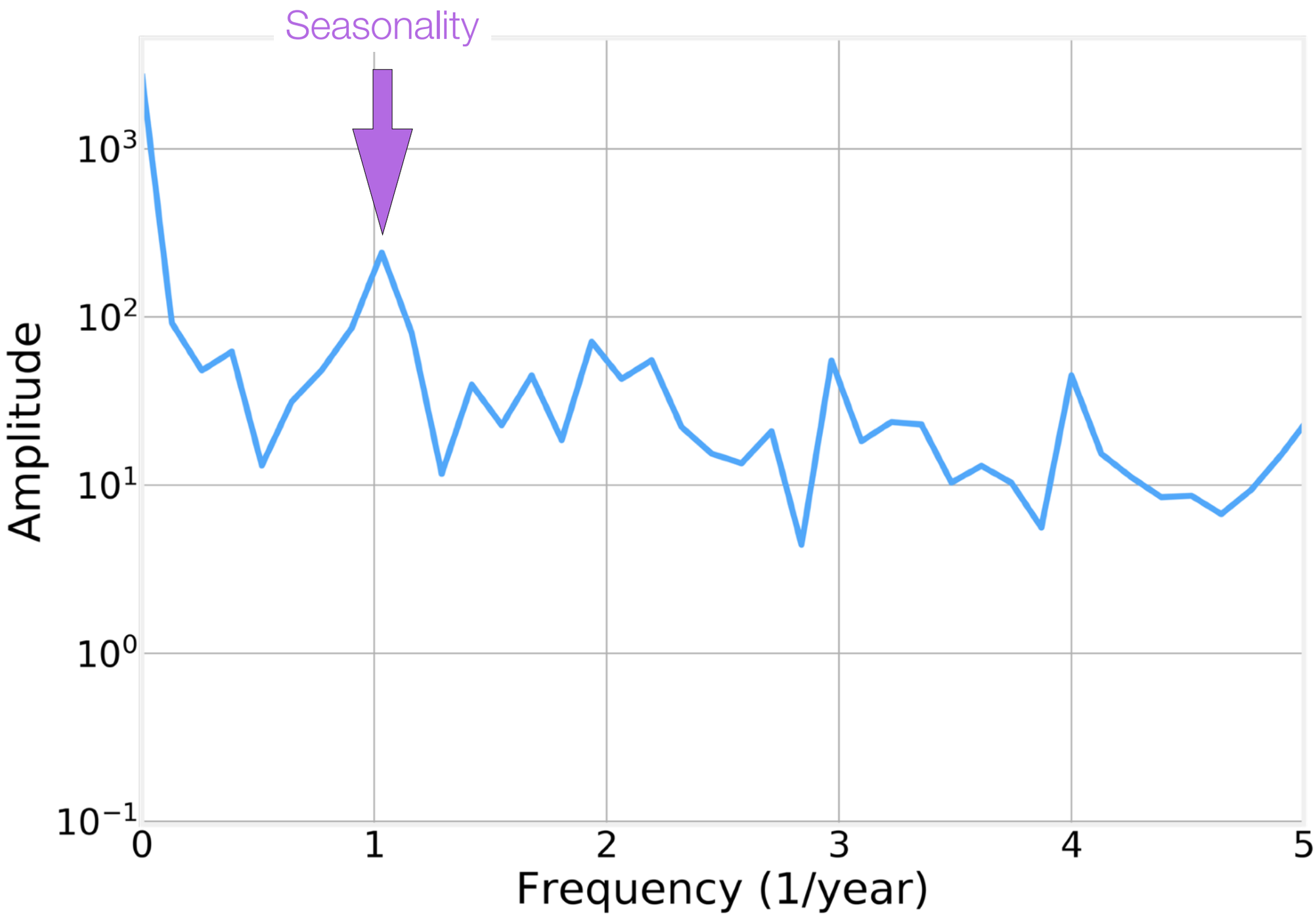
Filtering

- Common applications of Fourier Analysis are:
    - Seasonality - determine the main frequency underlying a time series
    - Filtering - remove higher order frequencies to eliminate noise
    - Processing - Several signal processing operations are simpler to compute in the frequency-space
    - Extrapolation/Forecasting - We can also extend the reconstructed signal to future values, with a couple of small caveats…

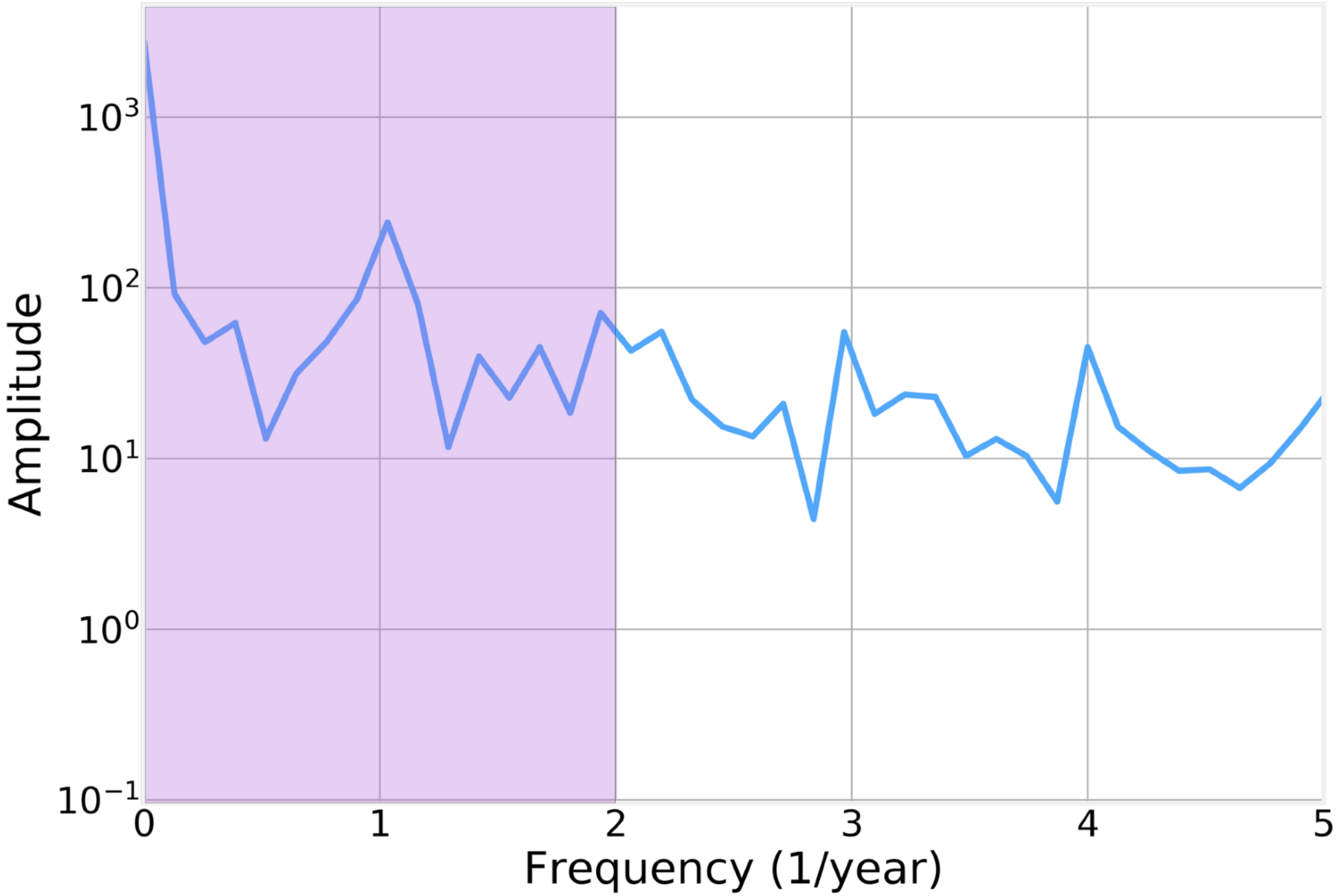
Filtering


% Mortality
11
10
9
8
7
6
5
Mortality
Expected
Threshold
2013
2014
2015
2016
2017
2018
2019
Date

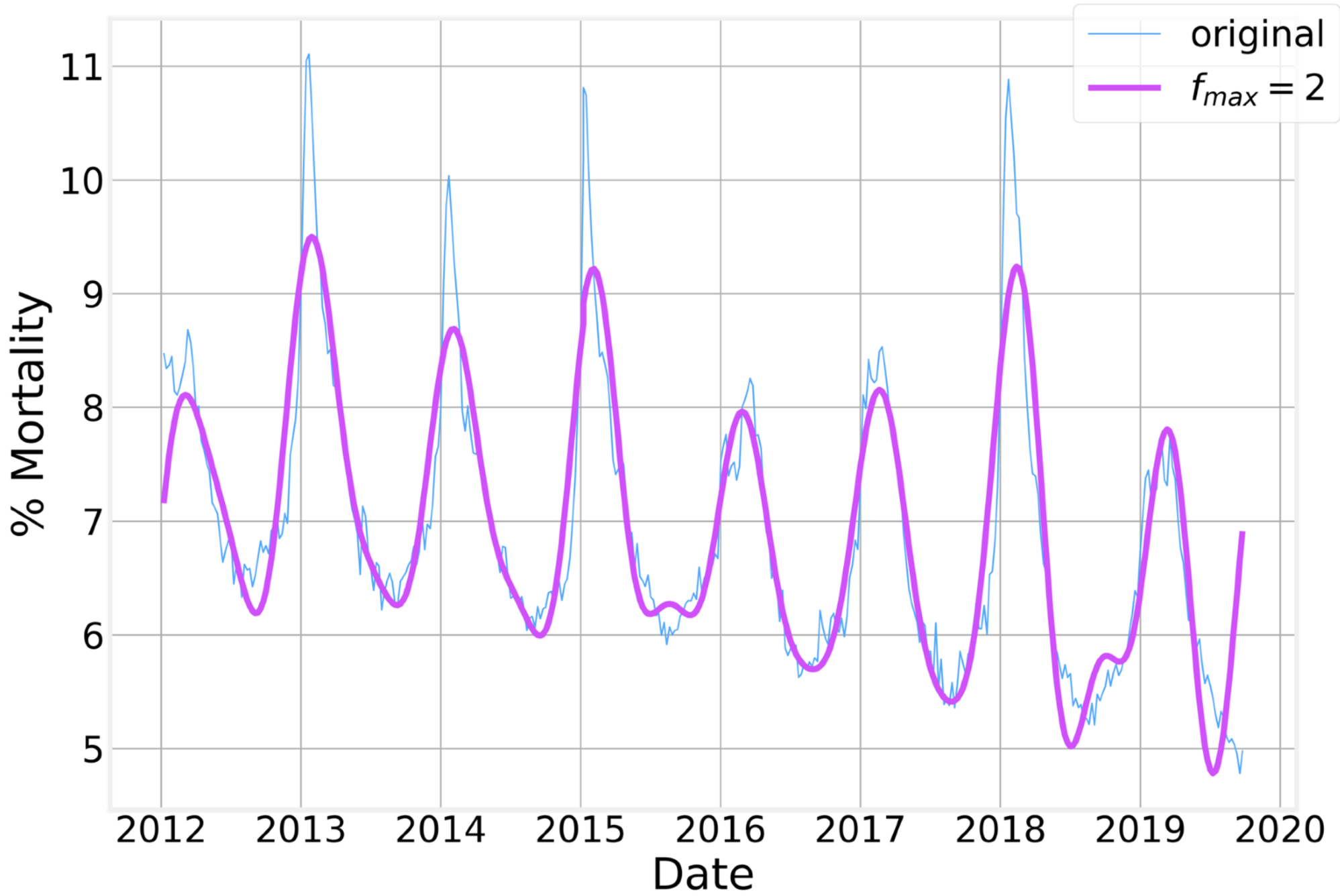
Filtering


Seasonality
10^3
10^2
10^1
10^0
10^{-1}
Amplitude
0
1
2
3
4
5
Frequency (1/year)

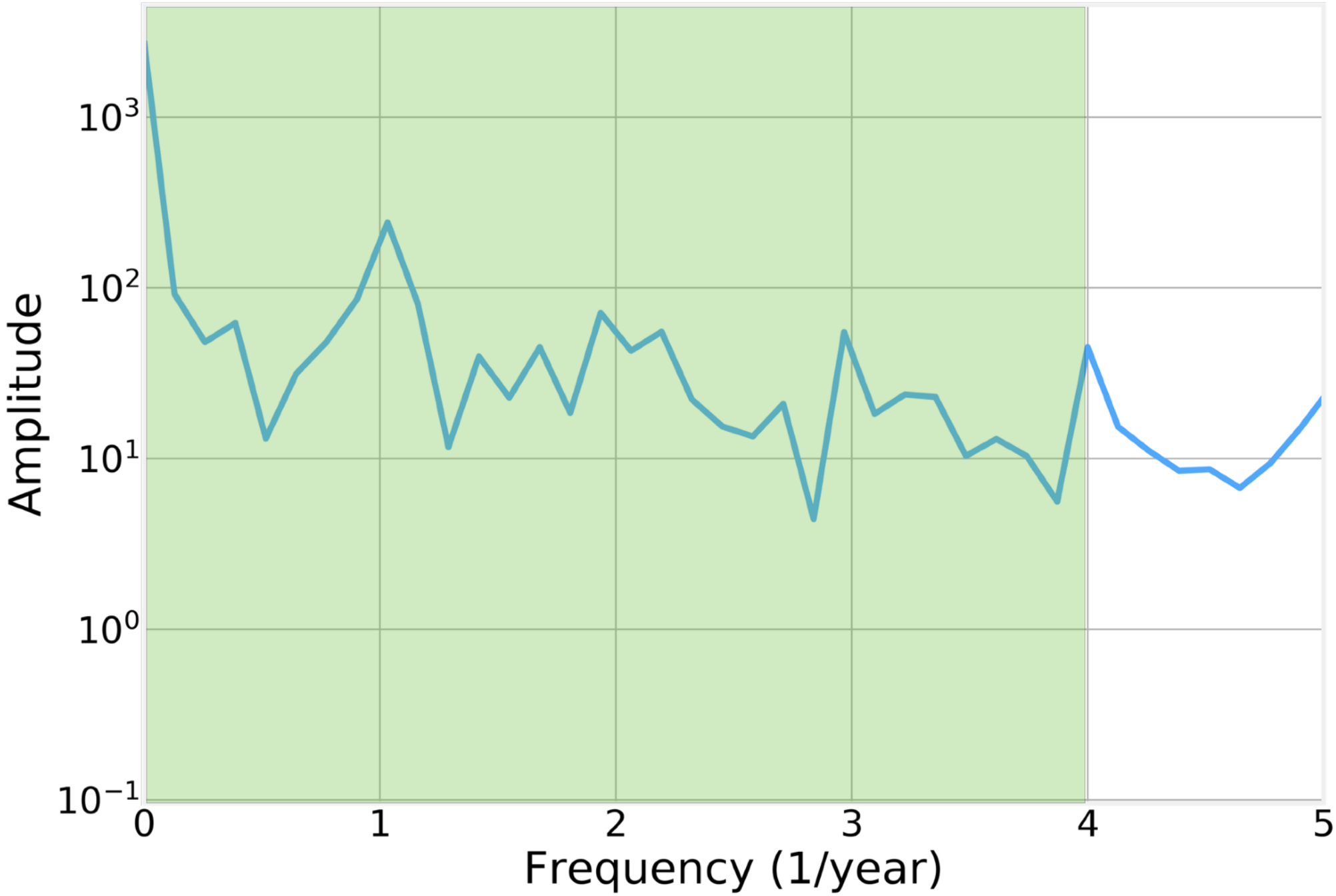
Filtering



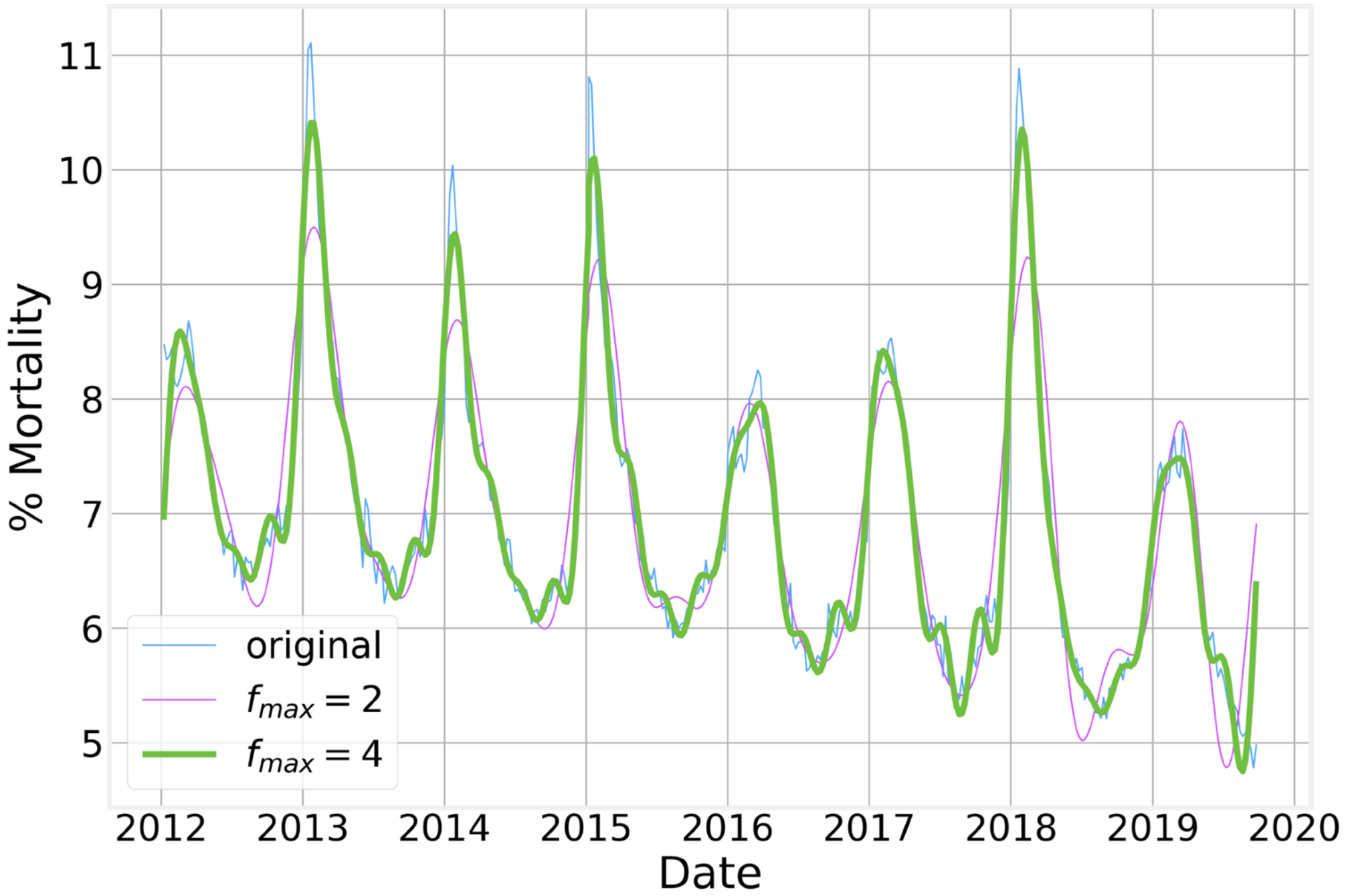
Filtering


original
$f_{max} = 2$
% Mortality
11
10
9
8
7
6
5
2012
2013
2014
2015
2016
2017
2018
2019
2020
Date

Filtering



Filtering


% Mortality
11
10
9
8
7
6
5
original
$f_{max} = 2$
$f_{max} = 4$
2012
2013
2014
2015
2016
2017
2018
2019
2020
Date

Extrapolation

- As we saw above, we can recover the original signal from the FFT values by using:

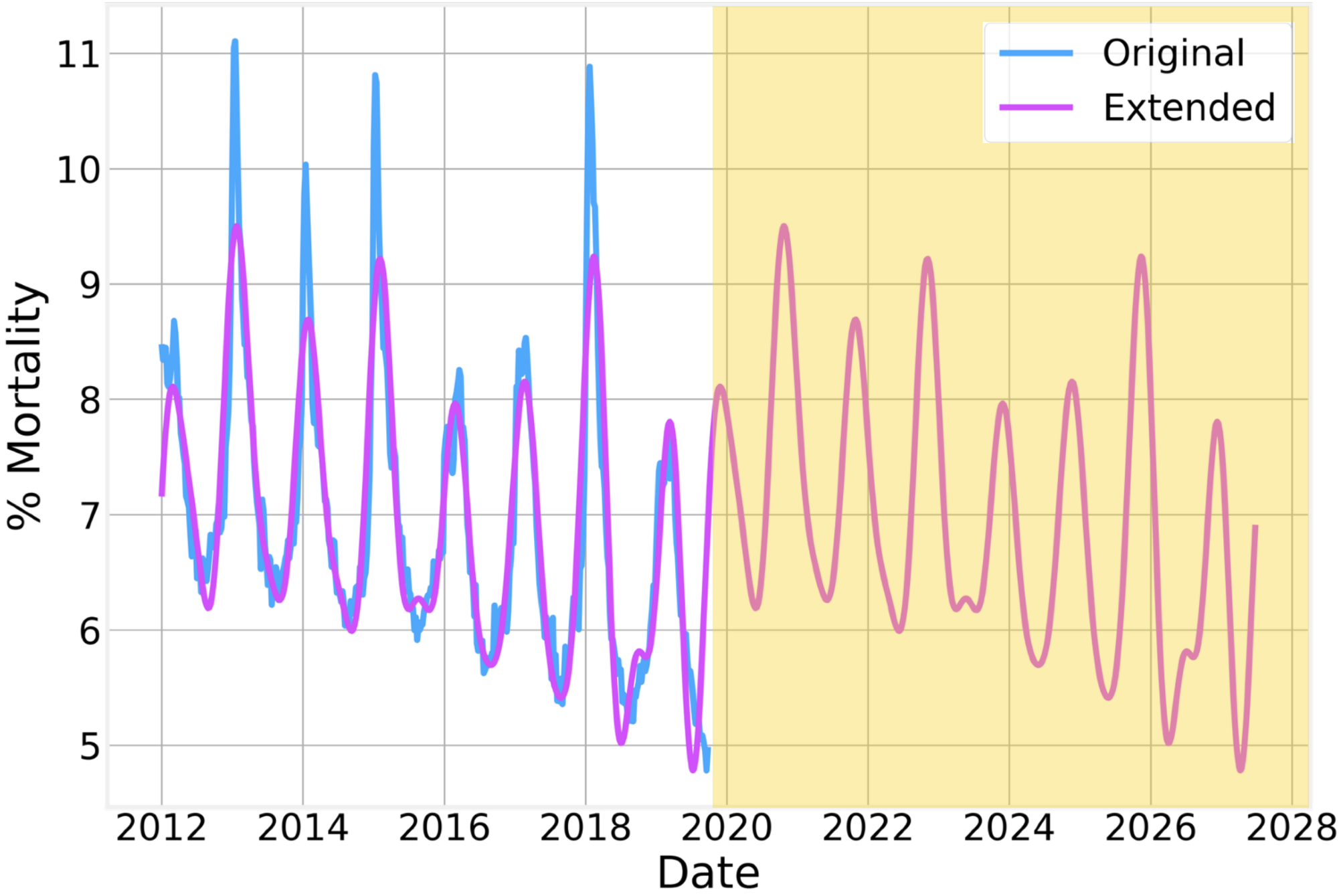
$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} x_k e^{i \frac{2\pi k n}{N}}$$

- Where n is our time variable.

- There's nothing stopping us from extending the values of n beyond the original domain of the signal

- The resulting extrapolated values correspond to a forecast into the future.

Extrapolation



Code - Fourier Analysis

https://github.com/DataForScience/Timeseries

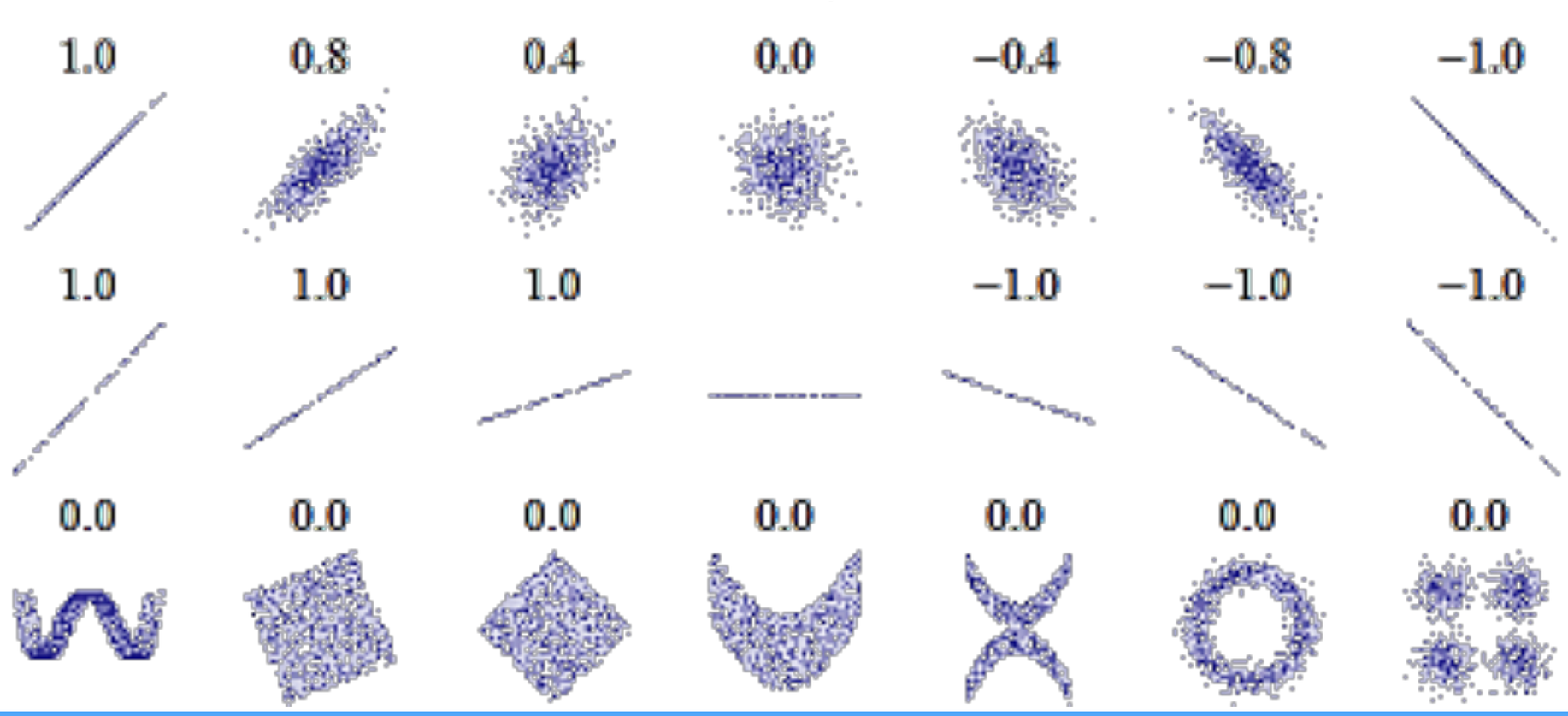
Correlations

Correlation

- Many correlation measures have been proposed over the years
- The most well known one is the Pearson Correlation

$$\rho\left(x,y\right) = \sum_{i=1}^{N} \frac{\left(x_i - \mu_x\right)\left(y_i - \mu_y\right)}{\sigma_x \sigma_y}$$

- Assumes a linear relationship between x and y.

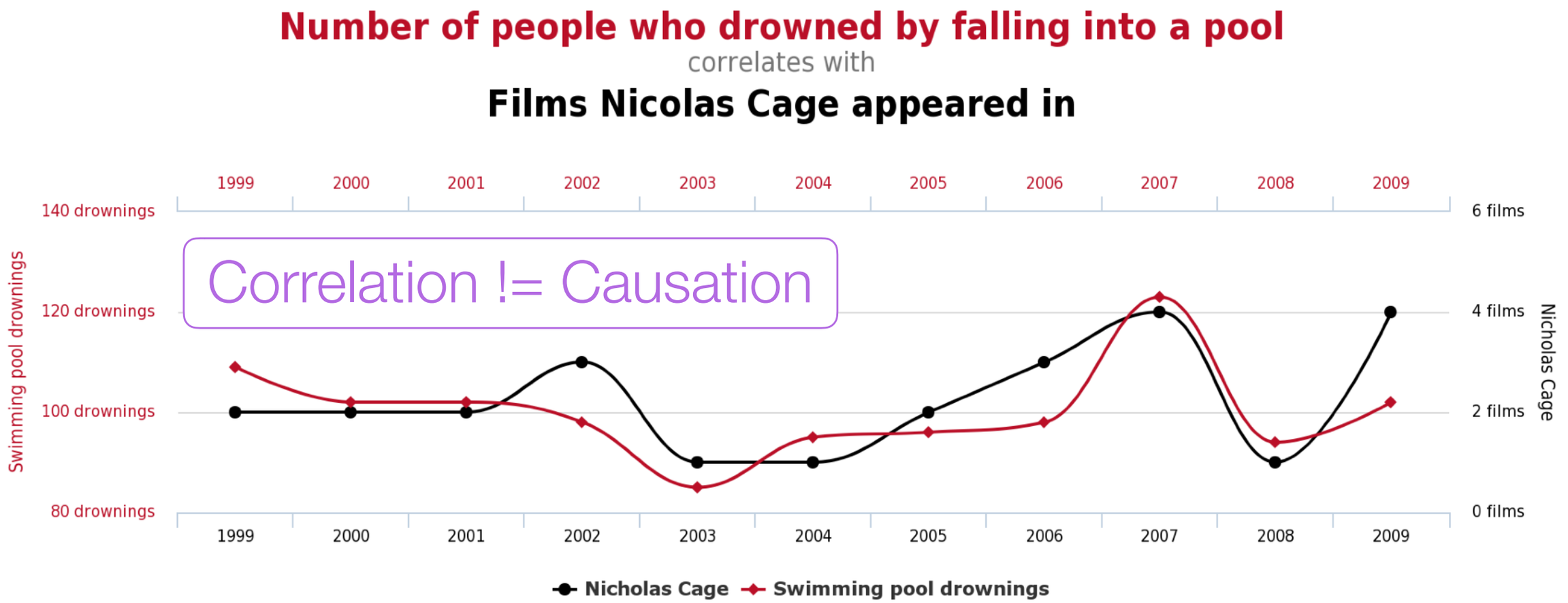


Correlations of 2 time series

- The correlation of two time series gives you an indication of how similar their behavior is
- Two completely unrelated time series (say, two sequences of random numbers) will have a Pearson correlation coefficient of 0

Correlations of 2 time series

- The correlation of two time series gives you an indication of how similar their behavior is
- Two completely unrelated time series (say, two sequences of random numbers) will have a Pearson correlation coefficient of 0



Correlations of 2 time series

- The correlation of two time series gives you an indication of how similar their behavior is
- Two completely unrelated time series (say, two sequences of random numbers) will have a Pearson correlation coefficient of 0
- Correlation != Causation!
- Adding a trend to both series we immediately observe a significant correlation

The Pearson correlation of two trending series is overwhelmed by the trend

Auto-correlation

https://en.wikipedia.org/wiki/Correlogram

- It follows from the previous slide that a series will have a perfect correlation with itself, but what about lagged versions of itself?

- We define the Auto-correlation function as the Pearson correlation between values of the time series at different lags, as a function of the lag:

$$ACF_x\left(l\right) = \rho\left(x_t, x_{t-l}\right)$$

- By definition, $ACF_x(0) \equiv 1$

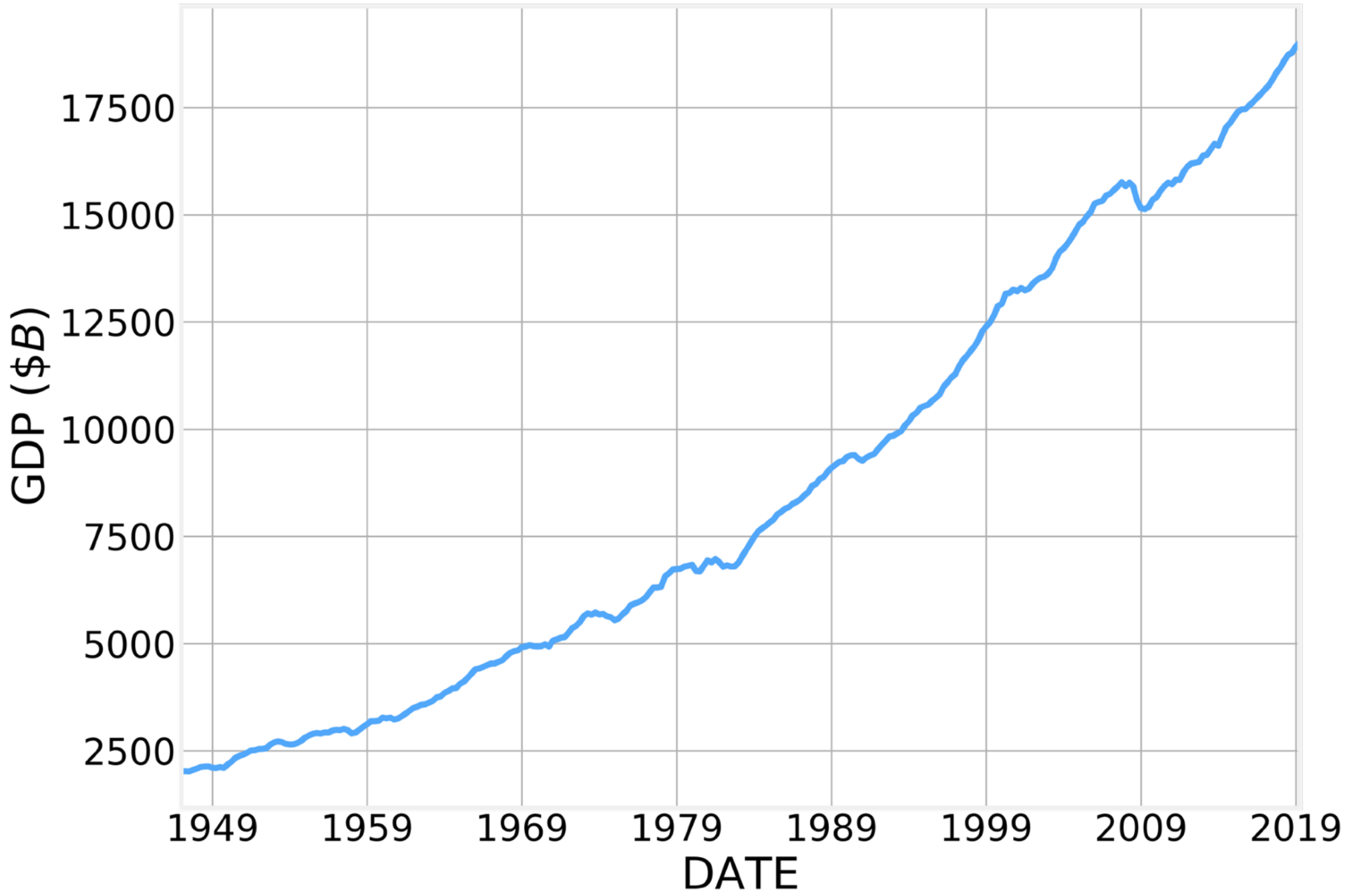
- And as the lag l increases the value of the ACF tends to decrease.

- We can calculate the confidence interval for the ACF using:

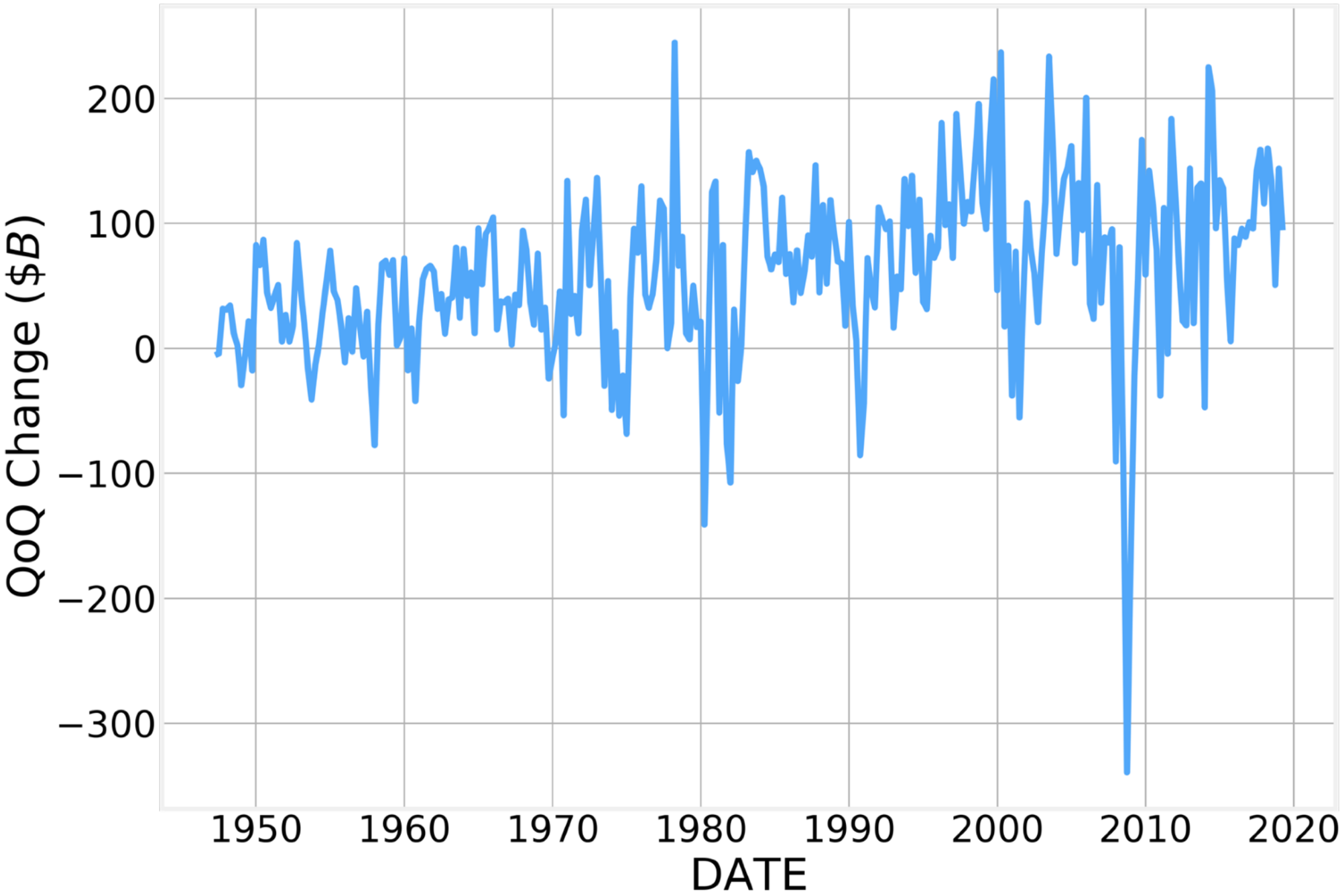
$$CI = \pm z_{1-\alpha/2}\sqrt{\frac{1}{N}\left(1 + 2\sum_{l=1}^{k} r_l^2\right)}$$

- where $z_{1-\alpha/2}$ is the quantile of the normal distribution corresponding to significance level α and r_l are the values of the ACF for a specific lag l

Auto-correlation

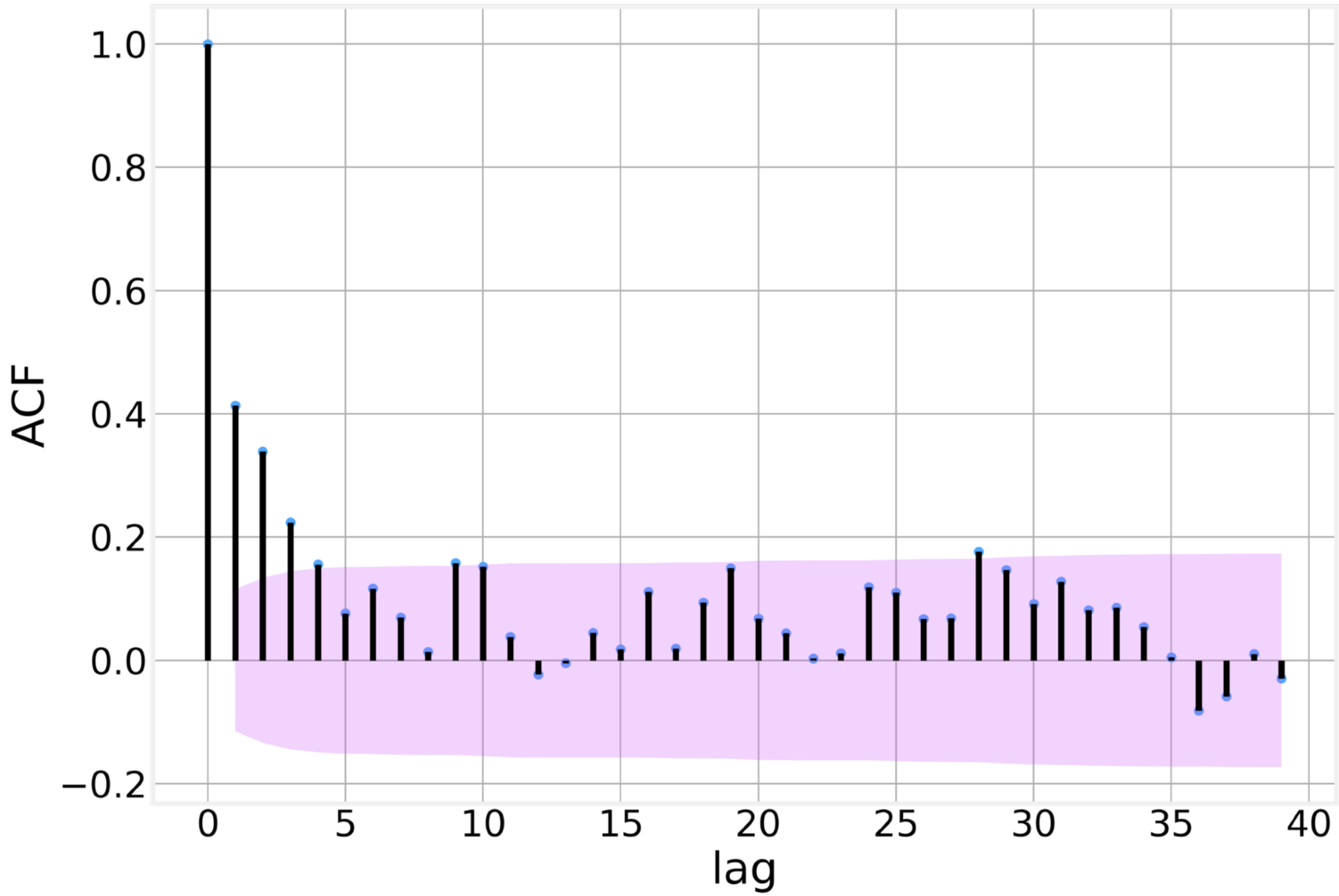


Auto-correlation



Auto-correlation

https://en.wikipedia.org/wiki/Correlogram



Partial Autocorrelation

https://en.wikipedia.org/wiki/Partial_autocorrelation_function

- One of the disadvantages of the Autocorrelation function is that it still considers the intermediate values

- The Partial Autocorrelation function calculate the correlation function between x_t and x_{t-l} after explaining away all the intermediate values $x_{t-1} \cdots x_{t-l+1}$

- Intermediate values are "explained away" by fitting a linear model of the form:

$$\hat{x}_t = f\left(x_{t-1} \cdots x_{t-l+1}\right)$$

$$\hat{x}_{t-l} = f\left(x_{t-1} \cdots x_{t-l+1}\right)$$

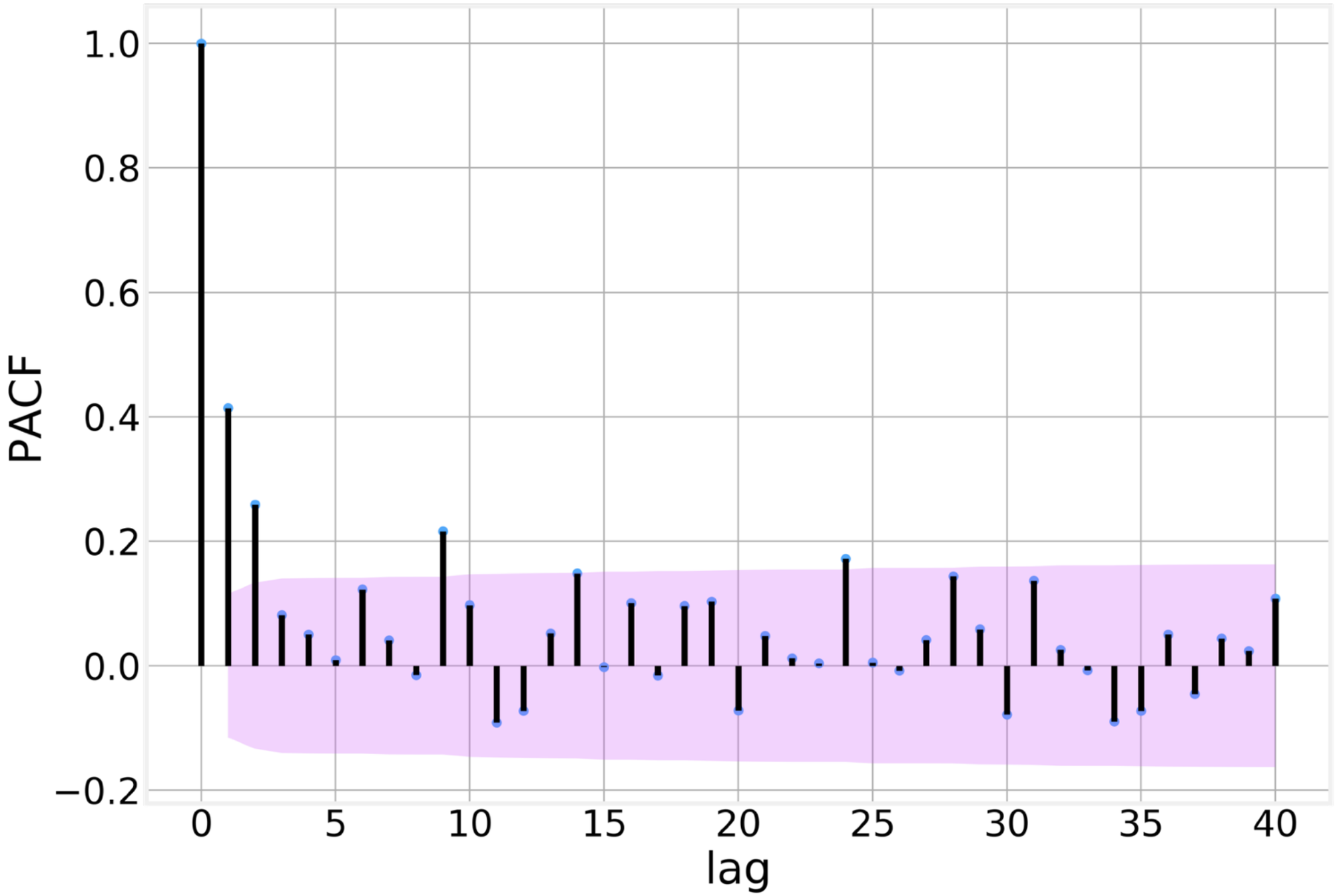
- And then calculating the Pearson correlation function between the values and their residuals:

$$PACF_x\left(l\right) = \rho\left(x_t - \hat{x}_t, x_{t-l} - \hat{x}_{t-l}\right)$$

- Confidence intervals can be computed using the same formula used for the ACF

Partial Autocorrelation

https://en.wikipedia.org/wiki/Partial_autocorrelation_function



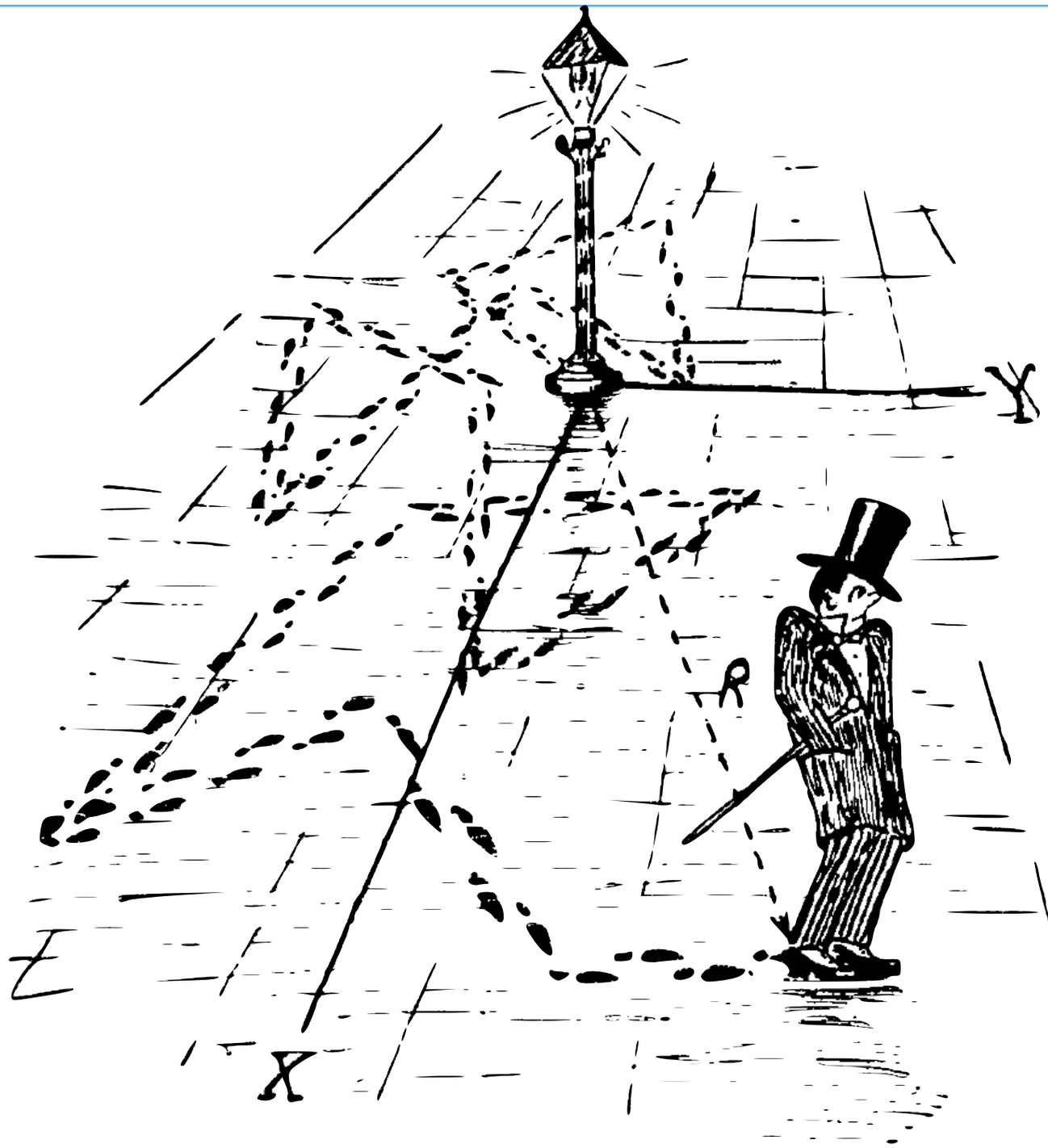
Code - Correlations

https://github.com/DataForScience/Timeseries_long

Random Walks

Random Walks

Illustration by George Gamow



- At each step flip a coin
  - Heads: Move right
  - Tails: Move left
- If you start at position 0, do you ever reach position L?
- On average, we expect the position to be always close to 0.
- What if the coin is biased as in the previous example?

Random Walks

- Mathematically, we can describe the position of our random walker at time t as:

$$x_t = x_{t-1} + \epsilon_t$$

- Where ϵ_t is the stochastic value generated by our coin flip ($+1$ or -1)
- We can further write:

$$x_t = x_0 + \sum_i \epsilon_i$$

- which shows that the current position is just the sum across all coin flips in our walk.
- Naturally, we can treat a random walk as a realization of a time series, but is it stationary?
- The mean position is:

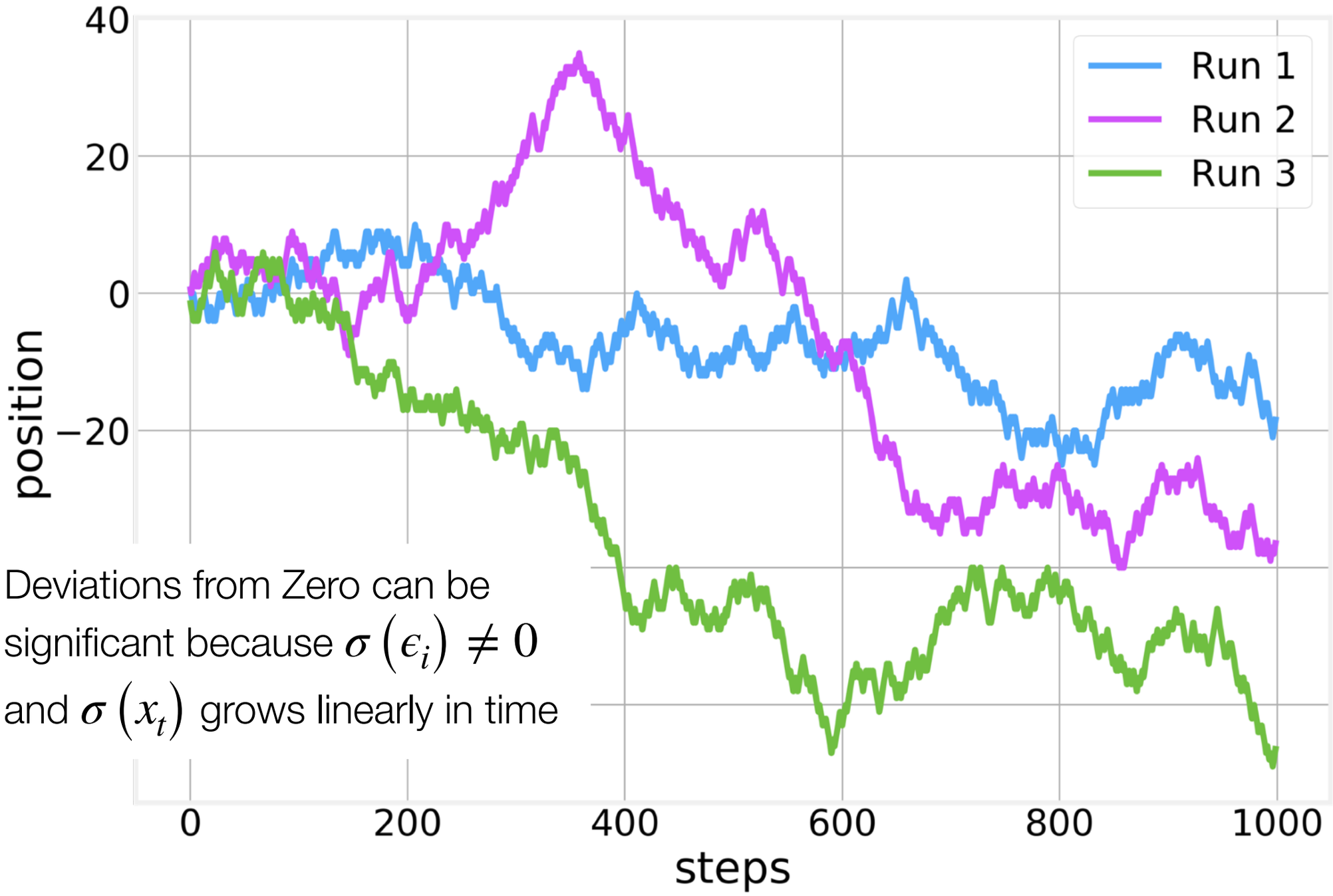
$$\mu = \langle x_t \rangle = \langle x_0 \rangle + \sum_i \langle \epsilon_i \rangle$$

- If the coin is unbiased, $\langle \epsilon_i \rangle = 0$ and the mean is constant. On the other hand, the variance is:

$$\sigma = \sigma\left(x_t\right) = \sigma\left(x_0\right) + \sum_i \sigma\left(\epsilon_i\right) = \sigma\left(x_0\right) + t \cdot \sigma\left(\epsilon\right)$$

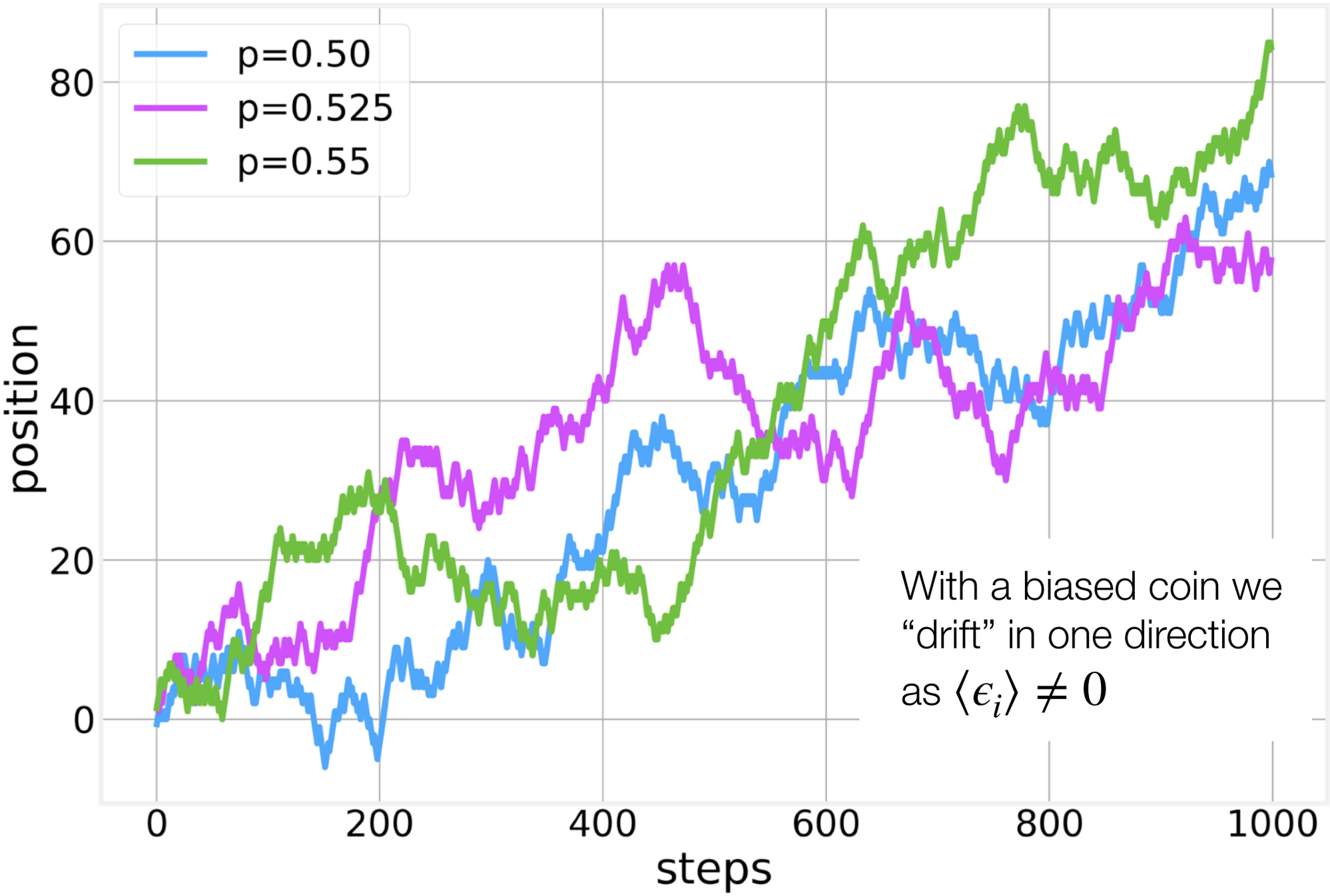
- which is not constant. So even the simple random walk is not a stationary process.

Random Walks



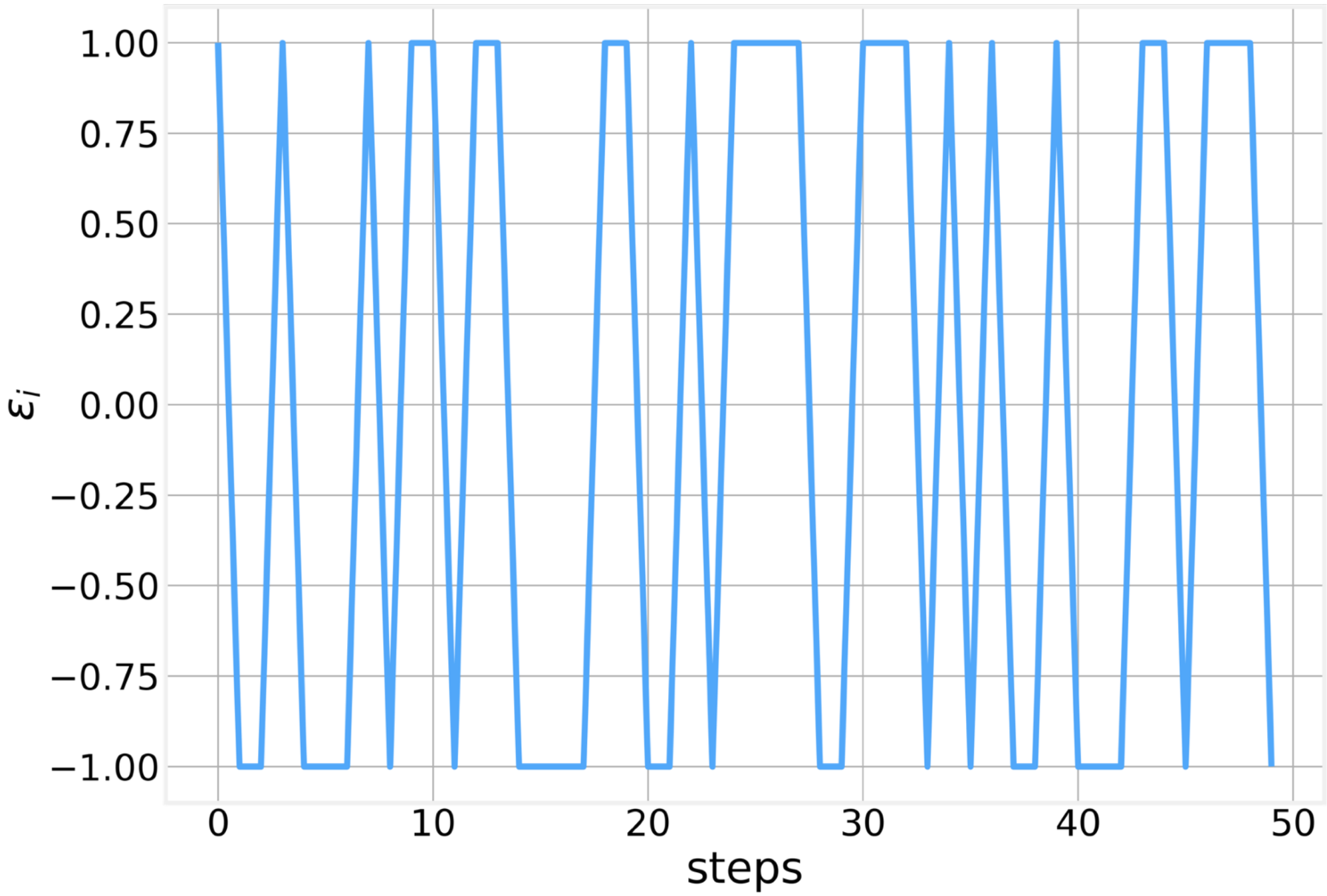
Deviations from Zero can be significant because $\sigma\left(\epsilon_i\right) \neq 0$ and $\sigma\left(x_t\right)$ grows linearly in time

Random Walk with a drift



White noise

Let's take a deep look at our stochastic variables (the outcomes of the “coin flips”)



Stationary vs Non-Stationary

- We can also describe this process in the same mathematical framework as:

$$x_t = \epsilon_i$$

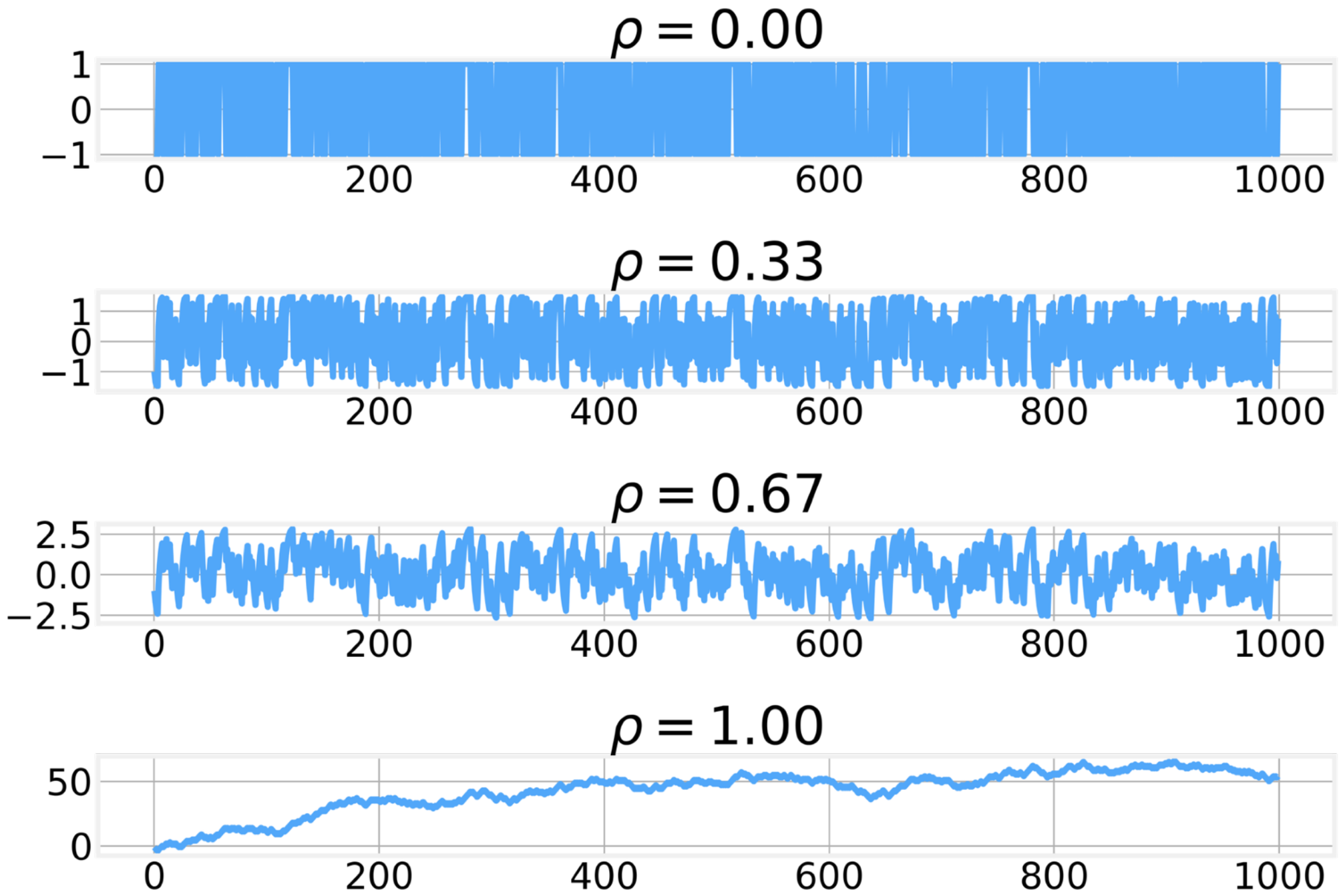
- which is clearly a stationary process.

- We can combine both expressions into a single one:

$$x_t = \rho x_{t-1} + \epsilon_i$$

- Where ρ gives us a "knob" to interpolate between the two extremes, between a stationary and a non-stationary process.

Stationary vs Non-Stationary



Dickey-Fuller Test

- The Dickey-Fuller Test is a test of stationarity inspired by a simple: Can we show that

$$\rho \neq 1$$

- with some degree of certainty?

- Numerically, we can express this as a linear regression fit

$$x_t - x_{t-1} = \gamma \; x_{t-1} + \epsilon_t$$

- where $\gamma = \rho - 1$

- The slope of the regression is then our expected value for $\rho - 1$.

- If the process is non-stationary then we expect $\gamma \neq 0$.

Dickey-Fuller Test

- From the residuals of γ we compute the Dickey-Fuller statistic:

$$DF = \frac{\hat{\gamma}}{SE\left(\gamma\right)}$$

- The value of this statistic is then compared with a critical values table.

- In general, the more negative it is, the more certain we can be that we can reject the null hypothesis

- The Dickey-Fuller Test has many variants.

- The most common one is the known as the Augmented-Dickey-Fuller test and is able to account for multiple lags, trends, etc.

Critical values for Dickey–Fuller *t*-distribution.

	Without trend		With trend	
Sample size	1%	5%	1%	5%
T = 25	–3.75	–3.00	–4.38	–3.60
T = 50	–3.58	–2.93	–4.15	–3.50
T = 100	–3.51	–2.89	–4.04	–3.45
T = 250	–3.46	–2.88	–3.99	–3.43
T = 500	–3.44	–2.87	–3.98	–3.42
T = ∞	–3.43	–2.86	–3.96	–3.41

Source[2]:373

Hurst Exponent

- The Hurst Exponent is another metric that allows us to determine wether or not a time series is stationary.

- It measures the "speed of diffusion" defined as:

$$Var(\tau) = \langle \left| z(t+\tau) - z(t) \right|^2 \rangle \sim \tau^{2H}$$

- where H is the Hurst exponent
  - $H < 0.5$ - mean reverting series
  - $H = 0.5$ - geometric random walk
  - $H > 0.5$ - trending series
- Smaller values indicate stronger levels of mean reversion, while larger values represent stronger trends

Code - Random Walks

https://github.com/DataForScience/Timeseries_long

ARIMA Models

Moving Average (MA) Model

- We start our exploration of the ARIMA family of models by considering the Moving Average model.
- The simplest moving average model can be written as:

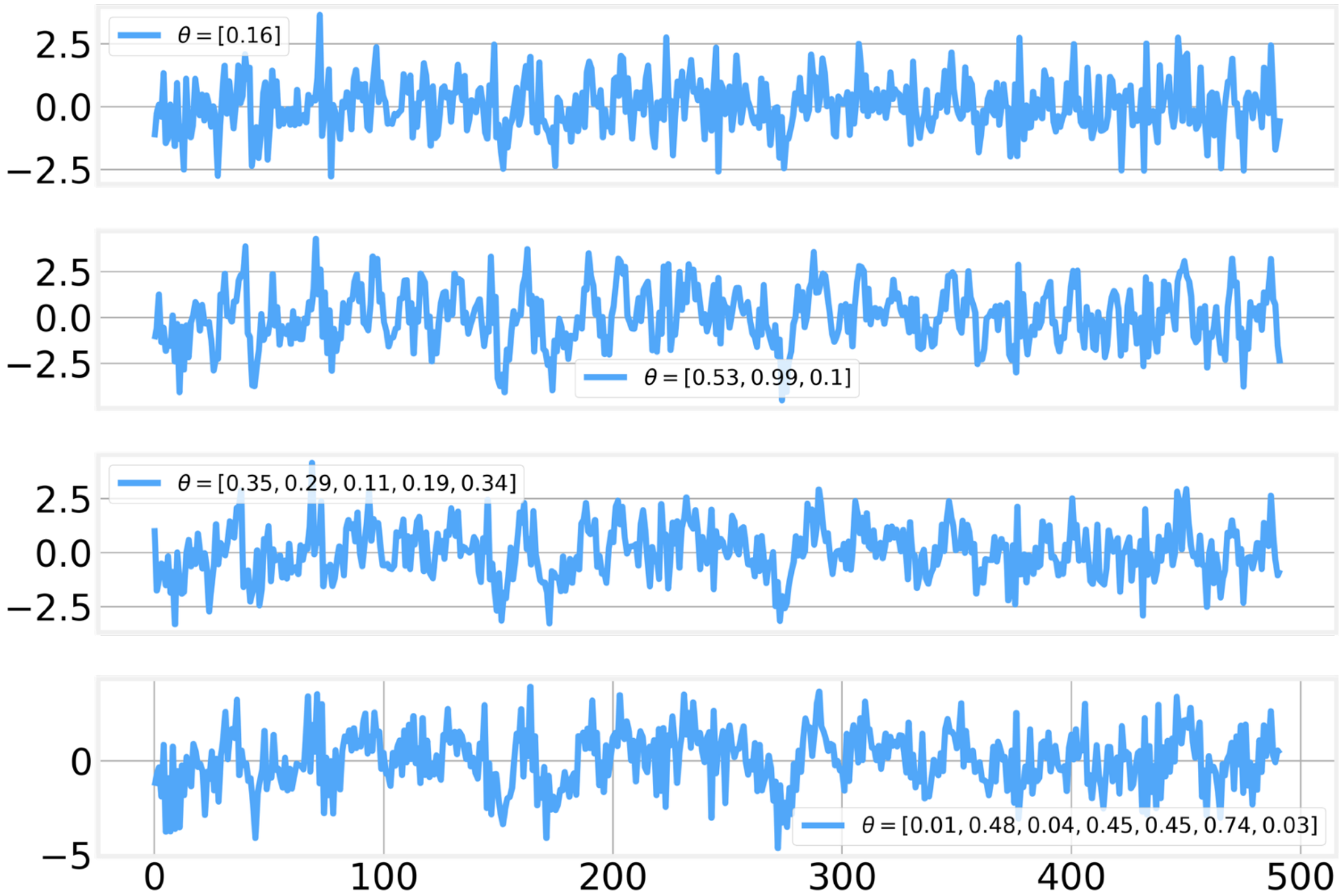
$$x_t = \epsilon_t$$

- Which we already saw in our discussion of random walks.
- A more general case can be written as:

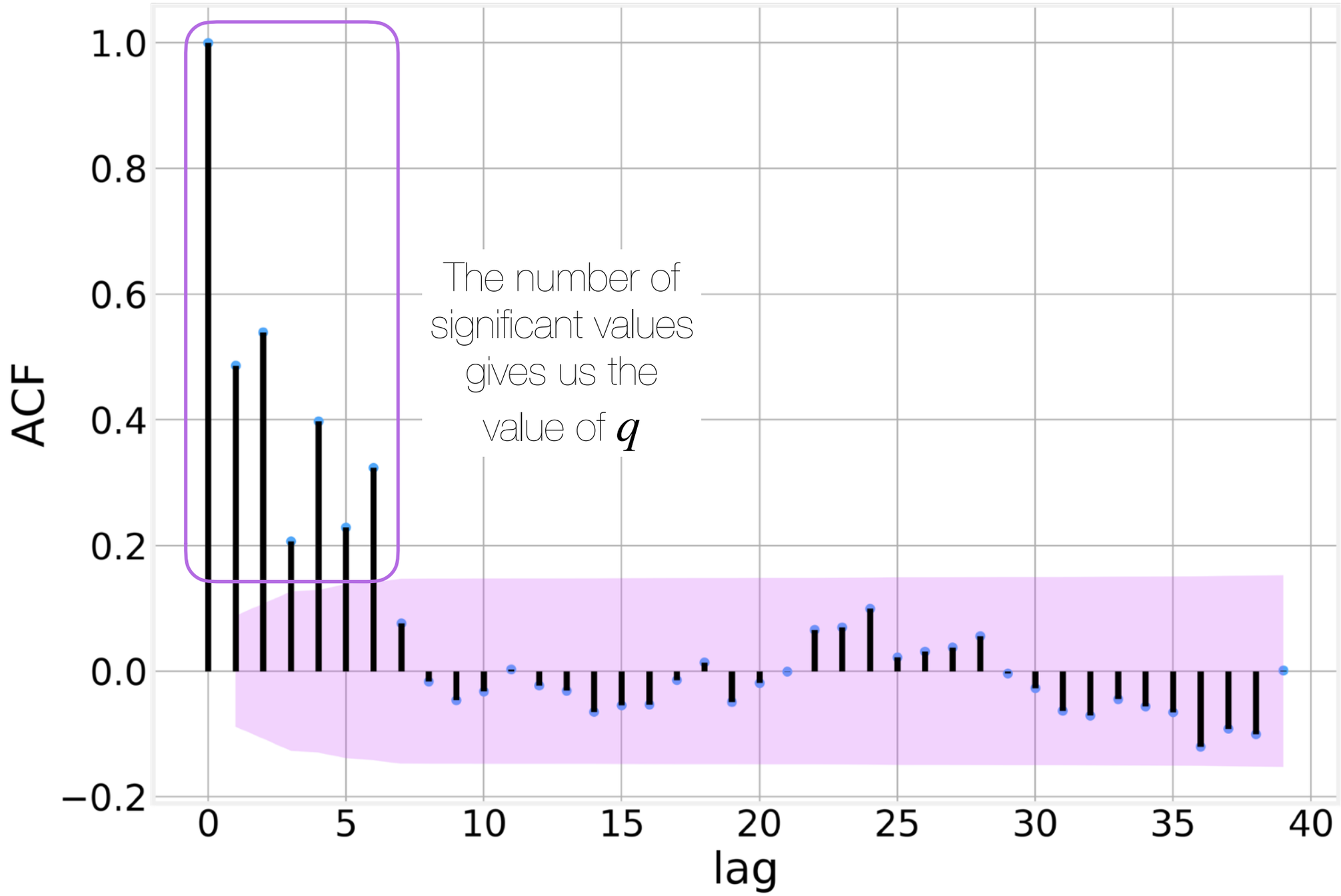
$$x_t = \beta + \sum_{l=0}^{q} \omega_l \, \epsilon_{t-l}$$

- where β is a constant offset, ω_l are the weights for the values at lag l and q is the moving window size. $\omega_0 \equiv 1$.
- The ϵ_t values are stochastic variables (often referred to as "errors") rather than the actual observed values x_t
- The generated x_t is uncorrelated with itself for any lag $l > q$

Moving Average (MA) Model



Moving Average (MA) Model



Auto-Regressive (AR) Models

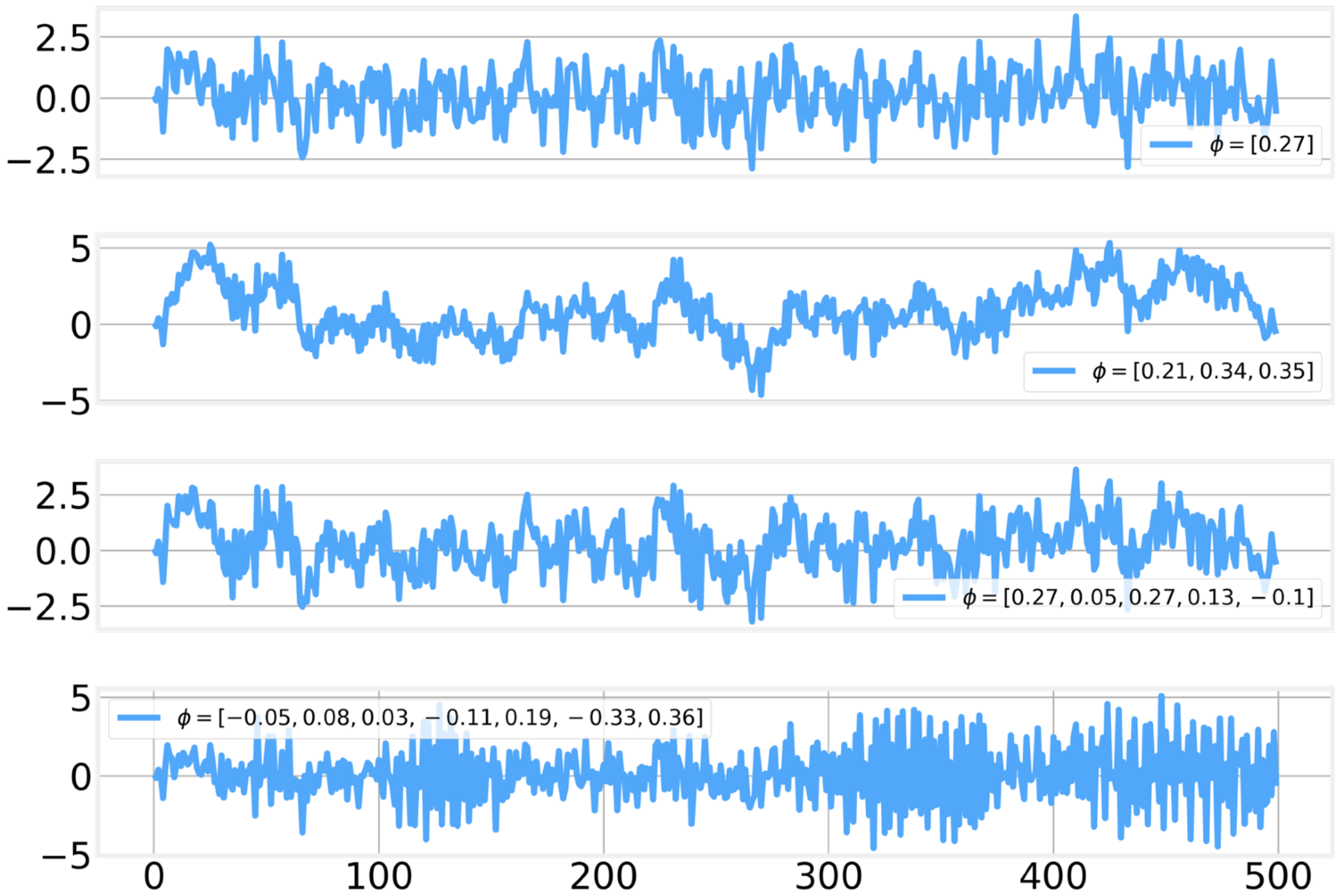
- Auto-Regressive models rely on the fact that in stationary models any deviation from the mean must be compensated. The series must "revert to the mean".

- AR models can be defined as:

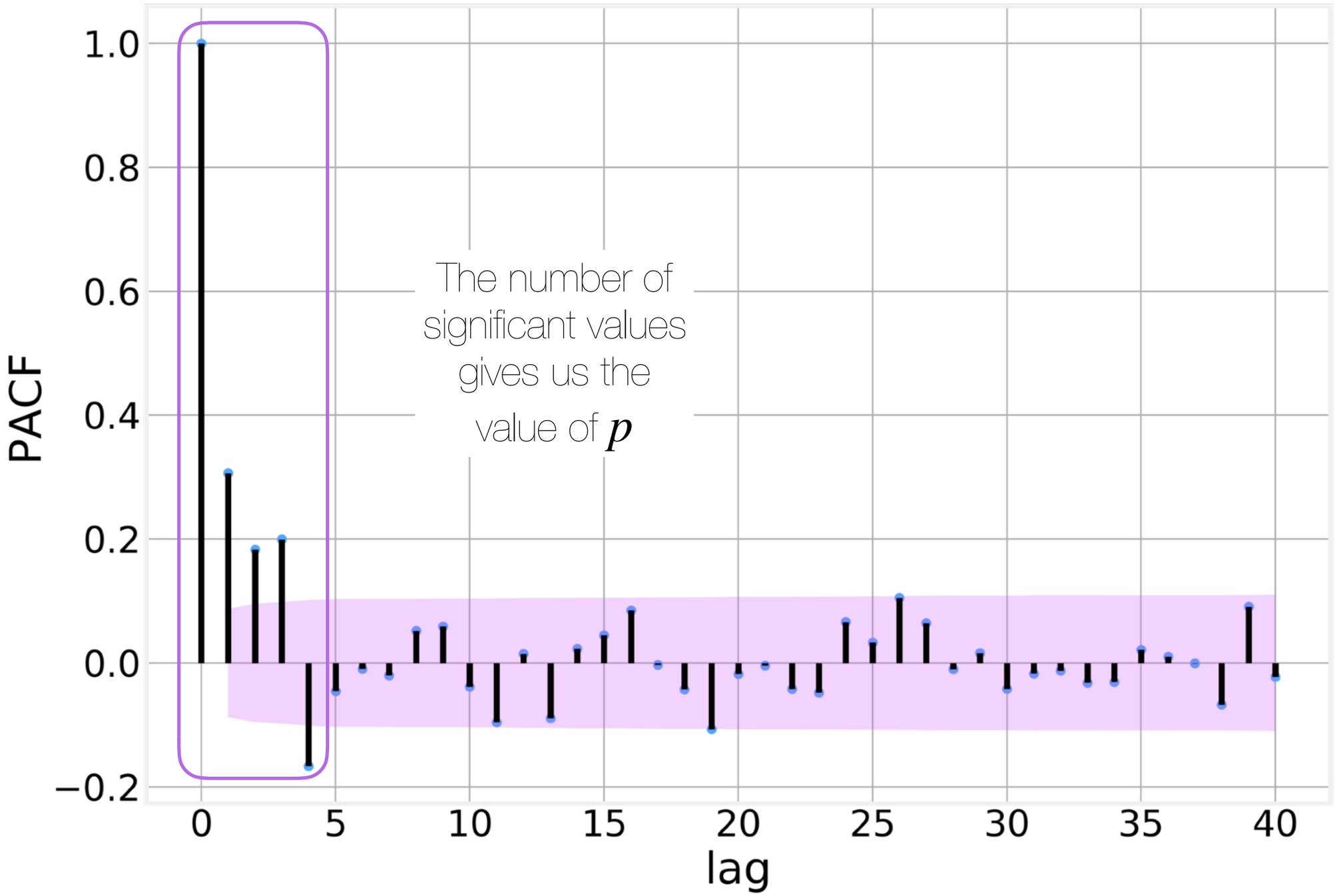
$$x_t = \alpha + \epsilon_t \sum_{l=1}^{p} \phi_l \; x_{t-l}$$

- Where the constant c represents the process' average value and the x_{t-l} are the observed values at a given lag l and ϕ_l are the corresponding weights.

Auto-Regressive (AR) Models



Auto-Regressive (AR) Models



Integrative (I) “model”

- We already saw that we can take differences to “stationarize" the time series.
- To recover the original values, we must then integrate
- While not a model by itself, it is often an important first step in modeling time series

ARIMA model

- The three classes of models we described above can be integrated into a single model:

Auto
Regressive
Integrated
Moving
Average

$$x_t = c + \sum_i^p \phi_i x_{t-i} + \epsilon_t$$

$$x_t = \mu + \epsilon_t + \sum_i^q \theta_i \epsilon_i$$

- The complete model can be written as:

$$\hat{x}_t = c + \mu + \sum_i^p \phi_i x_{t-i} + \sum_j^q \theta_i \epsilon_{t-i} + \epsilon_t$$

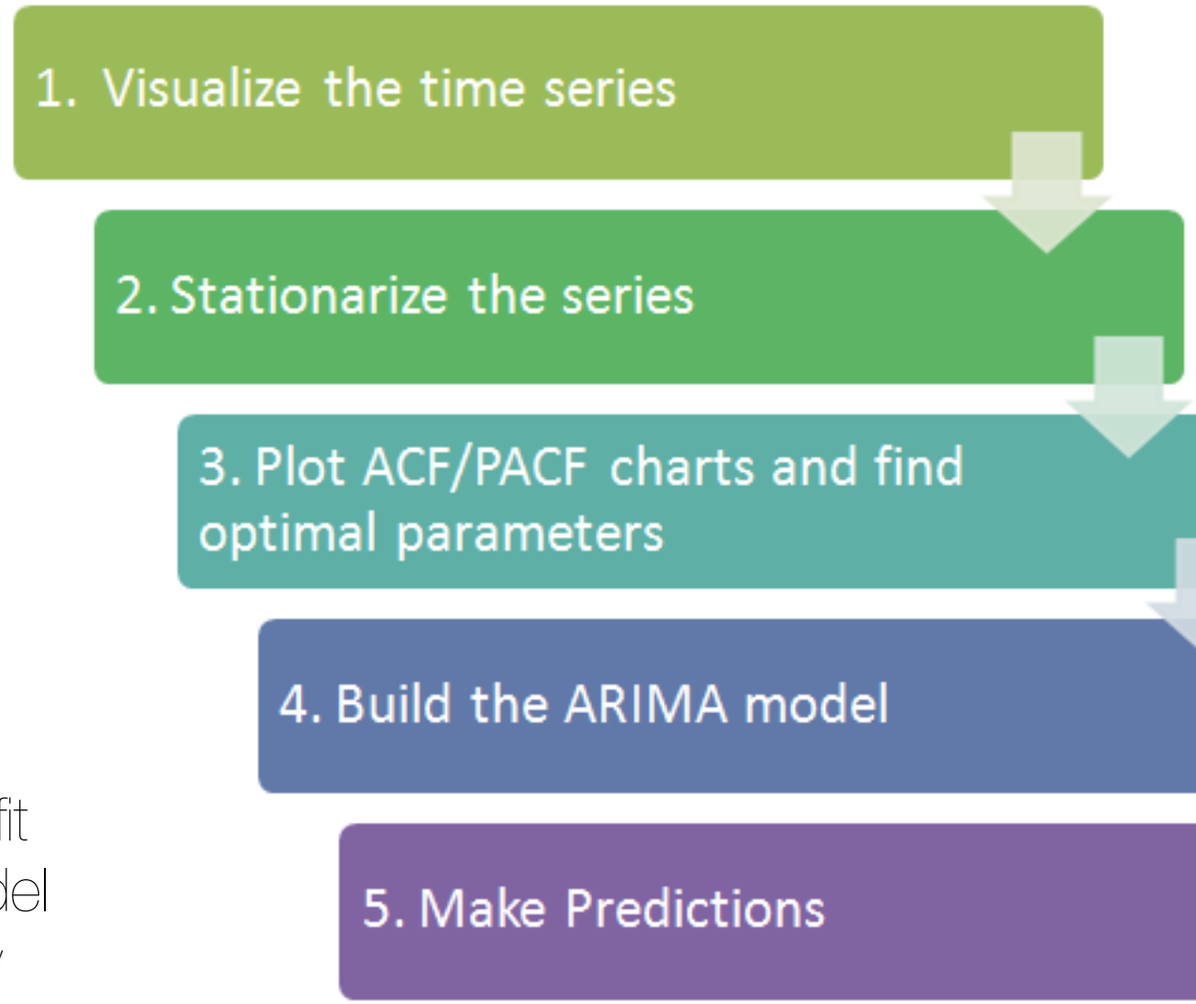
- where $\hat{x}_t$ is the properly differentiated time series

ARIMA model

- From this simple definition we can easily recover several interesting special cases:
- $ARIMA\,(0,1,0)$ - Random Walk (with or without drift)
  - $x_t - x_{t-1} = c + \epsilon_t$
- $ARIMA\,(0,0,0)$ - White noise (the sequence of stochastic variables)
  - $x_t = \epsilon_t$
- $ARIMA\,(0,1,1)$ - Exponential Smoothing
  - $x_t - x_{t-1} = \epsilon_t + \theta_1 \epsilon_{t-1}$
- $ARIMA\,(0,2,2)$ - Double exponential Smoothing
  - $x_t - 2x_{t-1} + x_{t-2} = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2}$

Fitting ARIMA models

https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/



The general procedure to fit an ARIMA model was originally proposal by Box-Jenkins

- d - degree of differencing
- p - number of lag observations included in the model ($PACF$)
- q - size of the moving average window (ACF)

Code - ARIMA

https://github.com/DataForScience/Timeseries_long

Deep Learning For Everyone

Mar 27, 2020 - 5am-9am (PST)

Time Series for Everyone

Apr 8, 2020 - 5am-9am (PST)

Applied Probability Theory for Everyone

Apr 19, 2020 - 5am-9am (PST)

Natural Language Processing (NLP) from Scratch

http://bit.ly/LiveLessonNLP - On Demand