Time Series

Fátima Rodrigues
Departamento Engenharia Informática
mfc@isep.ipp.pt

Importance of Time Series Analysis

Ample of time series data is being generated from a variety of fields. And hence the study of time series analysis holds a lot of applications

Time series analysis is important in different areas:

- Economics
- Finance
- Healthcare
- Environmental Science
- Sales Forecasting

Why & where Time Series is used?

Time series data can be analysed in order to extract meaningful statistics and other characteristics

It's used in at least in the four scenarios:

- Business Forecasting
- Understanding past behaviour
- Plan the future
- Evaluate current accomplishment

Areas of application for time series analysis

- **Time series forecasting**

  predict the future values of a time series, given the past values

  Ex: predict the next day's temperature using the last 5 years of temperature data

- **Time series classification**

  predict the class of a time series based on others time series

  Ex: given a history of an electroencephalogram (EEG) or an electrocardiogram (EKG) predict whether the result of an EEG or an EKG is normal or abnormal

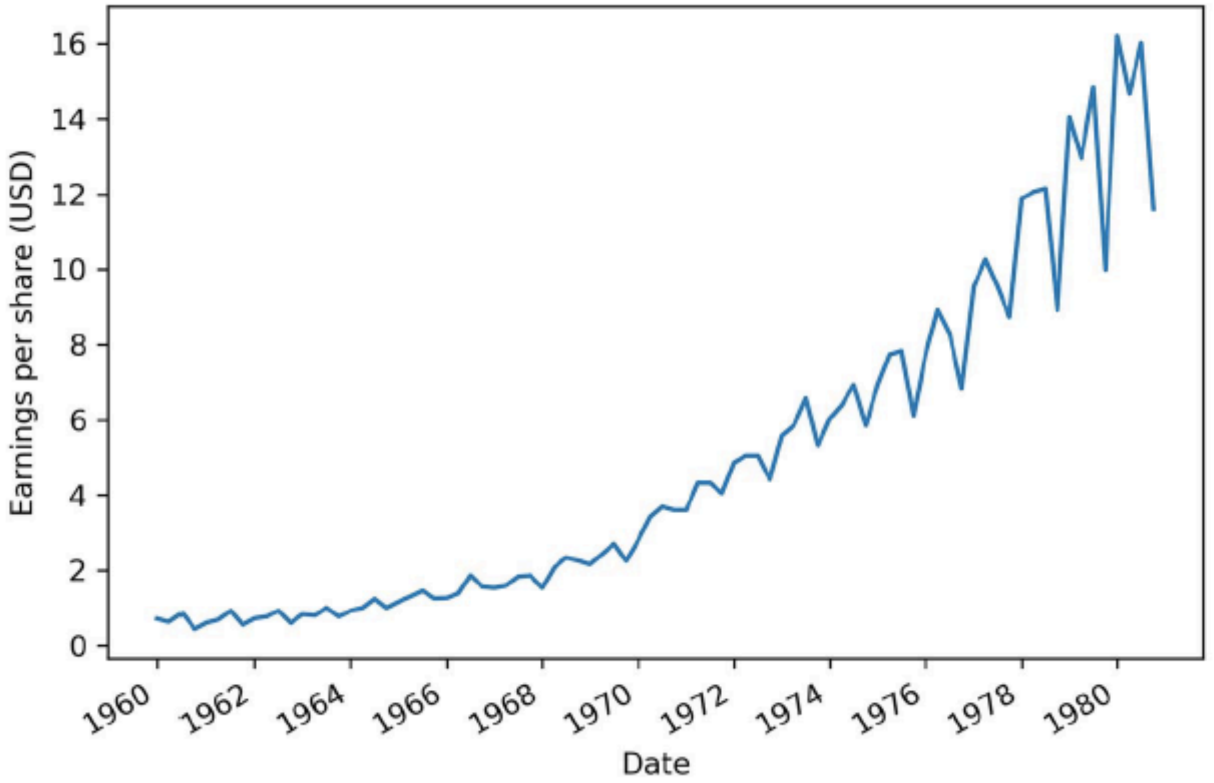
- **Interpretation and causality**

  understand the interrelationships among several related time series, or derive causal inference based on time series data

Time Series

A time series is a set of observations, $y_1, y_2, y_3, \ldots$, ordered with equal time intervals, $t_1, t_2, t_3, \ldots$

In a time series, the data are recorded sequentially over a certain period of time, so the existence of serial correlation over time is assumed



Components of a Time Series

Time series decomposition is a process by which we separate a time series into its components:

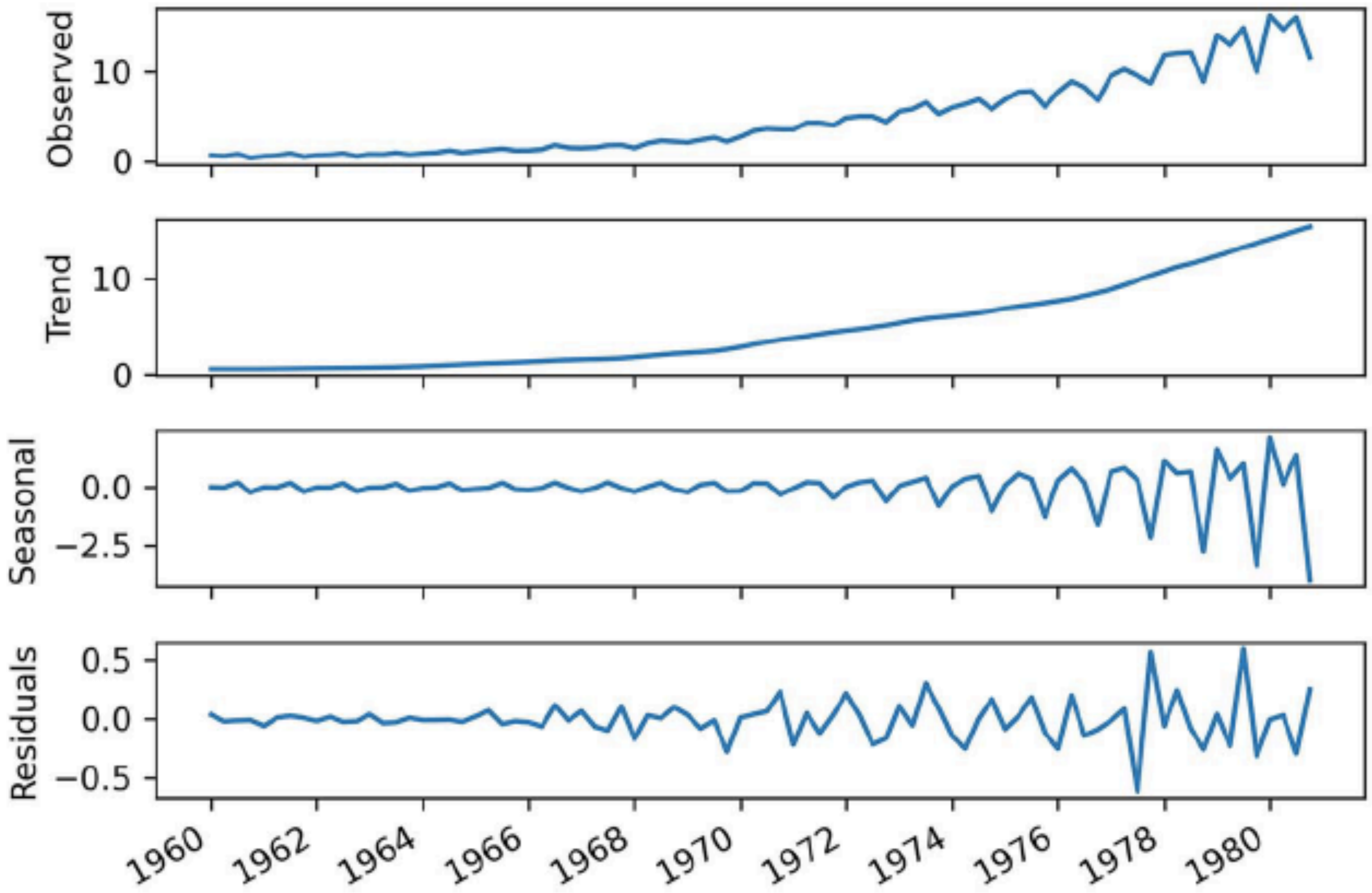
- **Trend**

  Represents the slow-moving changes in a time series. It is responsible for making the series gradually increase or decrease over time
- **Seasonality**

  Represents the seasonal pattern in the series. The cycles occur repeatedly over a fixed period of time
- **Residuals**

  Represent the behaviour that cannot be explained by the trend and seasonality components. They correspond to random errors, also termed white noise

Components of a Time Series



Decomposition of a time series

The modelling of the decomposed components can be:

Additive: when the original time series can be reconstructed by adding all three components

$$y_t = T_t + S_t + R_t$$

Multiplicative: the time series can be reconstructed by multiplying all three components

$$y_t = T_t \times S_t \times R_t$$

Baseline models

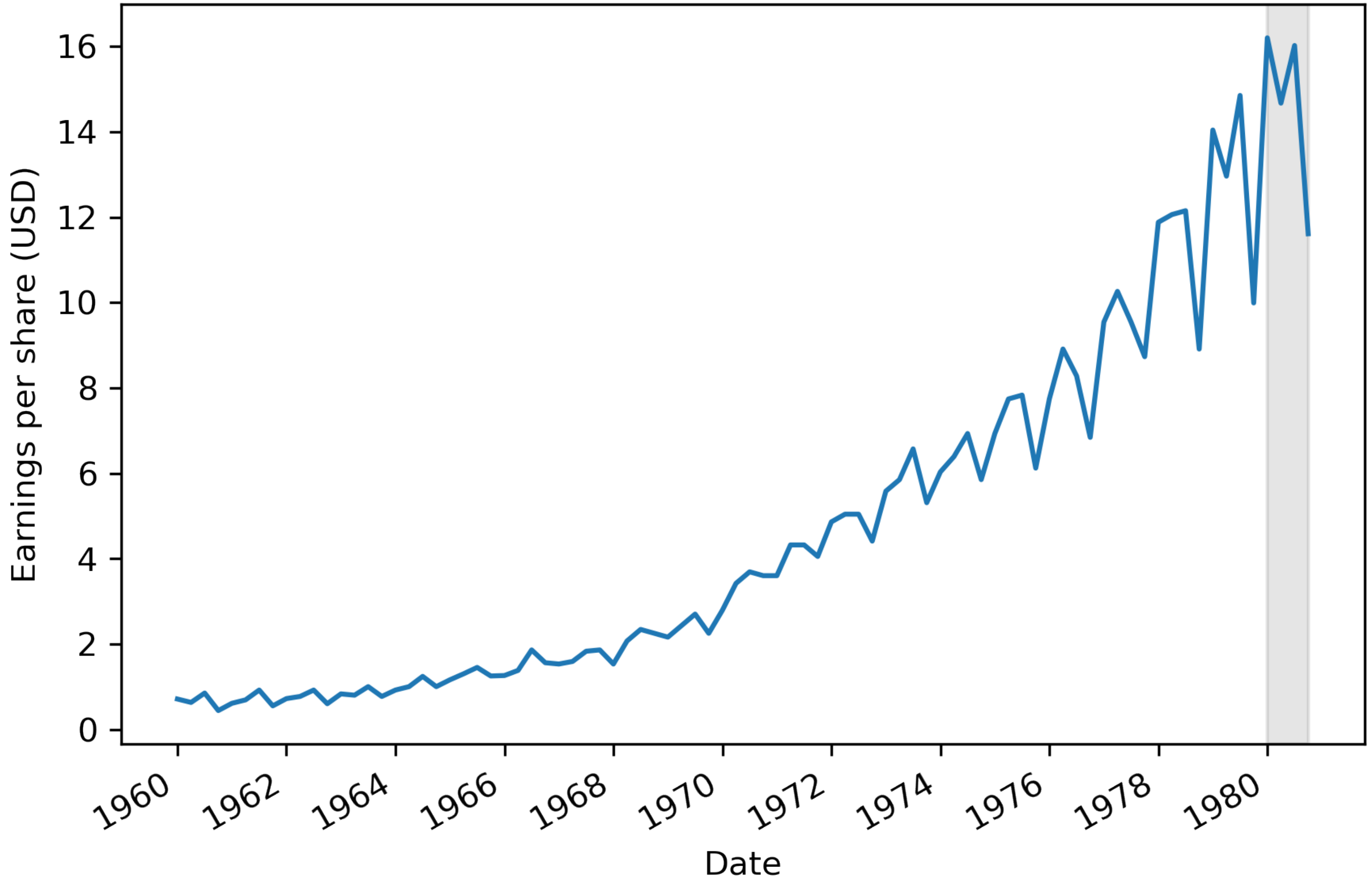
Baseline model

- A baseline model is a trivial solution for forecasting a problem
- It relies on heuristics or simple statistics and is usually the simplest solution
- It does not require model fitting, and it is easy to implement

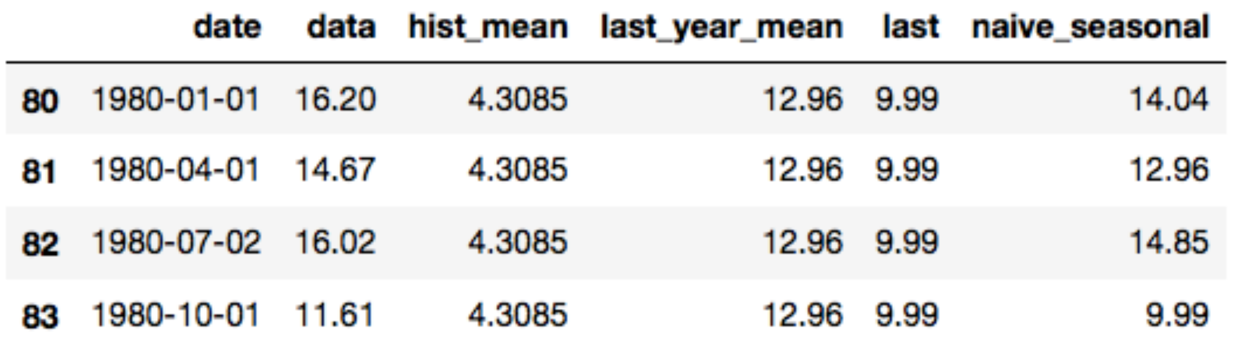
Examples of baseline models:

- historical mean
- last year’s mean
- last known value
- naive seasonal forecast

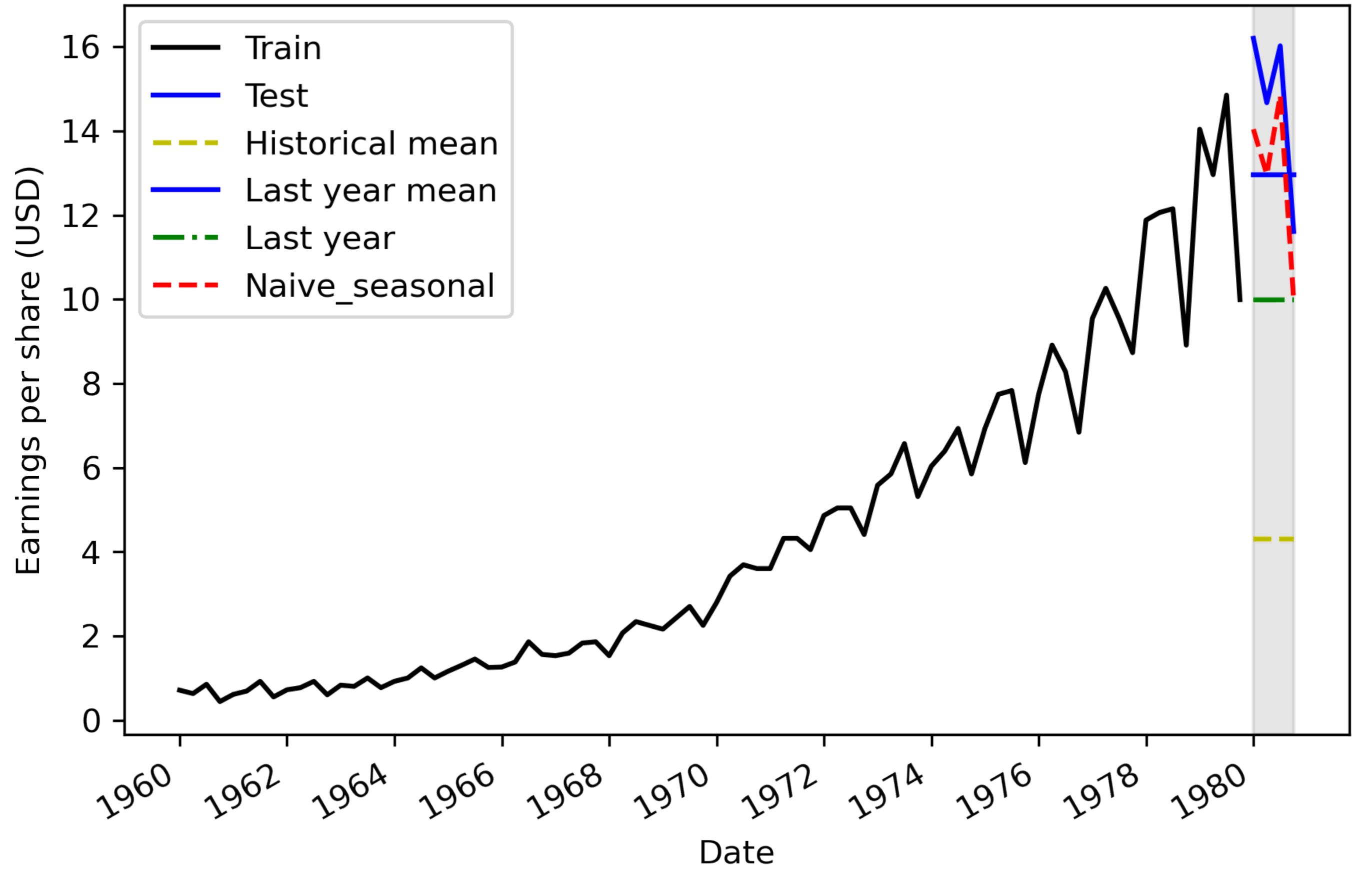
Predict the last four quarters with baseline models

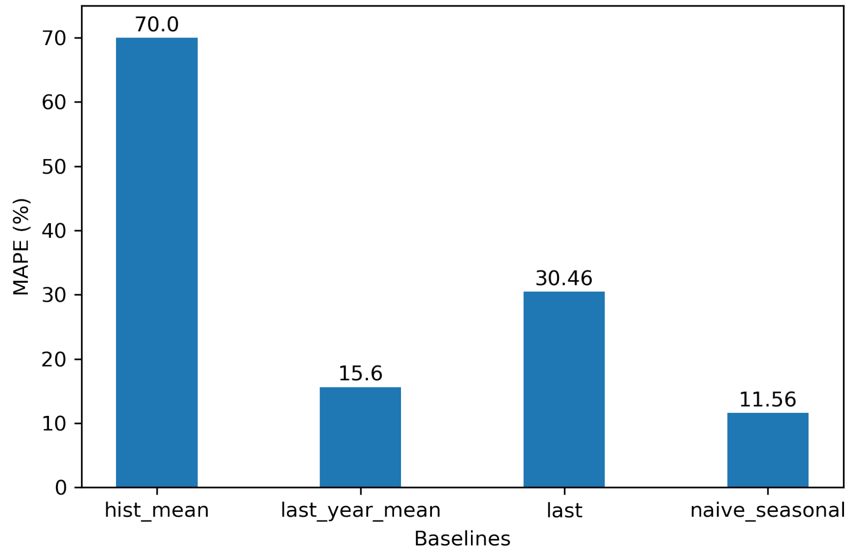


Predict the last four quarters with baseline models



	date	data	hist_mean	last_year_mean	last	naive_seasonal
80	1980-01-01	16.20	4.3085	12.96	9.99	14.04
81	1980-04-01	14.67	4.3085	12.96	9.99	12.96
82	1980-07-02	16.02	4.3085	12.96	9.99	14.85
83	1980-10-01	11.61	4.3085	12.96	9.99	9.99





Random Walk

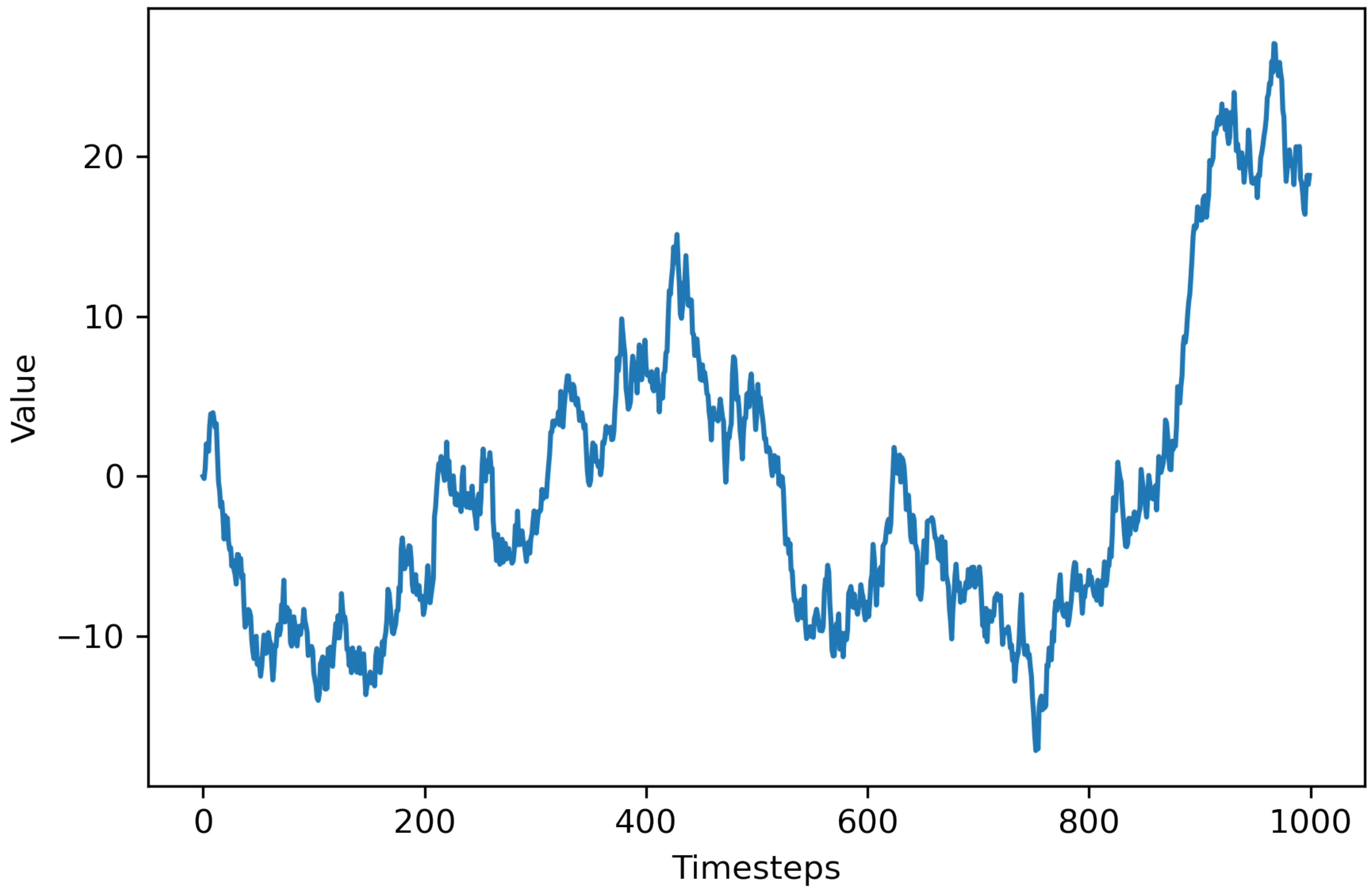
Random Walk

- A random walk is a process in which there is an equal chance of going up or down by a random number
- Random walks often expose long periods where a positive or negative trend can be observed. They are also often accompanied by sudden changes in direction

A random walk can be mathematically express with the following equation

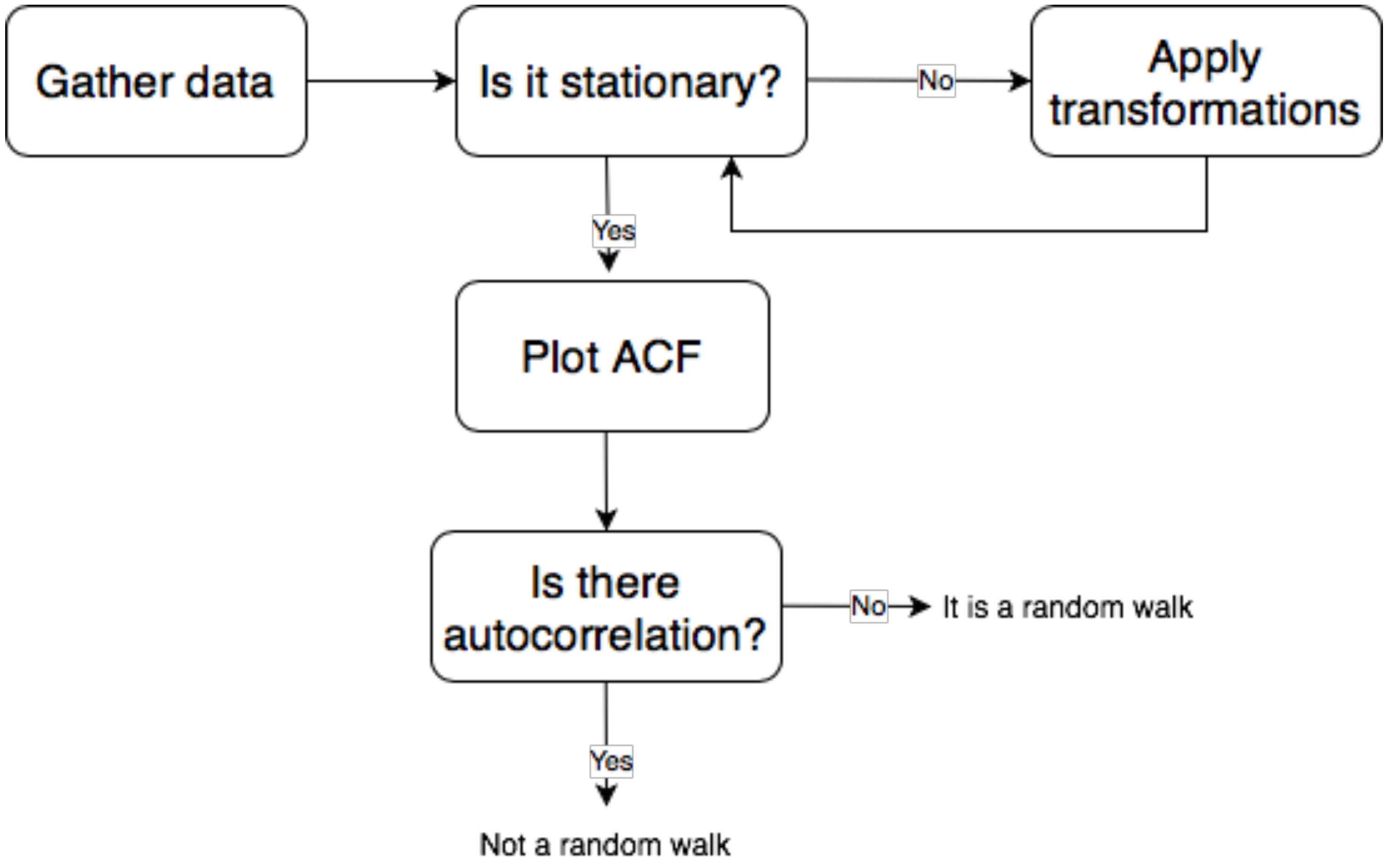
$$y_t = C + y_{t-1} + \epsilon_t$$

Random walk example



Identifying a Random Walk

- A random walk is a series whose first difference is **stationary** and **uncorrelated**



Stationarity

Before applying any statistical model on a time series, the series has to be **stationary**, which means that, over different time periods:

1. it should have constant mean
2. It should have constant variance or standard deviation
3. Autocorrelation do not change on time

How to make the time series stationary

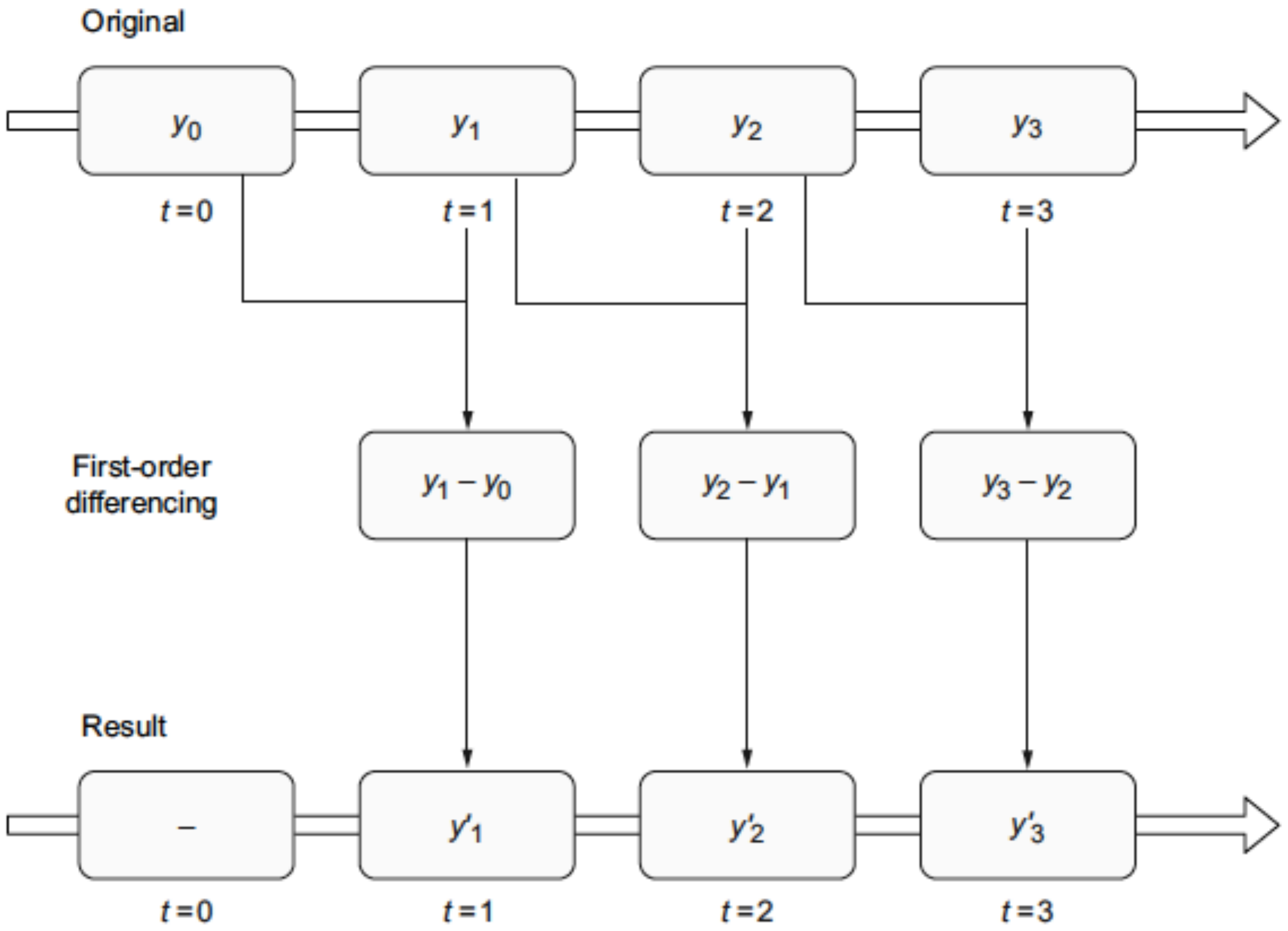
- **Differencing** is a transformation that calculates the change from one timestep to another
- This transformation helps stabilize the mean, which in turn removes or reduces the trend and seasonality effects
- Differencing involves calculating the series of changes from one timestep to another

$$y'_t = y_t - y_{t-1}$$

It is possible to difference a time series many times:

- taking the difference once is applying a **first-order** differencing
- taking it a second time would be a **second-order** differencing

Visualizing a first-order difference



Test to check if a series is stationary

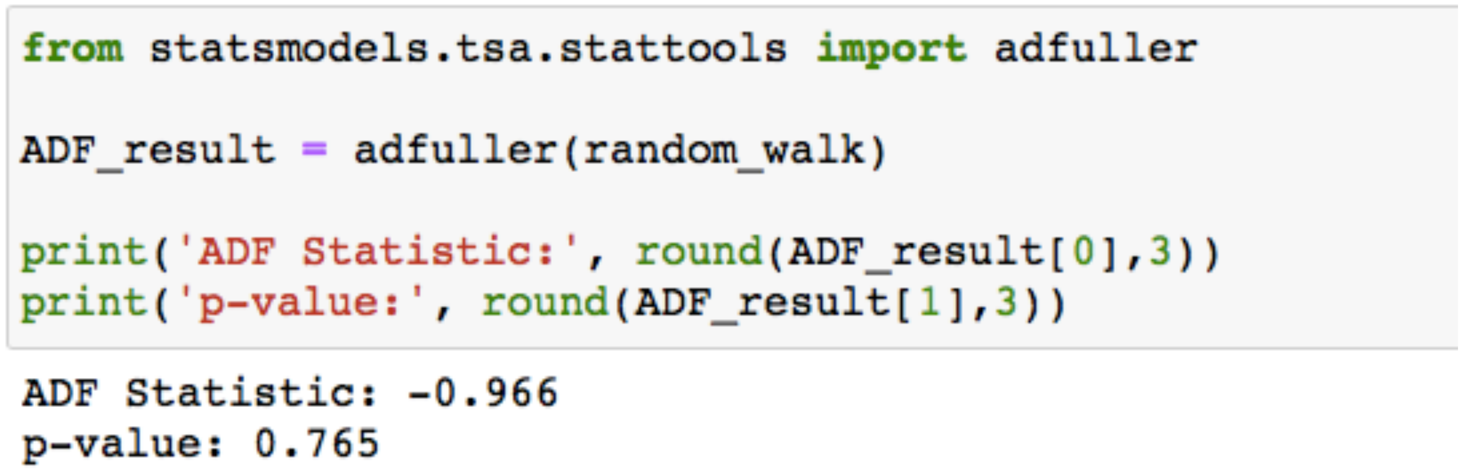
ADCF Test - **Augmented Dickey–Fuller test**

Null hypothesis: says that the time series is non-stationary

The result of this test is the ADF statistic, which is a negative number. The more negative it is, the stronger the rejection of the null hypothesis

If the **p-value** is **less than 0.05**, we can also **reject the null hypothesis** and say the **series is stationary**

ADCF Test to check if a series is stationary



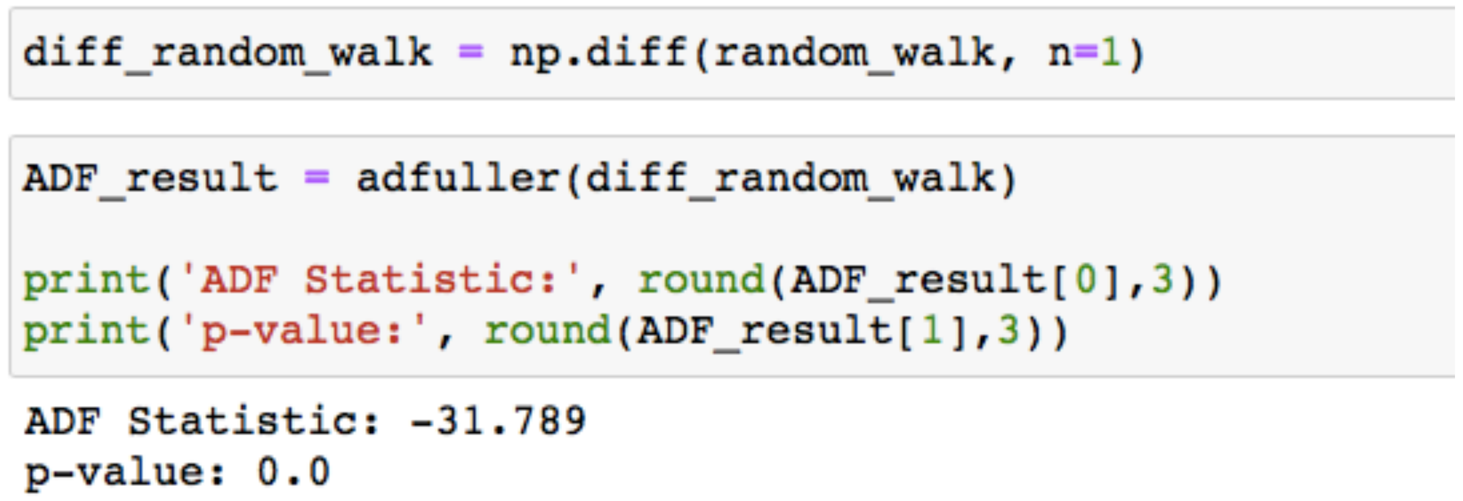
```
from statsmodels.tsa.stattools import adfuller

ADF_result = adfuller(random_walk)

print('ADF Statistic:', round(ADF_result[0],3))
print('p-value:', round(ADF_result[1],3))
```

```
ADF Statistic: -0.966
p-value: 0.765
```

Since the series is not stationary a first-order differencing must be applied



```
diff_random_walk = np.diff(random_walk, n=1)
```

```
ADF_result = adfuller(diff_random_walk)

print('ADF Statistic:', round(ADF_result[0],3))
print('p-value:', round(ADF_result[1],3))
```

```
ADF Statistic: -31.789
p-value: 0.0
```

Autocorrelation function (ACF)

The autocorrelation function (ACF) measures the linear relationship between lagged values of a time series

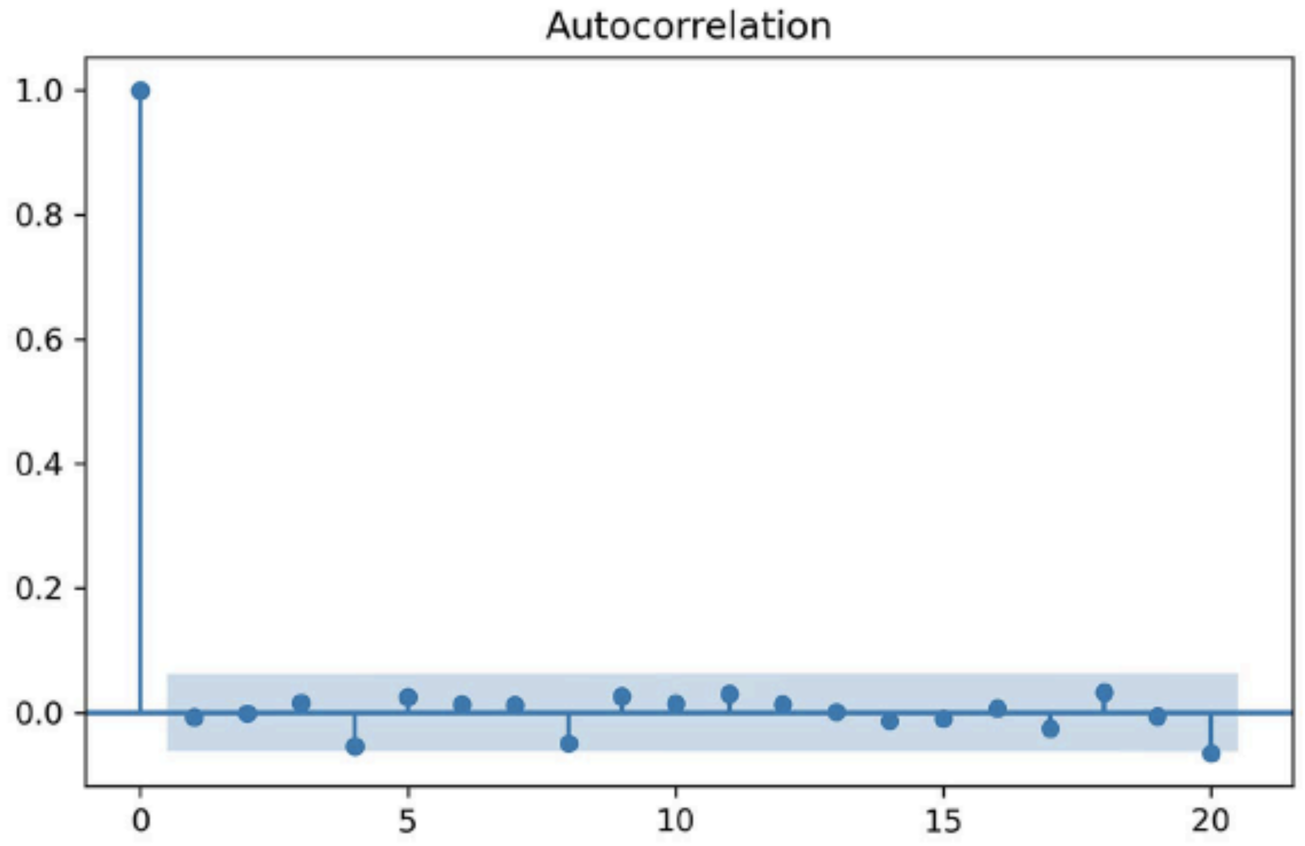
It measures the correlation of the time series with itself

The ACF calculates the autocorrelation coefficient between:

$$y_t \text{ and } y_{t-1} : r_1$$
$$y_t \text{ and } y_{t-2} : r_2$$
$$\ldots$$

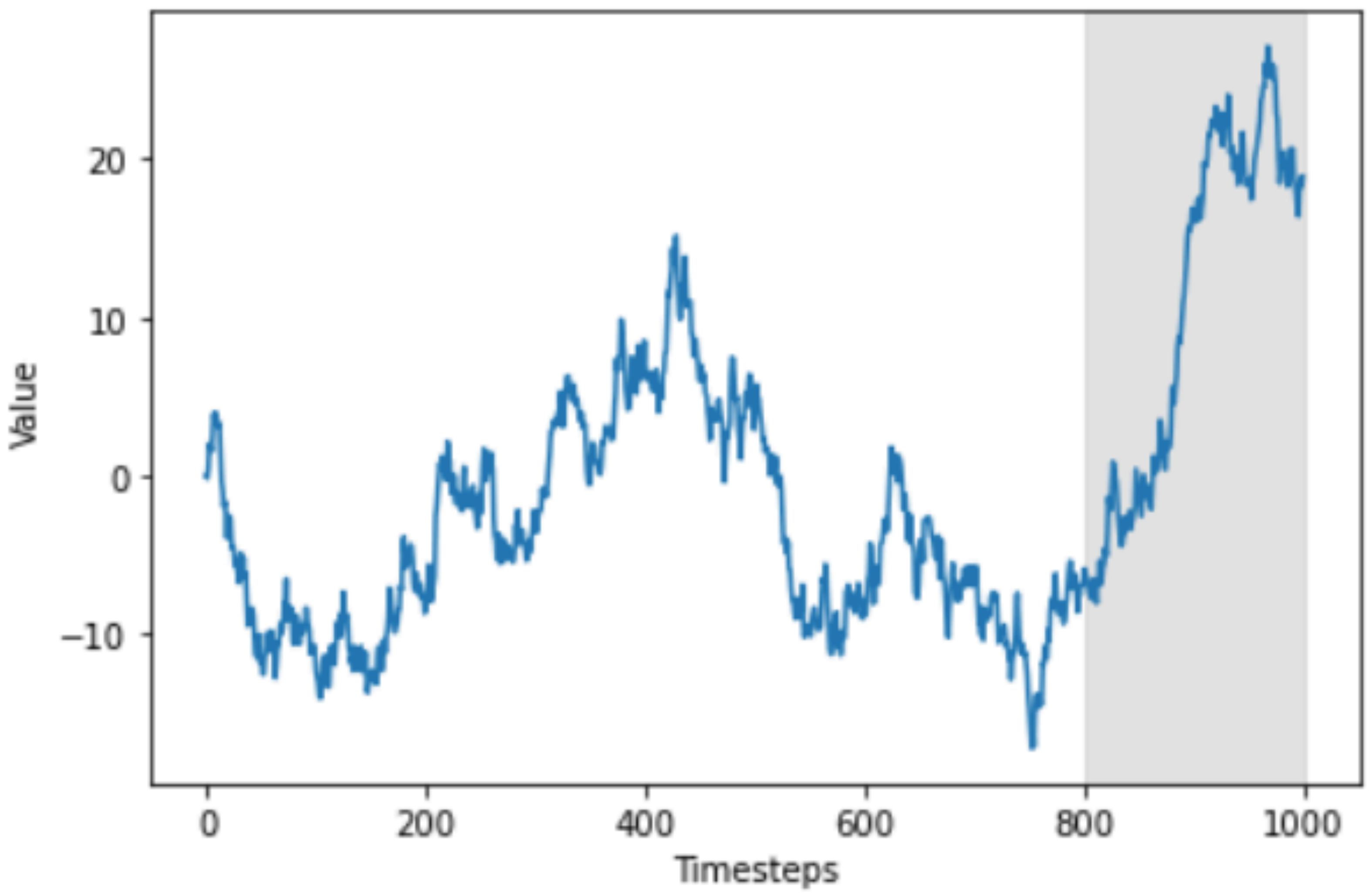
In the ACF plot the coefficient is the dependent variable, while the lag is the independent variable

ACF plot of the random walk



There are no significant coefficients after lag 0, which is a clear indicator of a **random walk** - can be described as **white noise**

Forecasting a random walk



Forecasting a random walk

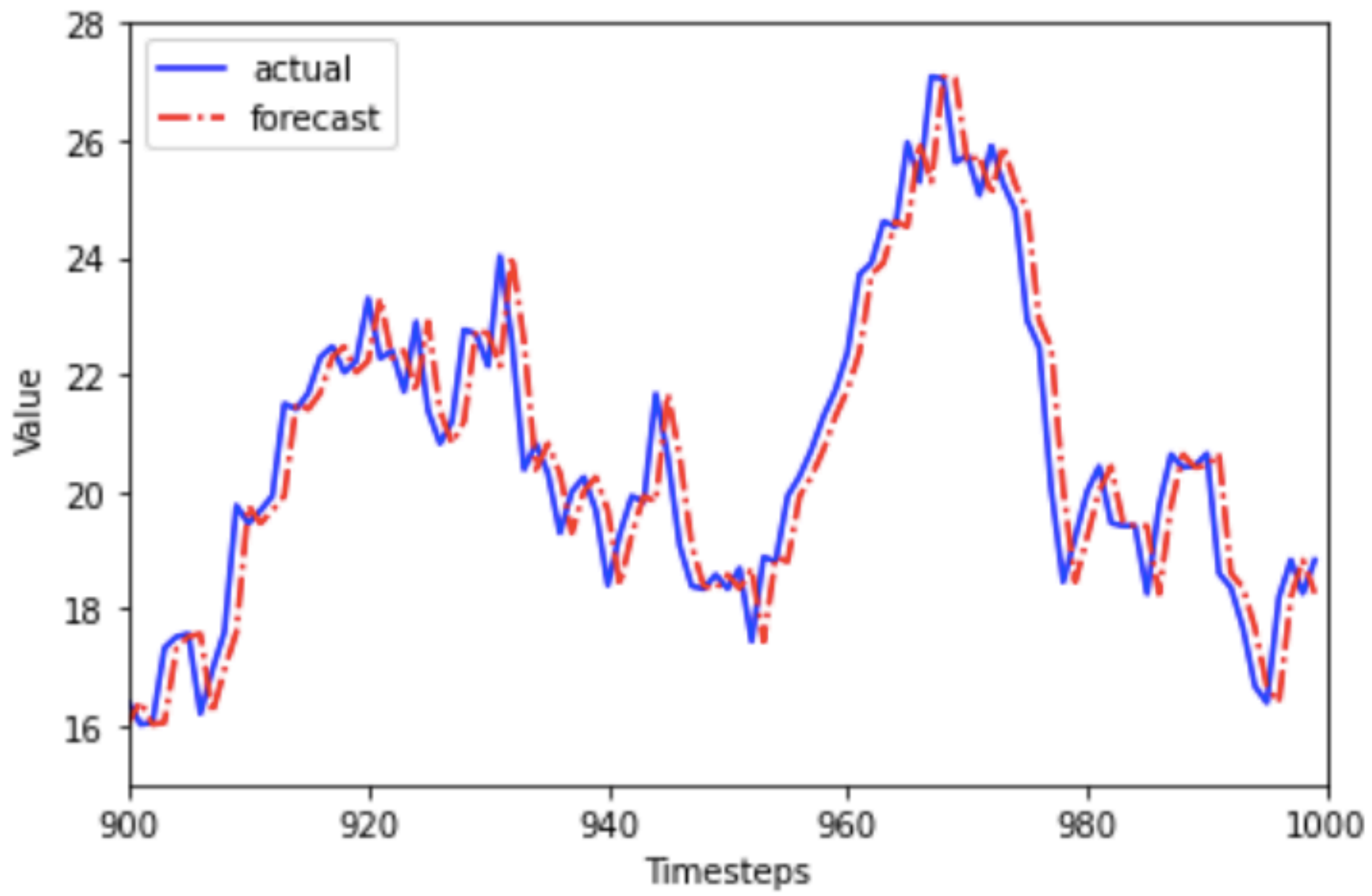
- To predict a random walk, we can only use **naive forecasting** methods or baseline methods

- Forecasting a random walk on a **long horizon** does not make sense — the randomness portion is magnified in a long horizon where many random numbers are added over the course of many timesteps

- In a random walk, it is only possible to forecast the **next timestep**

- The present observed value is used as a forecast for the next timestep. Once a new value is recorded, it will be used as a forecast for the following timestep

Forecasting a random walk



```
In [39]: mse_one_step = mean_squared_error(test['value'], df_shift[800:])

         mse_one_step

Out[39]: 0.9256876651440581
```

Forecasting with statistical models

Statistical models for time series forecasting

- MA(q) models
- AR(p) models
- ARMA(p,q) models
- Exponential Smoothing
- ARIMA(p,d,q) models for non-stationary time series
- SARIMA(p,d,q)(P,D,Q)$_m$ for seasonal time series
- SARIMAX models to include external variables in the forecast
- VAR(p) model for predicting many time series at once

Stationary time series

Moving Average
MA(q)

Moving Average model

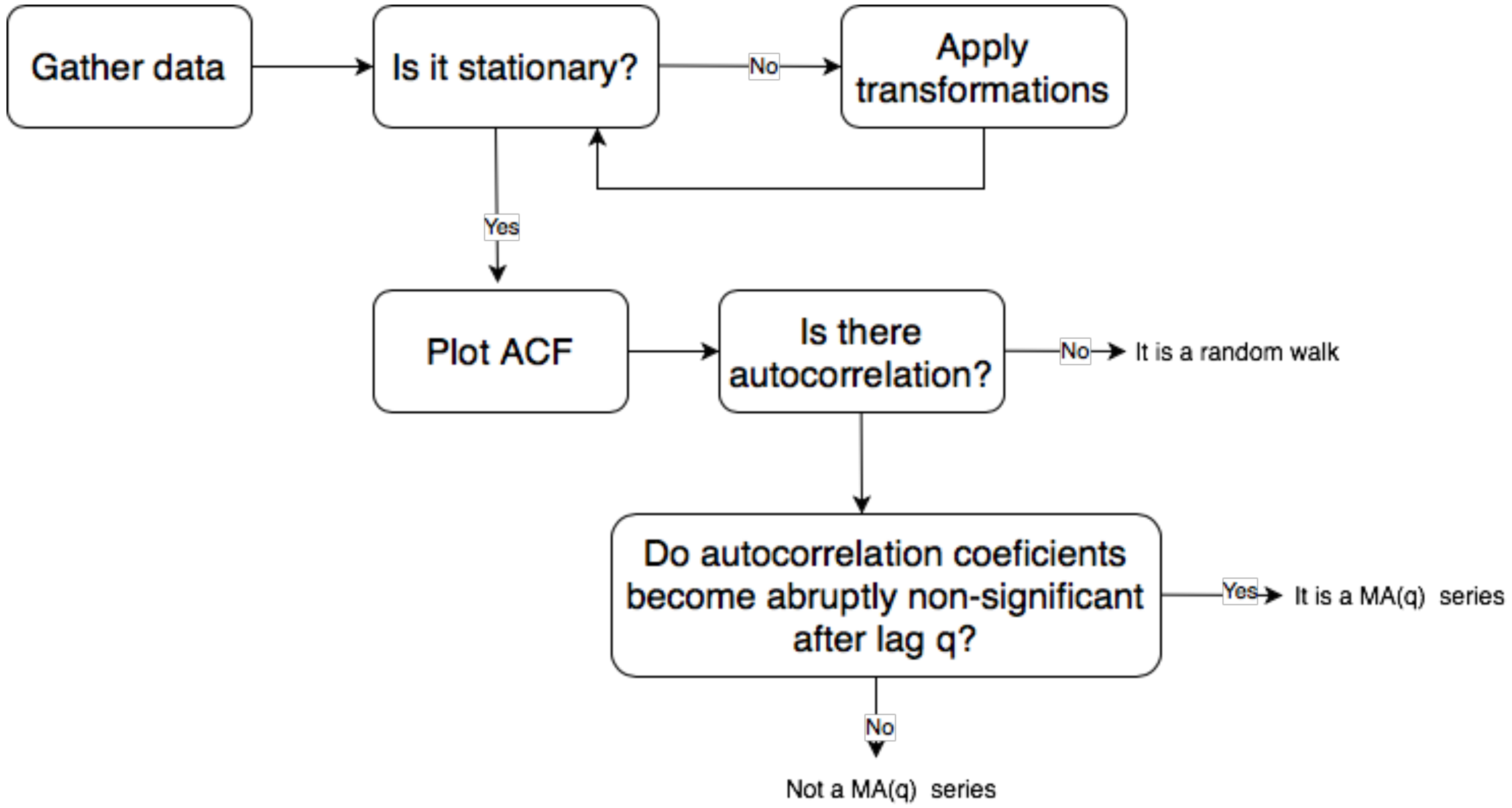
- In a moving average (MA) model, the current value depends linearly on the mean of the series, the current error term, and past error terms
- The moving average model is denoted as MA(q), where q is the order

The general expression of an MA(q) model is

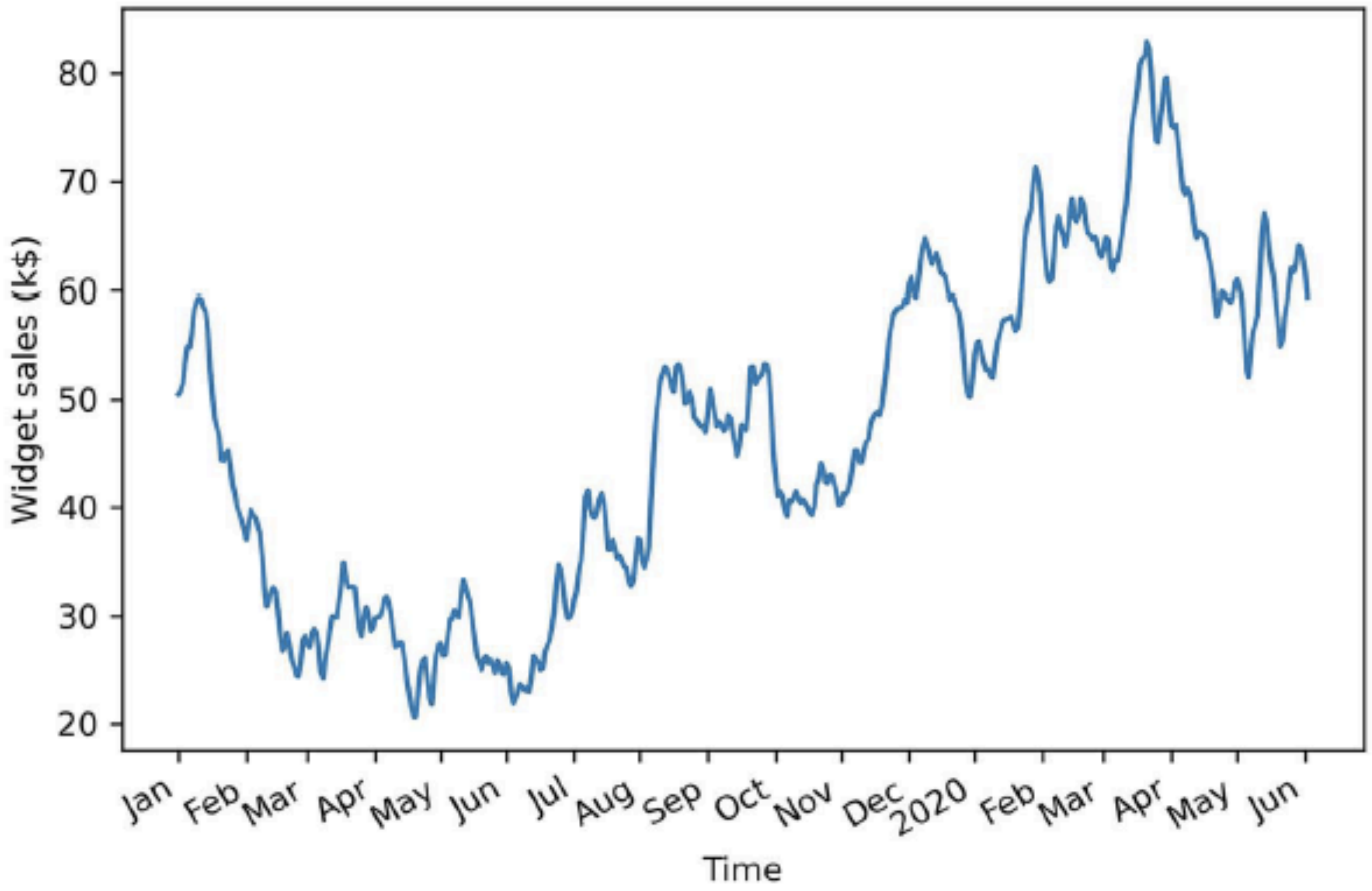
$$y_t = \mu + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + .. + \theta_q e_{t-q}$$

The order q of the moving average model determines the number of past error terms that affect the present value

Identifying a Moving Average series



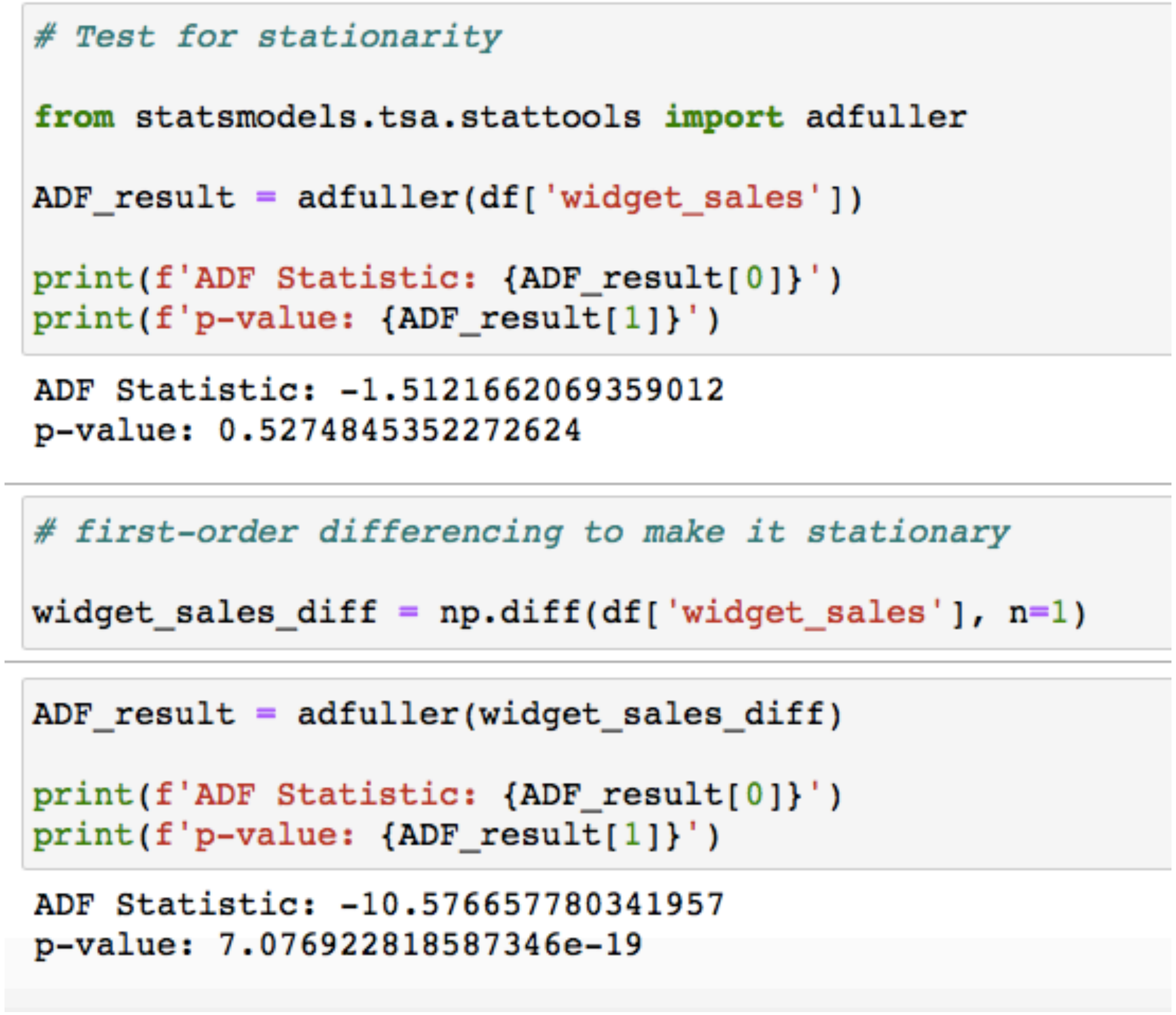
Moving Average series example



Volume of sales for a Company over 500 days, starting on January 1, 2019

Identifying the order of a Moving Average series

Test for stationarity



```python
# Test for stationarity

from statsmodels.tsa.stattools import adfuller

ADF_result = adfuller(df['widget_sales'])

print(f'ADF Statistic: {ADF_result[0]}')
print(f'p-value: {ADF_result[1]}')
```

```
ADF Statistic: -1.5121662069359012
p-value: 0.5274845352272624
```

```python
# first-order differencing to make it stationary

widget_sales_diff = np.diff(df['widget_sales'], n=1)
```

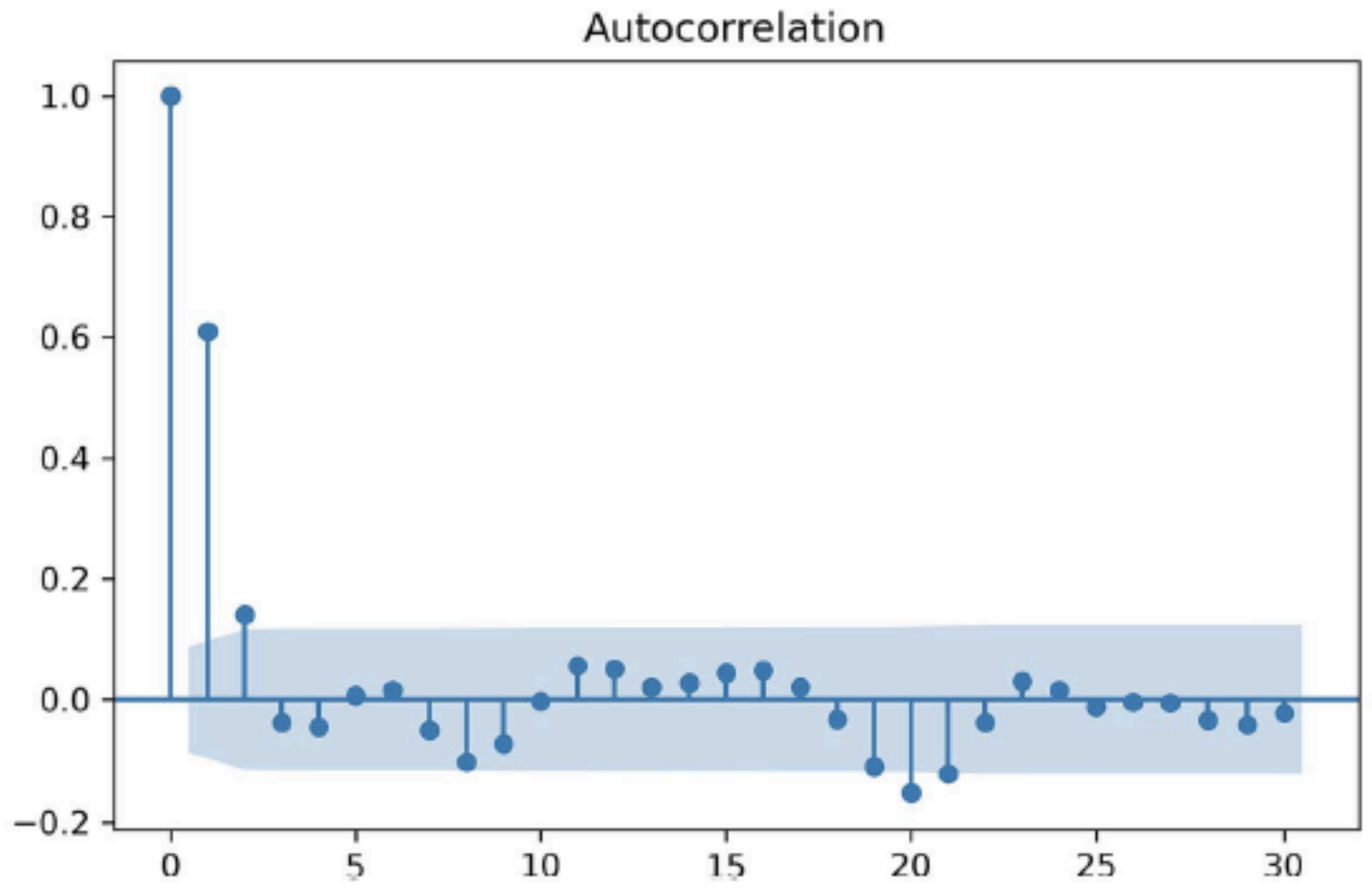
```python
ADF_result = adfuller(widget_sales_diff)

print(f'ADF Statistic: {ADF_result[0]}')
print(f'p-value: {ADF_result[1]}')
```

```
ADF Statistic: -10.576657780341957
p-value: 7.076922818587346e-19
```

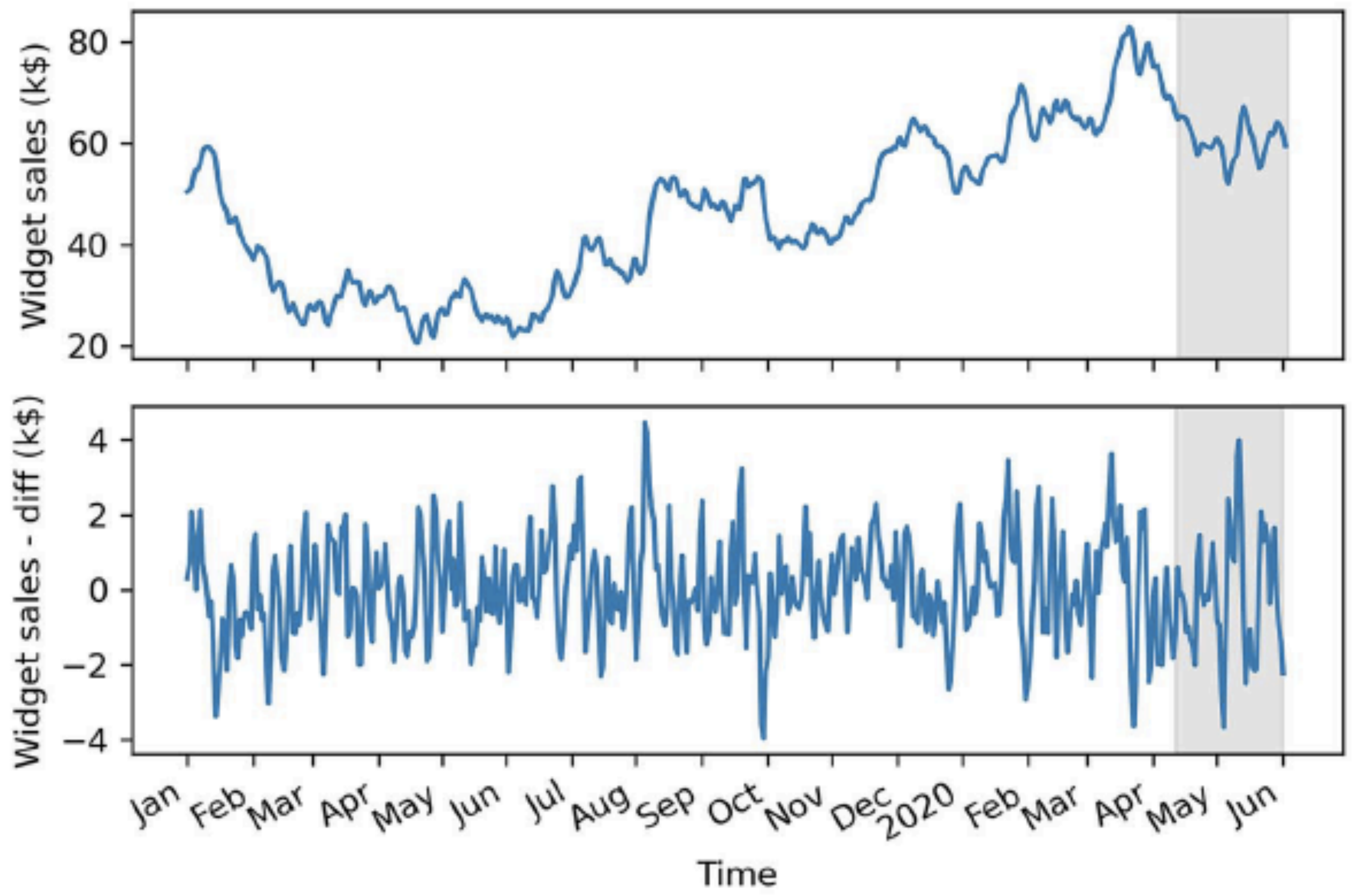
Identifying the order of a Moving Average series

Plot the autocorrelation function - ACF



Stationary moving average series of order 2 - MA(2)

Forecasting a Moving Average series



Forecasting using the MA(q) model

- When using an MA(q) model, forecasting **beyond q steps** into the future will simply return the mean - there are no error terms to estimate beyond q steps

- It is possible to use ***rolling forecasts*** to predict **up to q steps** at a time

- In a dataset with 500 steps, to predict the last 50 steps:
  - First pass: train on the first 449 timesteps to predict timesteps 450 and 451
  - Second pass: train on the first 451 timesteps to predict timesteps 452 and 453
  - …
  - This is repeated until the values at timesteps 498 and 499 are predicted

A function for rolling forecasts on a horizon

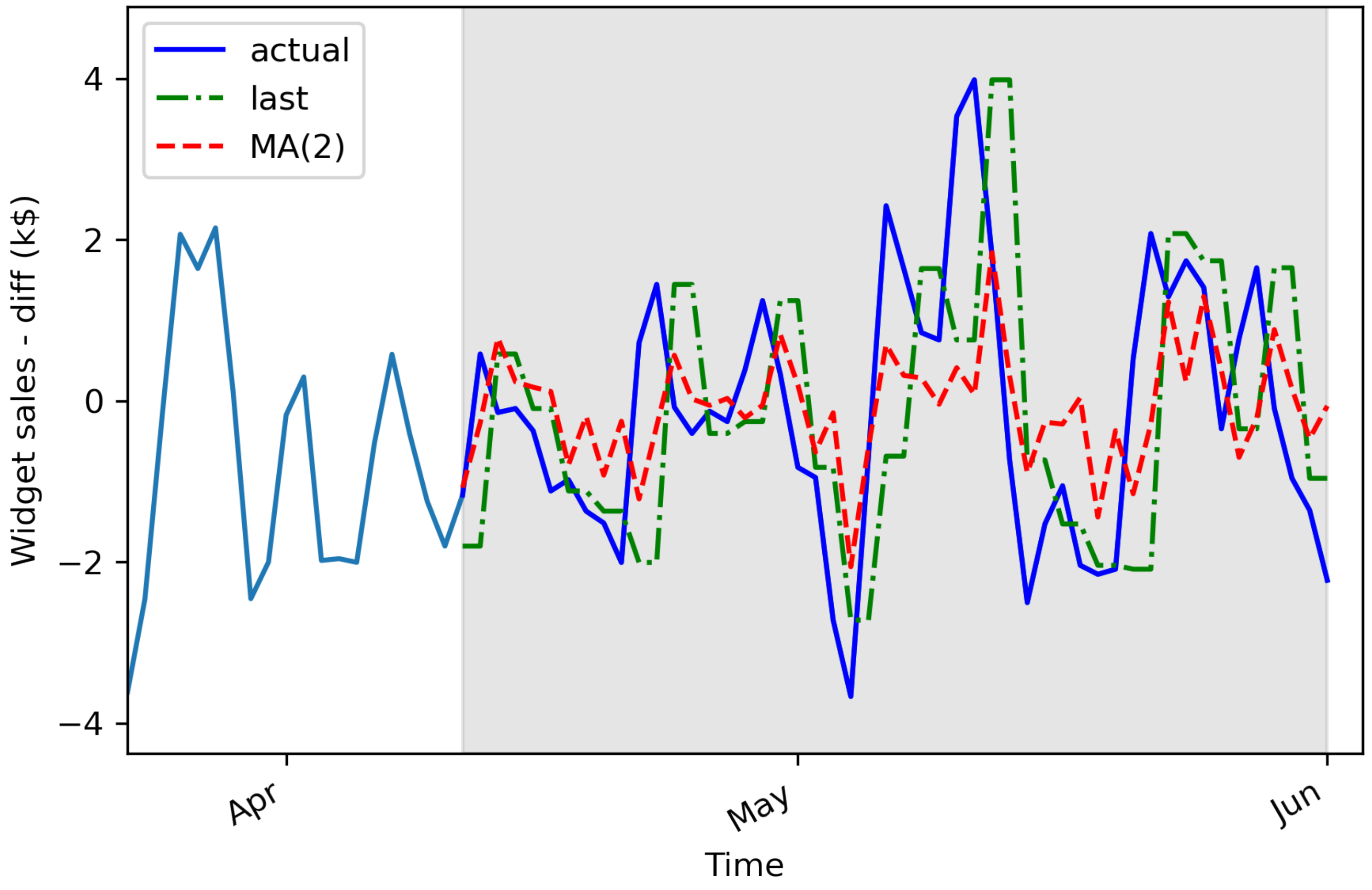
```
from statsmodels.tsa.statespace.sarimax import SARIMAX

def rolling_forecast(df: pd.DataFrame, train_len: int, horizon: int, window: int):
   total_len = train_len + horizon
   pred_MA = []

   for i in range(train_len, total_len, window):
       model = SARIMAX(df[:i], order=(0,0,2))
       res = model.fit(disp=False)
       predictions = res.get_prediction(0, i + window - 1)
       oos_pred = predictions.predicted_mean.iloc[-window:]
       pred_MA.extend(oos_pred)

    return pred_MA
```

Forecasting the MA(2) series



MSE last value: 3.249
MSE MA(2): 1.948

Differencing to obtain the series to the original scale

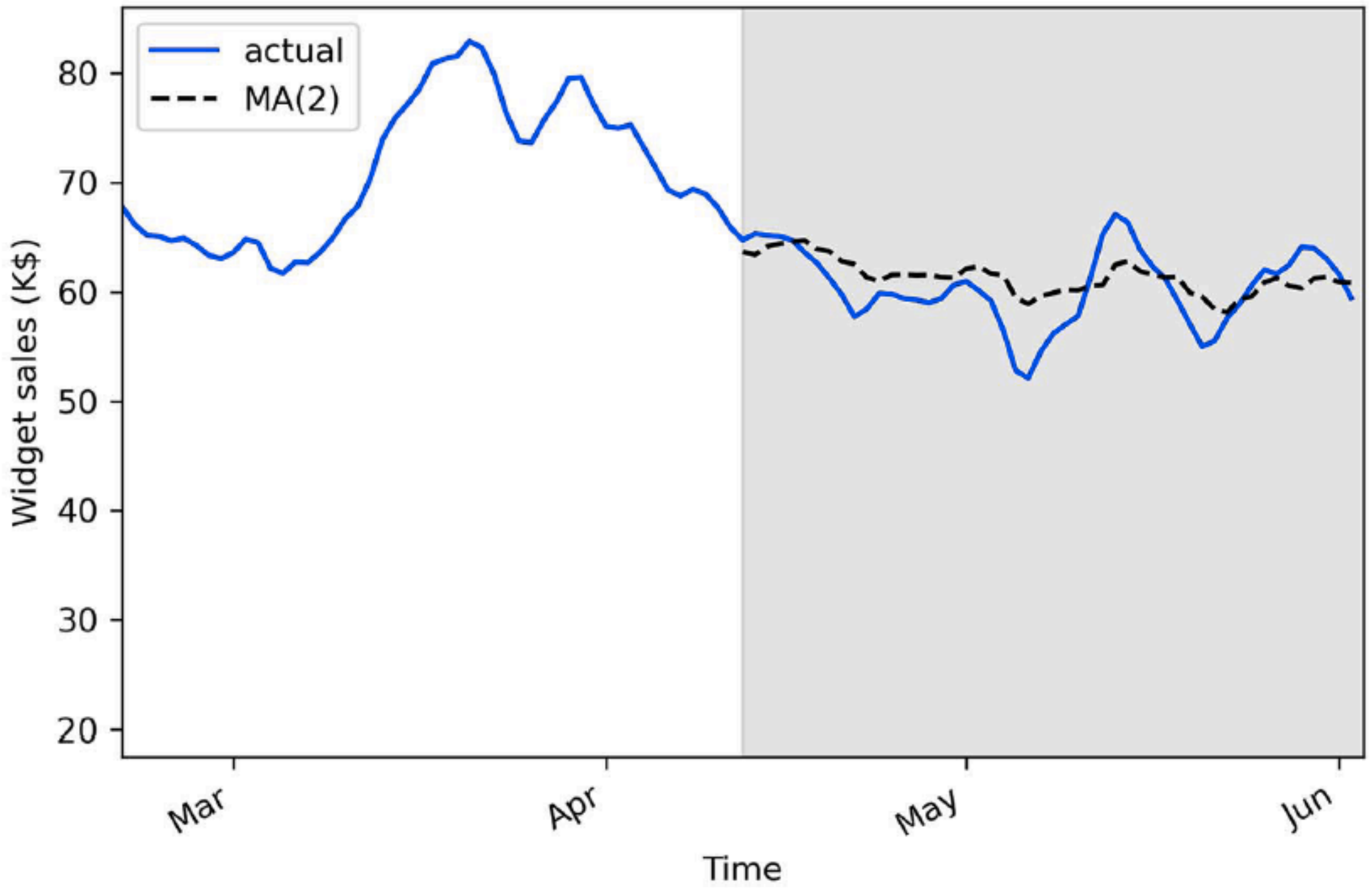
In order to reverse the first-order difference it is necessary to add an initial value y_0 to the first differenced value y_1'

$$y_1 = y_0 + y_1'$$

Then y_2 can be obtained using a cumulative sum of the differenced values

$$y_2 = y_0 + y_1' + y_2'$$

Inverse-transformed MA(2) series



MAE MA(2): 2.32

AutoRegressive
AR(p)

Autoregressive model

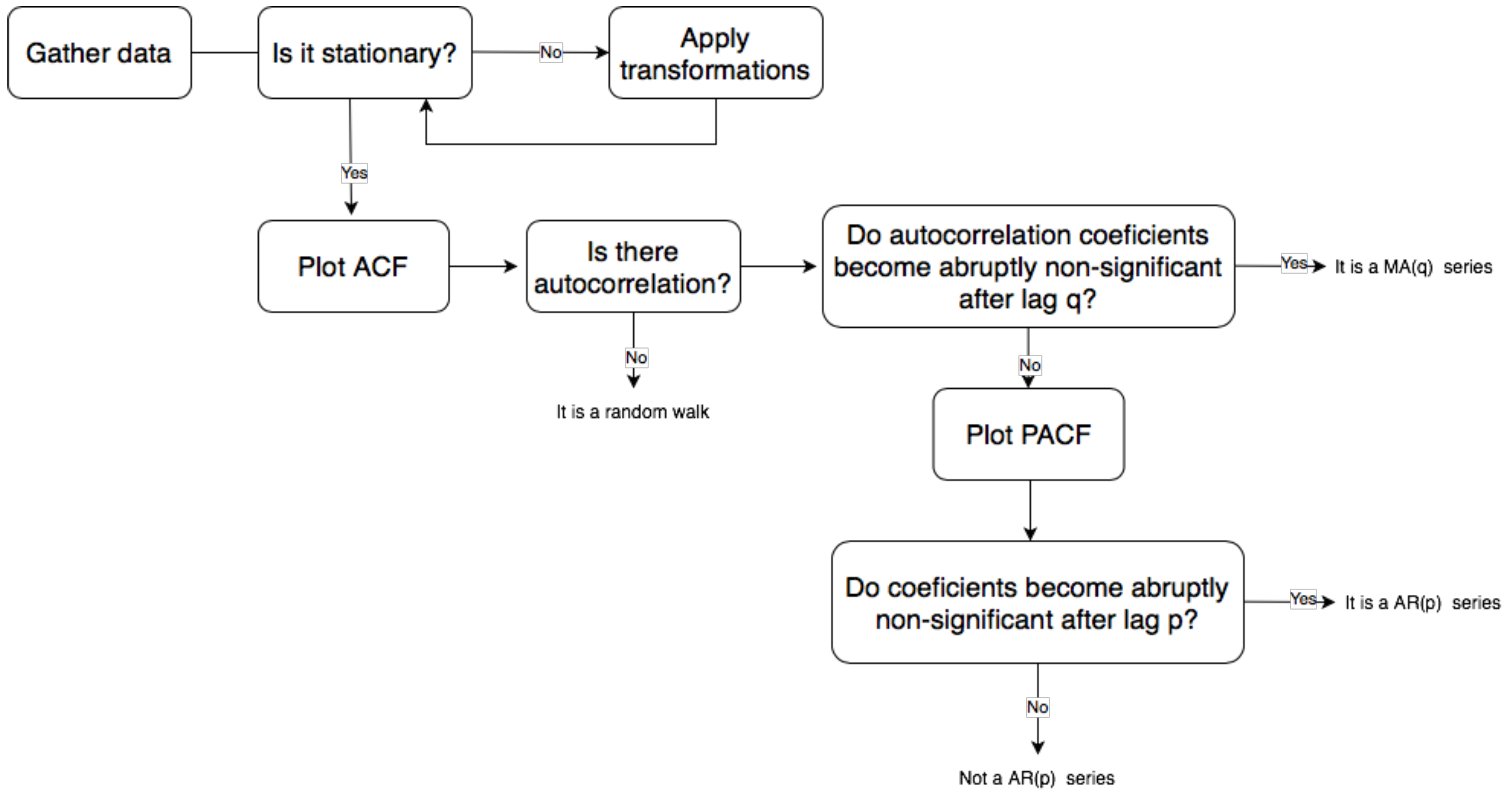
- An autoregressive model is a regression of a variable against itself. In a time series, this means that the present value is linearly dependent on its past values
- The autoregressive process is denoted as AR(p), where p is the order

The general expression of an AR(p) model is

$$y_t = C + \phi_1 y_{t-1} + \phi_2 y_{t-2} + .. + \phi_p y_{t-p} + \epsilon_t$$

Similar to the MA(q) series, the order p of an autoregressive process determines the number of past values that affect the present value.

Identifying a Autoregressive series



Partial Autocorrelation function (PACF)

In a second-order autoregressive series or AR(2)

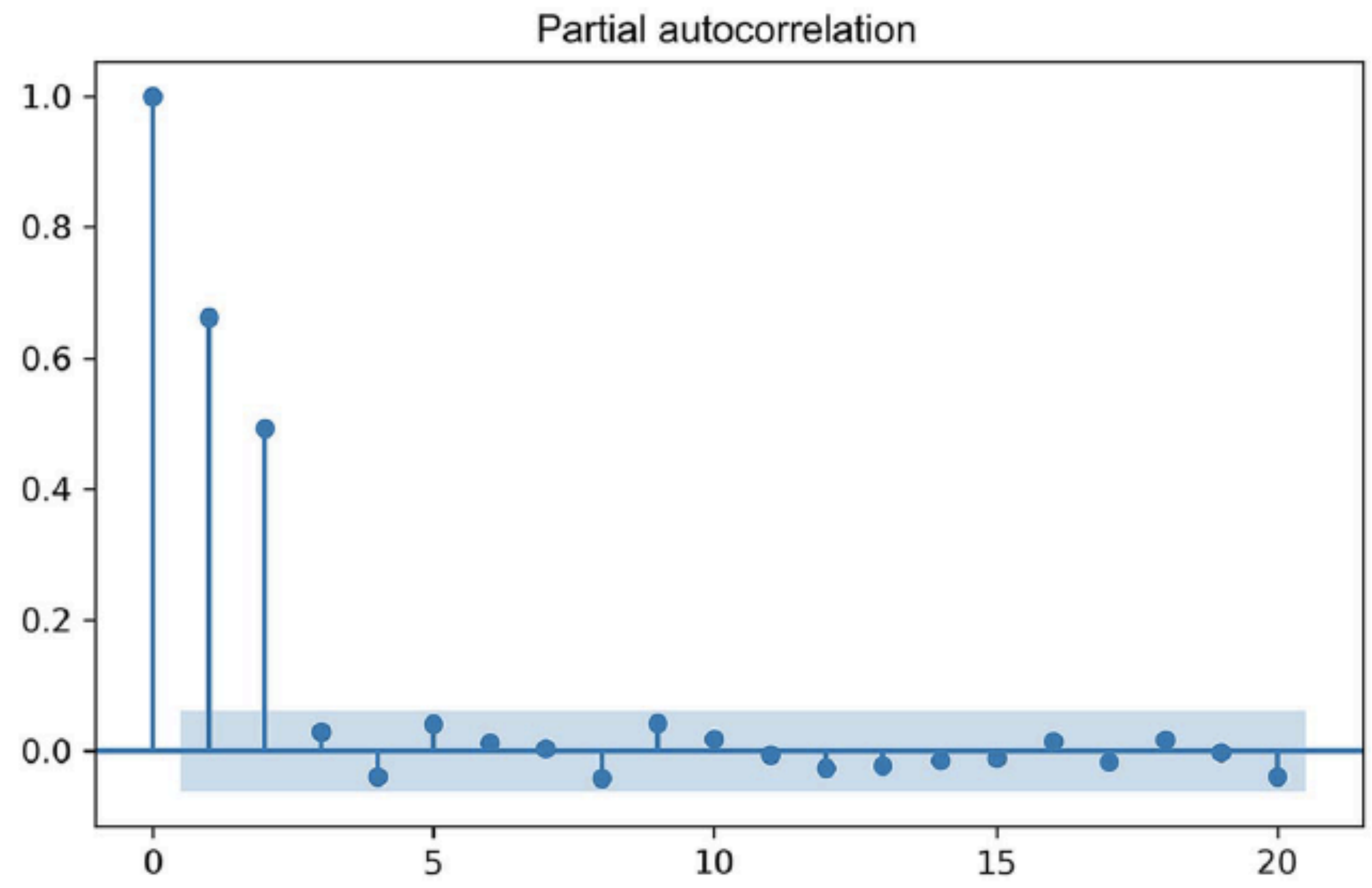
$$y_t = C + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \epsilon_t$$

The autocorrelation between yt and yt-2 using the ACF does not take into account the fact that y_{t-1} has an influence on both y_t and y_{t-2}

To do so, it is necessary to remove the effect of y_{t-1}. Thus, measuring the **partial autocorrelation** between y_t and y_{t-2}

The partial autocorrelation function measures the correlation between lagged values in a time series when the influence of correlated lagged values in between are removed

PACF plot of a AR(2) series



The partial autocorrelation function can be used to determine the order of a stationary AR(p) series - the coefficients will be non-significant after lag p

AutoRegressive Moving Average ARMA(p,q)

Autoregressive moving average series

- An autoregressive moving average series is a combination of the autoregressive and the moving average series
- It is denoted as ARMA(p,q), where p is the order of the autoregressive process, and q is the order of the moving average process

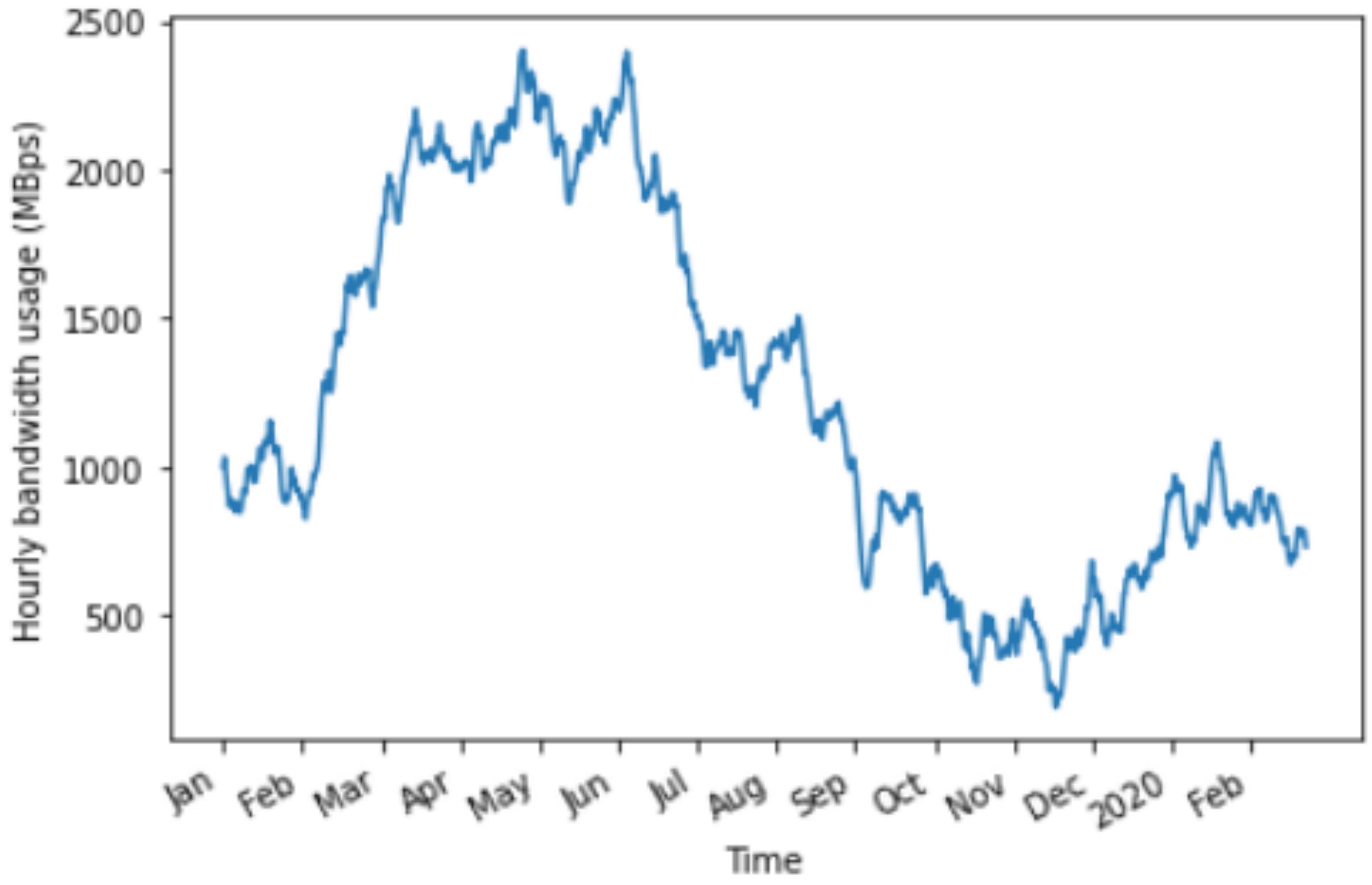
The general equation of the ARMA(p,q)

$$y_t = C + \phi_1 y_{t-1} + \phi_2 y_{t-2} + .. + \phi_p y_{t-p} + \mu + \epsilon_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + .. + \theta_q e_{t-q}$$

ARMA(0,q) ≈ MA(q), since the order p = 0 cancels the AR(p) portion

ARMA(p,0) ≈ AR(p), since the order q = 0 cancels the MA(q) portion

Autoregressive moving average series example



Bandwidth usage for a large data center in bits per second (bps)

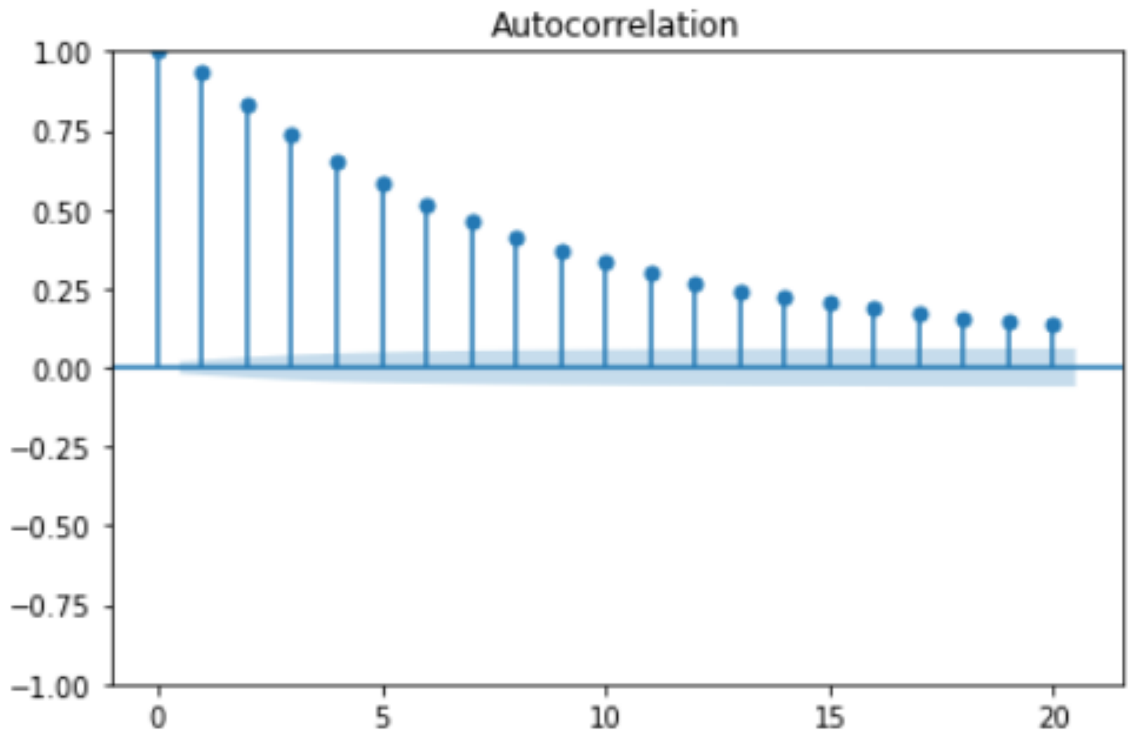
Identifying a stationary ARMA series

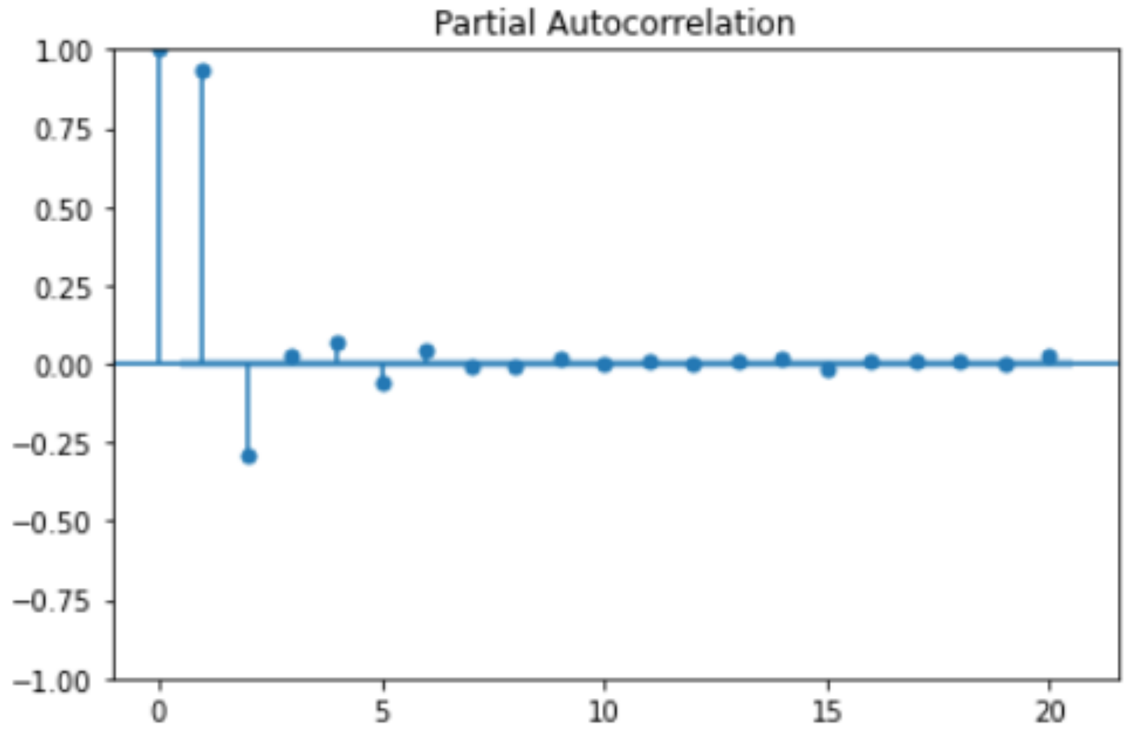
- If the series is a stationary ARMA(p,q) process both the ACF and PACF plots show a decaying or sinusoidal pattern

- The **ACF and PACF plots cannot** be used to determine the orders q and p of an ARMA(p,q) process

- The solution is to fit many ARMA(p,q) models with various combinations of values for p and q, then choosing a model using the **Akaike information criteria**

ACF and PACF plots





Akaike information criterion (AIC)

- Is a metric that aims to find a balance between a model's maximum likelihood and a model's simplicity

The AIC is a function of the number of parameters k in a model and the maximum value of the likelihood function $\hat{L}$:

$$AIC = 2k - 2\ln(\hat{L})$$

- The AIC quantifies the relative amount of information lost by the model
- The **better the model**, the **lower the AIC value** and the less information is lost

Function to fit several ARMA(p,q) models

```
def optimize_ARMA(data, order_list) -> pd.DataFrame:

    results = []

    for order in order_list:
        try:
            model = SARIMAX(data, order=(order[0], 0, order[1]), simple_differencing=False)
        except:
            continue

        aic = model.aic
        results.append([order, aic])

    result_df = pd.DataFrame(results)
    result_df.columns = ['(p,q)', 'AIC']

    #Sort in ascending order, lower AIC is better
    result_df = result_df.sort_values(by='AIC', ascending=True).reset_index(drop=True)

    return result_df
```

Residual analysis

The residuals of a model are simply the difference between the predicted values and the actual values

If the model has captured all predictive information from a dataset, the residuals of the model are **white noise**; there is only a random fluctuation left that cannot be modelled

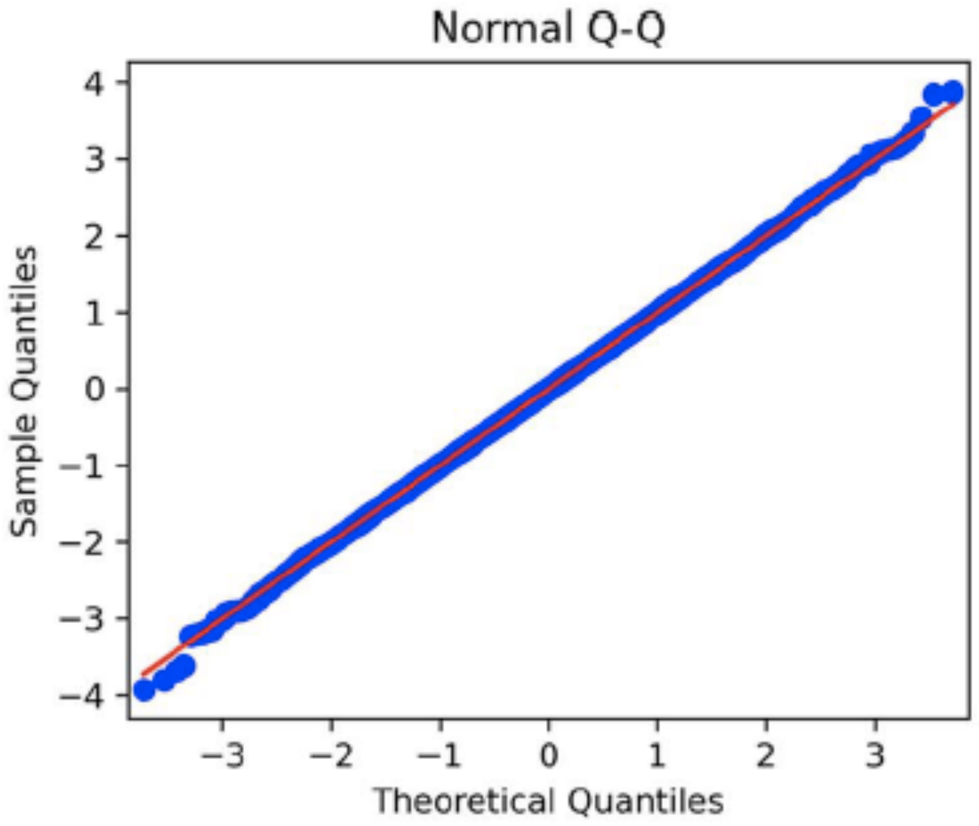
To have a good model for making forecasts, the **residuals** must be **uncorrelated** and have a **normal distribution**

Two residual analysis

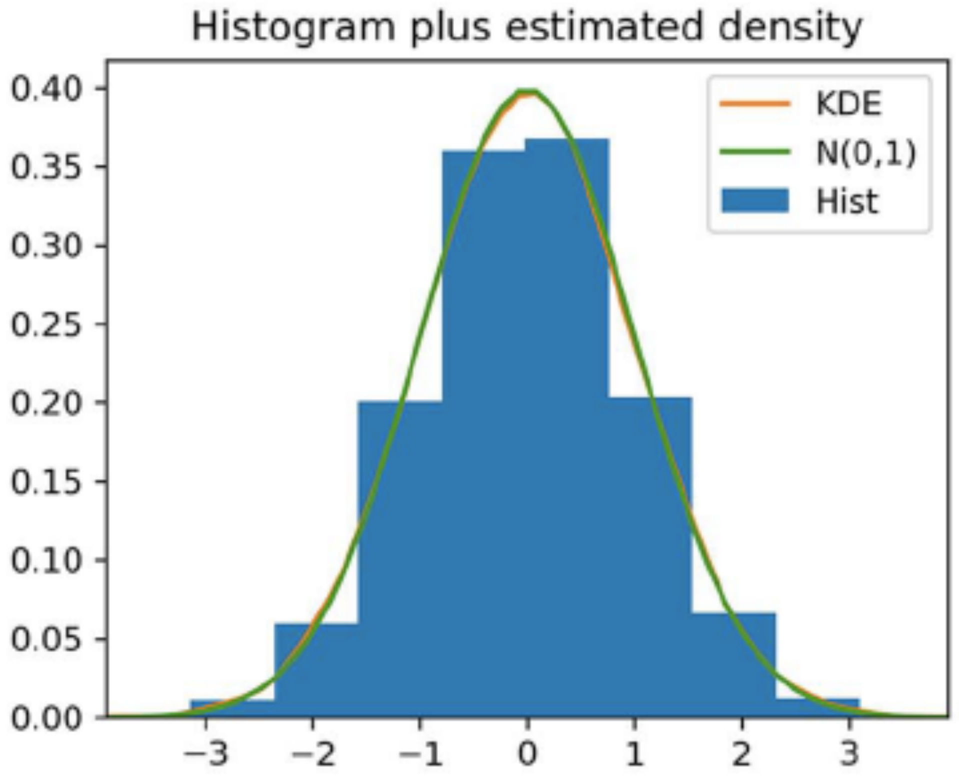
- A qualitative analysis through the study of the ***quantile-quantile plot** (*Q-Q plot), for verifying if the model’s residuals are normally distributed

- A quantitative analysis applying the ***Ljung-Box test*** to demonstrate that the residuals are uncorrelated

Two residual analysis: quantile-quantile plot

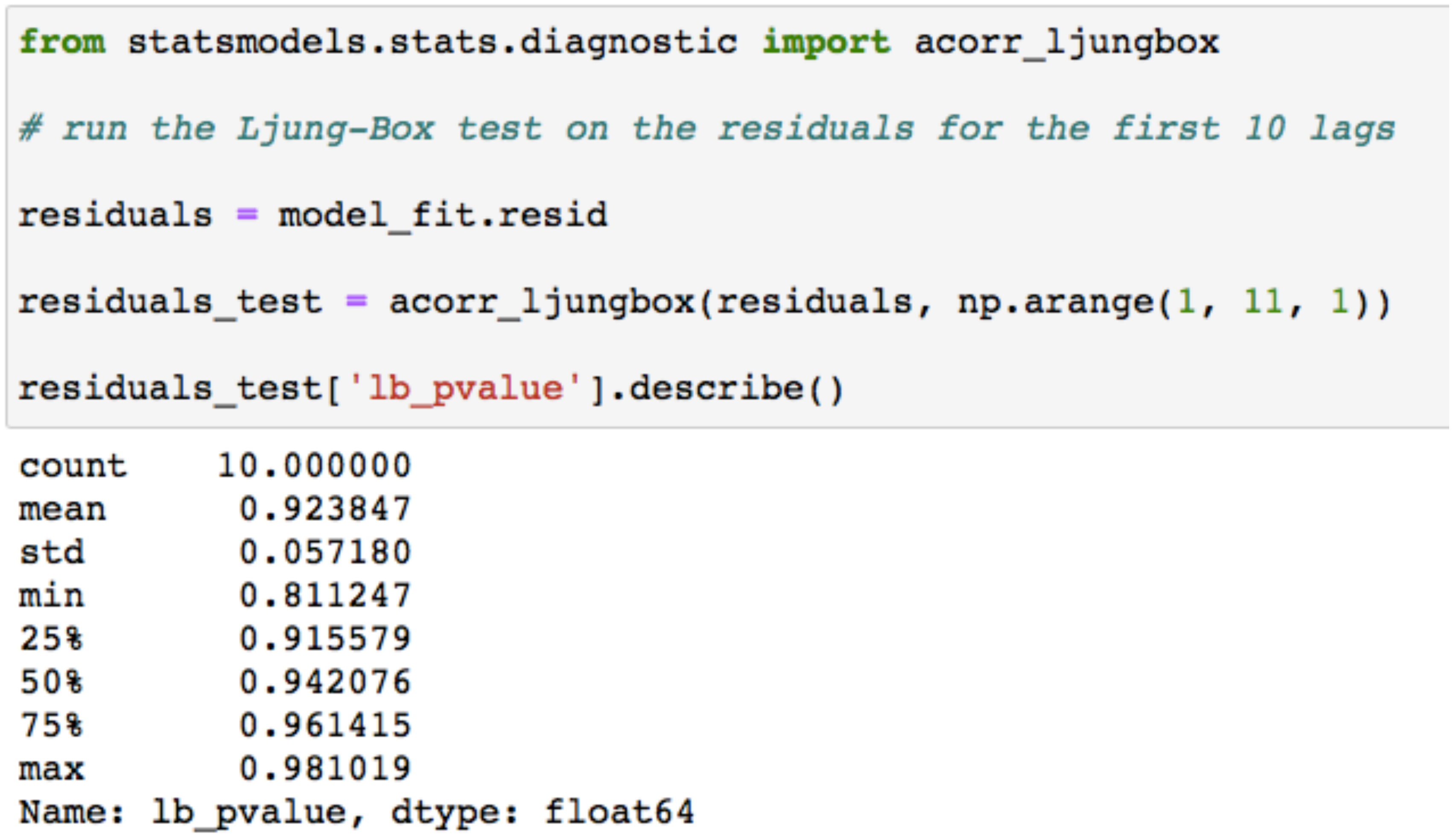


Q-Q plot of the residuals



Histogram of the residuals

Two residual analysis: Ljung-Box test



```python
from statsmodels.stats.diagnostic import acorr_ljungbox

# run the Ljung-Box test on the residuals for the first 10 lags

residuals = model_fit.resid

residuals_test = acorr_ljungbox(residuals, np.arange(1, 11, 1))

residuals_test['lb_pvalue'].describe()
```

```
count    10.000000
mean      0.923847
std       0.057180
min       0.811247
25%       0.915579
50%       0.942076
75%       0.961415
max       0.981019
Name: lb_pvalue, dtype: float64
```

All the returned p-values exceed 0.05, the residuals are uncorrelated

Non-stationary time series

Exponential Smoothing

Exponential Smoothing

- Time series forecasting method that uses an exponentially weighted average of past observations to predict future values
- The weights are then used to calculate a weighted moving average of the data, which is used as the forecast for the next period

- It is particularly useful for **short to medium-term** forecasting

There are three types of exponential smoothing methods:

- Single Exponential Smoothing (SETS)
- Double Exponential Smoothing (DETS)
- Triple Exponential Smoothing (TETS)

Single Exponential Smoothing (SETS)

- Adequate to univariate series with **no trend** and **no seasonal** pattern
- Needs a single parameter alpha (α) - **the smoothing factor**, that controls the rate at which the influence of past observations decreases exponentially, 0 < α < 1

The simple exponential smoothing formula is given by:

$$y_t = y_{t-1} + \alpha(x_t - y_{t-1})$$

where

y_t = smoothed statistic (simple weighted average of current observation x_t)

$\mathrm{y}_{t-1} = \text{ previous smoothed statistic}$

Double Exponential Smoothing (DETS)

- Adequate to time-series with a linear trend, but no seasonal pattern
- In addition to the alpha parameter (α) it needs another smoothing factor called beta (β), $0 < \beta < 1$, which controls the decay of the influence of change in trend
- The method supports trends that change in additive ways (smoothing with linear trend) and trends that change in multiplicative ways (smoothing with exponential trend)

The double exponential smoothing formulas are:

$s_1 = x_1$

$b_1 = x_1 - x_0$

$$y_t = \alpha x_t + (1-\alpha)(y_{t-1} + b_{t-1})$$

$$\beta_t = \beta(y_t - y_{t-1}) + (1-\beta) \times b_{t-1}$$

Where

b_t = best estimate of the trend at time t

Triple Exponential Smoothing (TETS)

- Adequate to time-series with a linear trend and seasonal patterns
- The technique applies exponential smoothing three times – level smoothing, trend smoothing, and seasonal smoothing. A new smoothing parameter called gamma ($\boldsymbol{\delta}$), 0 < $\boldsymbol{\delta}$ < 1 is added to control the influence of the seasonal component

The triple exponential smoothing formulas are given by:

$s_0 = x_0$ $\quad$ b_t = best estimate of a trend at time t

$$y_t = \alpha \frac{x_t}{c_{t-L}} + (1-\alpha)(y_{t-1} + b_{t-1})$$

$$t_t = \beta(y_t - y_{t-1}) + (1-\beta) \times b_{t-1}$$

$$c_t = \gamma \frac{x_t}{y_t} + (1-\gamma) \times c_{t-L}$$

Key limitations of Exponential Smoothing

- Sensitivity to smoothing parameters

- Inability to handle:
  - complex patterns, anomalies
  - sudden level or trend changes
  - outliers or abrupt seasonality

- Lagging Behind Trends
  - is biased towards past data, especially when the smoothing factor is low. This means it may not react quickly enough to new information or changes in the data

AutoRegressive **Integrated**
Moving Average
ARIMA(p,**d**,q)

Autoregressive integrated moving average series

- An autoregressive integrated moving average series is denoted as ARIMA(p,d,q), where p is the order of the AR(p) process, **d is the order of integration**, and q is the order of the MA(q) process

- Integration is the reverse of differencing, and the order of integration d is equal to the number of times the series has been differenced to be rendered stationary

The general equation of the ARIMA(p,d,q)

$$y'_t = C + \varphi_1 y'_{t-1} + .. + \varphi_p y'_{t-p} + \theta_1 \epsilon'_{t-1} + .. + \theta_q \epsilon'_{t-q} + \epsilon_t$$

General modelling procedure for using the ARIMA(p,d,q)

- Gather data
- If data is not stationary
  - d = the minimum number of times the series must be differenced to become stationary
- Fit several combinations ARIMA(p, d, q)
- Select the model with lowest AIC
- Make the residual analysis of the model:
  - with Q-Q plot to evaluate whether the residuals are normally distributed
  - with Ljung-Box test to determine whether the residuals are correlated or not
- If the ARIMA(p,d,q) model has passed all the checks on the residual analysis it can be used to forecast the series

Seasonal AutoRegressive Integrated Moving Average $SARIMA(p,d,q)(P,D,Q)_m$

Seasonal autoregressive integrated moving average series

- The SARIMA(p,d,q)(P,D,Q)$_m$ model adds seasonal parameters to the ARIMA(p,d,q) model
- It has four new parameters: P, D, Q, and m
- The parameter **m** is the **frequency**
- P is the order of the seasonal AR(P) process, D is the seasonal order of integration, Q is the order of the seasonal MA(Q) process
- A SARIMA(p,d,q)(0,0,0)m model is equivalent to an ARIMA(p,d,q) model

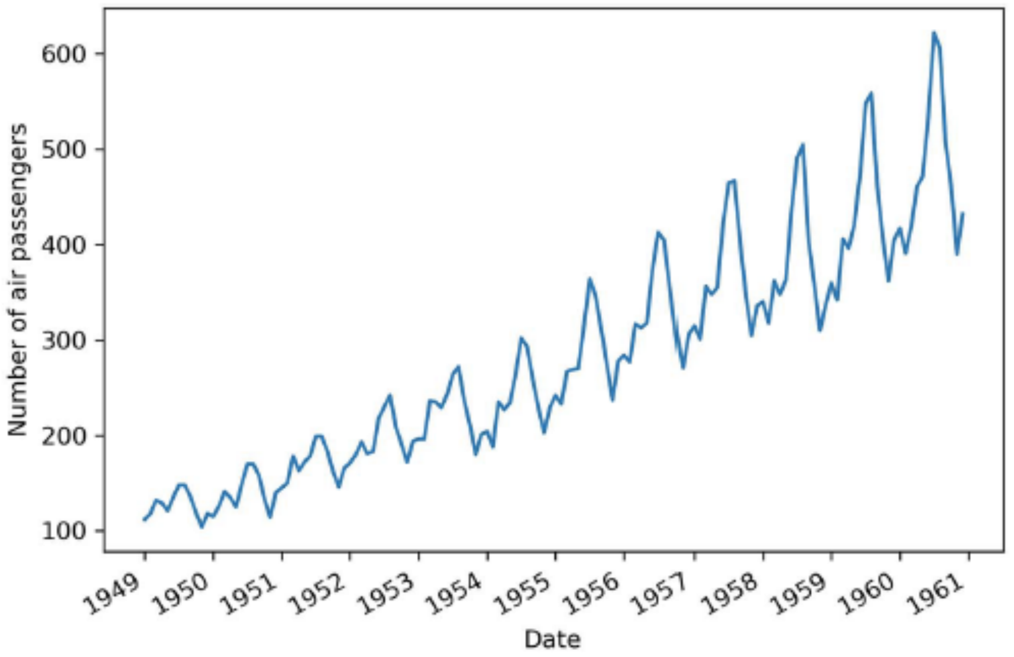
Parameter m — Frequency

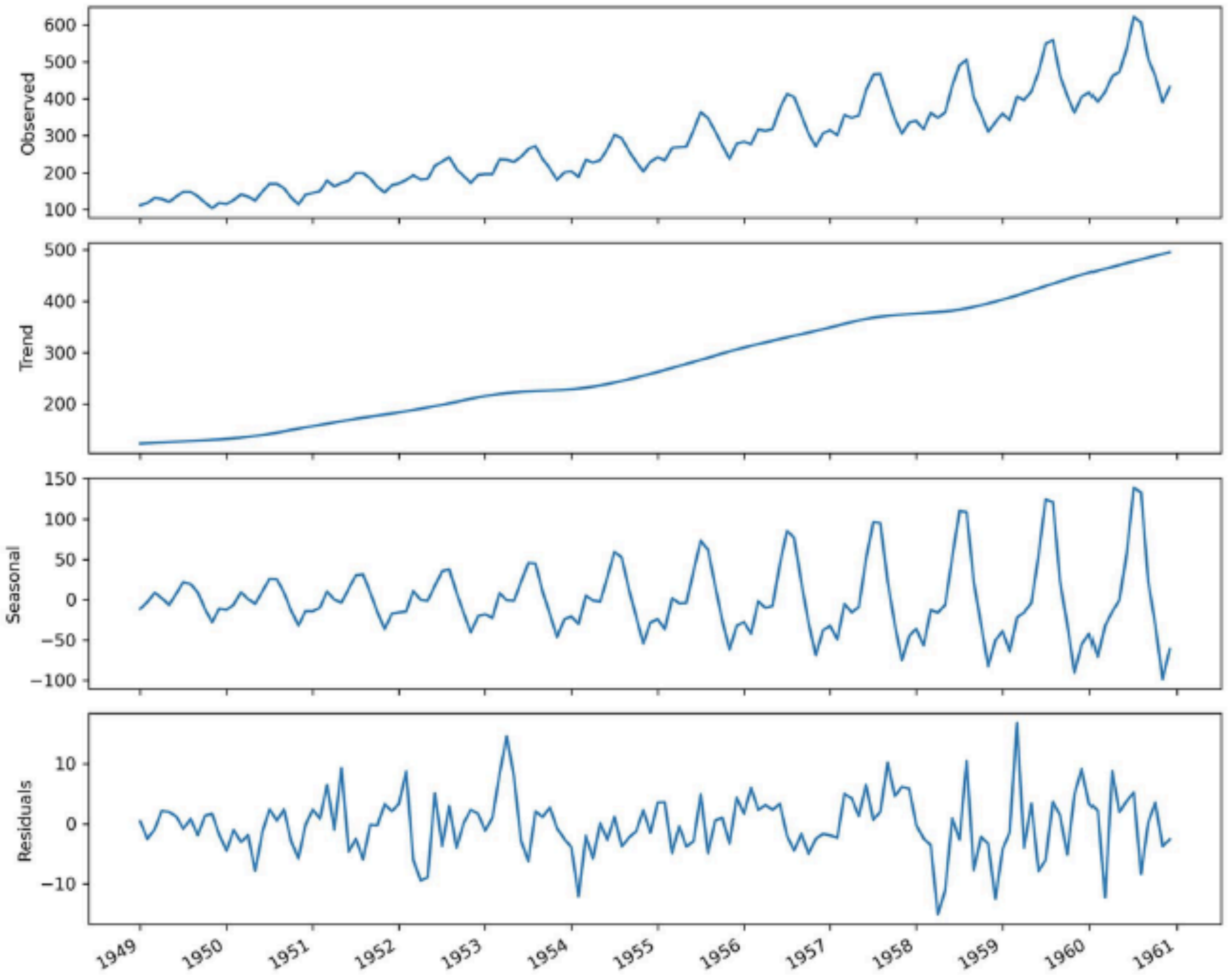
- The frequency is defined as the number of observations per cycle
- The length of the cycle will depend how the data was recorded

Data collection	Frequency	Frequency		
		day	**week**	**year**
Annual	1			
Quarterly	4			
Monthly	12			
Weekly	52			
Daily		1	7	365
Hourly		24	168	8766

Identifying seasonal patterns in a time series

By plotting the time series or using time series decomposition





Decomposition of a time series

The modelling of the decomposed components can be:

Additive: when the original time series can be reconstructed by adding all three components

$$y_t = T_t + S_t + R_t$$

Multiplicative: the time series can be reconstructed by multiplying all three components

$$y_t = T_t \times S_t \times R_t$$

Four new parameters: P, D, Q, and m

Example where m = 12

- If P=2, this means that we will be include two past values of the series at a lag that is a multiple of m, the values at y_{t-12} and y_{t-24}

- If D=1, this means that one seasonal difference makes the series stationary. The seasonal difference would be expressed as $y_t' = y_t - y_{t-12}$

- If Q=2, past error terms at lags that are a multiple of m will be included, the errors ϵ_{t-12} and ϵ_{t-24}

General modelling procedure for using the SARIMA

- Gather data
- Until the data is not stationary
  - make the difference of the data
  - make the seasonal difference of data

- When data is stationary d and D are defined

- Fit several combinations of p, q, P, D of SARIMA(p, d, q)(P,D,Q)$_m$
- Select the model with lowest AIC
- Make the residual analysis of the model
- If the SARIMA(p,d,q)(P,D,Q)$_m$ model has passed all the checks on the residual analysis it can be used to forecast the series

SARIMAX$(p,d,q)(P,D,Q)_m$

X – exogenous variables

SARIMAX model

- Extends the SARIMA(p,d,q)(P,D,Q)$_m$ model by adding the effect of exogenous variables
- The present value y$_t$ is expressed as a SARIMA(p,d,q)(P,D,Q)$_m$ model to which there are added any number of exogenous variables — X$_t$

The general equation of the SARIMAX(p,d,q)(P,D,Q)$_m$

$$y_t = SARIMA(p, d, q)(P, D, Q)_m + \sum_{i=1}^{n} \beta_i X_i^t$$

SARIMAX model: exogenous variables

- The exogenous variables may or may not be good predictors
- However, it is not necessary to perform feature selection
- The SARIMAX model will attribute a coefficient close to 0 for exogenous variables that are not significant in predicting the target

SARIMAX model: number of timesteps to forecast

- The **SARIMAX model** uses the SARIMA(p,d,q)(P,D,Q)$_m$ model and a linear combination of exogenous variables to predict **one timestep into the future**
- To predict **more timesteps into the future** the SARIMAX model requires to forecast the exogenous variables too

- There is no clear recommendation to predict only one timestep. It is dependent on the exogenous variables available:
  - If the exogenous variables can be accurately predicted, it is possible to forecast many timesteps into the future
  - if the forecast of exogenous variables have some error associated they can ampliated the prediction error of the target, as more timesteps are predicted into the future

General modelling procedure for using the SARIMAX

- Gather data
- Until the data is not stationary
    - make the difference of the data
    - make the seasonal difference of data

- When data is stationary d and D are defined

- Fit several combinations of p, q, P, D of SARIMAX(p, d, q)(P,D,Q)$_m$
- Select the model with lowest AIC
- Make the residual analysis of the model
- If the SARIMA(p,d,q)(P,D,Q)$_m$ model has passed all the checks on the residual analysis it can be used to forecast the series

Vector AutoRegression (VAR) Multivariate Forecasting

Vector Autoregression model

- The vector autoregressive model VAR(p) is a generalization of the AR(p) model that allows the forecast of multiple time series

- In this model, each time series has an impact on the others, the past values of one time series affect the other time series, and vice versa

- The order p of the VAR(p) model determines how many lagged values impact the present value of the series

Vector Autoregression model

For two time series, the general equation for the VAR(p) model is a linear combination of a vector of constants, past values of both time series, and a vector of error terms:

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} + \begin{bmatrix} \phi^1_{1,1} & \phi^1_{1,2} \\ \phi^1_{2,1} & \phi^1_{2,2} \end{bmatrix} \begin{bmatrix} y_{1,t-1} \\ y_{2,t-1} \end{bmatrix} + \begin{bmatrix} \phi^2_{1,1} & \phi^2_{1,2} \\ \phi^2_{2,1} & \phi^2_{2,2} \end{bmatrix} \begin{bmatrix} y_{1,t-2} \\ y_{2,t-2} \end{bmatrix} + \ldots + \begin{bmatrix} \phi^p_{1,1} & \phi^p_{1,2} \\ \phi^p_{2,1} & \phi^p_{2,2} \end{bmatrix} \begin{bmatrix} y_{1,t-p} \\ y_{2,t-p} \end{bmatrix} + \begin{bmatrix} \epsilon_{1,t} \\ \epsilon_{2,t} \end{bmatrix}$$

The time series must be stationary to apply the VAR model

Granger causality test

- The VAR model can only be used if each of the time series are predictive of each other
- The **Granger causality test** determines whether one time series is predictive of another
- This statistical test helps to determine if past values of a time series $y_{2,t}$ can help to forecast time series $y_{1,t}$ => If true $y_{2,t}$ Granger-causes $y_{1,t}$
- The Granger causality test tests causality only in one direction
- In order for the VAR model to be valid, the test must be repeated to verify that $y_{1,t}$ also Granger causes $y_{2,t}$

Modelling procedure for the VAR(p) model

- Gather data
- If the series are not stationary
  - must be differentiated the number of times needed to become stationary
- Fit many VAR(p) models to select the one with the smallest AIC — the p with the lowest AIC value is the order of VAR(p) model
- Make the **Granger causality test** for both series
  - If p-value less than 0.05, the null hypothesis can be rejected, meaning that the variables are Granger cause each other
- Make the residual analysis of the two series:
  - with **Q-Q plot** to evaluate whether the residuals are normally distributed
  - with **Ljung-Box test** to determine whether the residuals are correlated or not
- If the VAR(p) model passed the residual analysis it can be used to forecast the series

Time series forecasting with Machine Learning

When to use ML/DL for time series forecasting

- When statistical models take too much time to fit or when they result in correlated residuals that do not approximate white noise

- This can be due to the fact that data has multiple seasonal periods that cannot be considered in the model, or simply because there is a nonlinear relationship between the features and the target

- In those cases, machine learning/deep learning models can be used to capture this nonlinear relationship
- These models also have the added advantage of being faster to train

Types of ML models for time series forecasting

Single-step model

- Predicts one step into the future for one variable

Multi-step model

- Predicts many steps into the future for one variable

Multi-output model

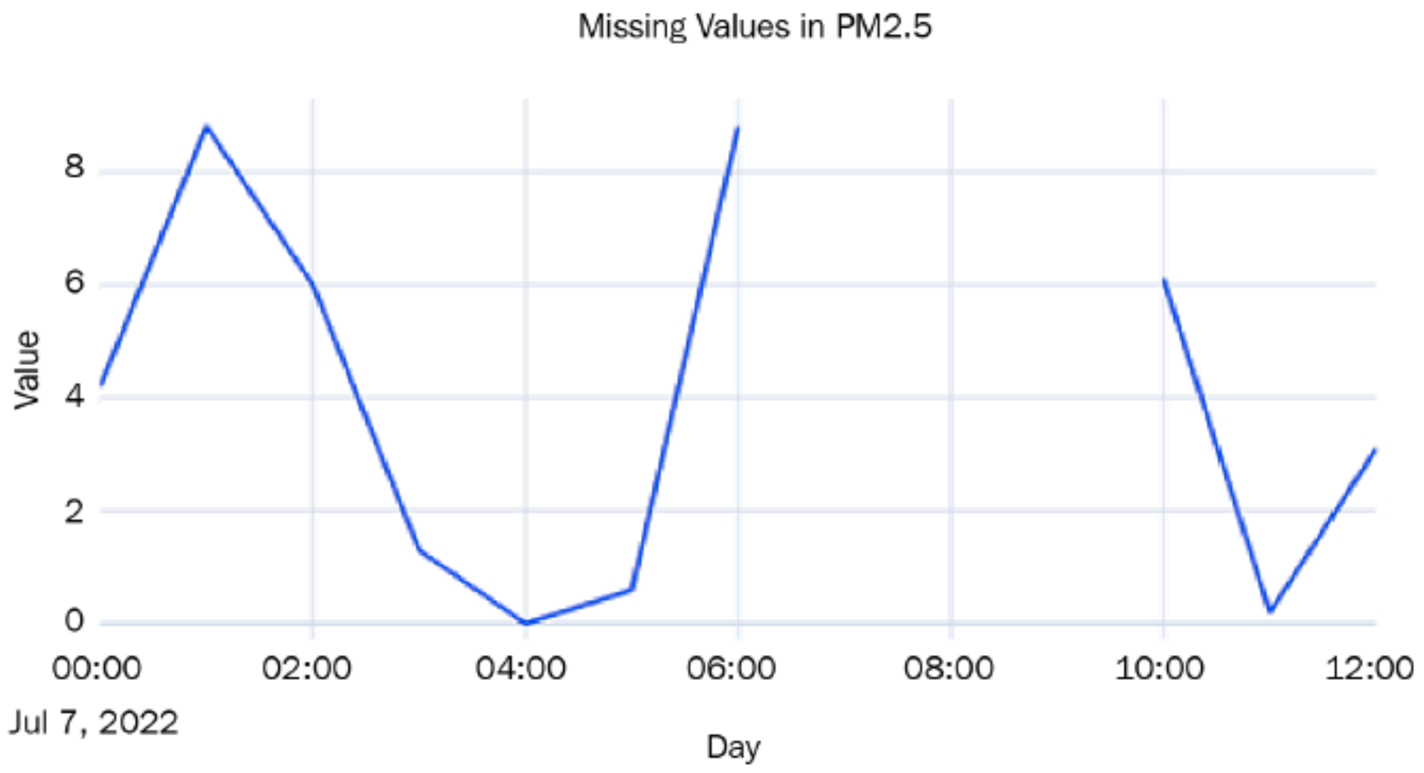
- Has more than one time dependent variable. Each variable not only depends on its past values but also has some dependency on other variables

Transform time series into a two-dimensional format

- Explore the time series: trend, seasonality, univariate and bivariate analysis
- Define its frequency
- Treat missing values
- Treat outliers
- Split data into train/test
- Scaling data
- Put the time series in a tabular format (slide window)

Handling missing values

Missing values are represented in the time series by a **sequence gap** in the time stamp variable or in other values



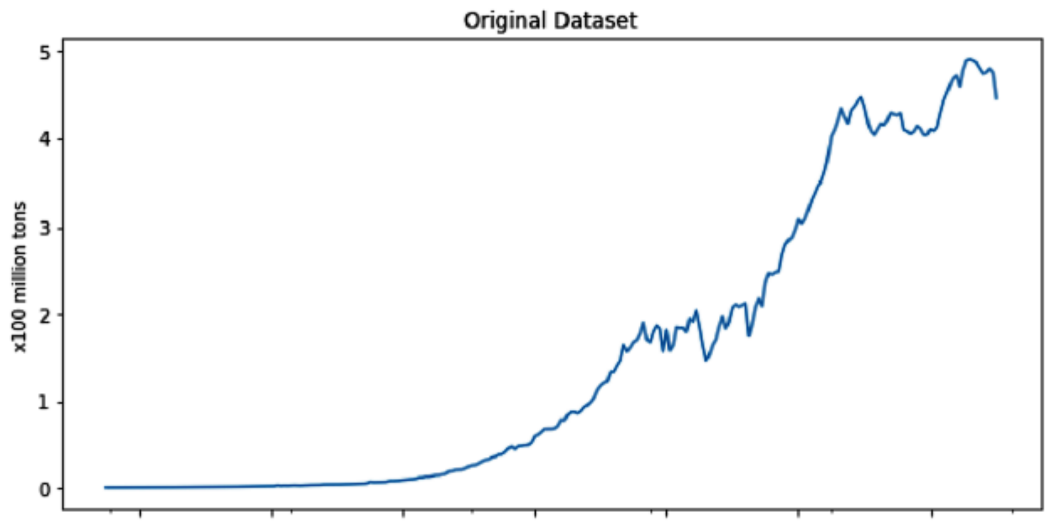
The first thing to do is to evaluate if the data in question is indeed missing or not

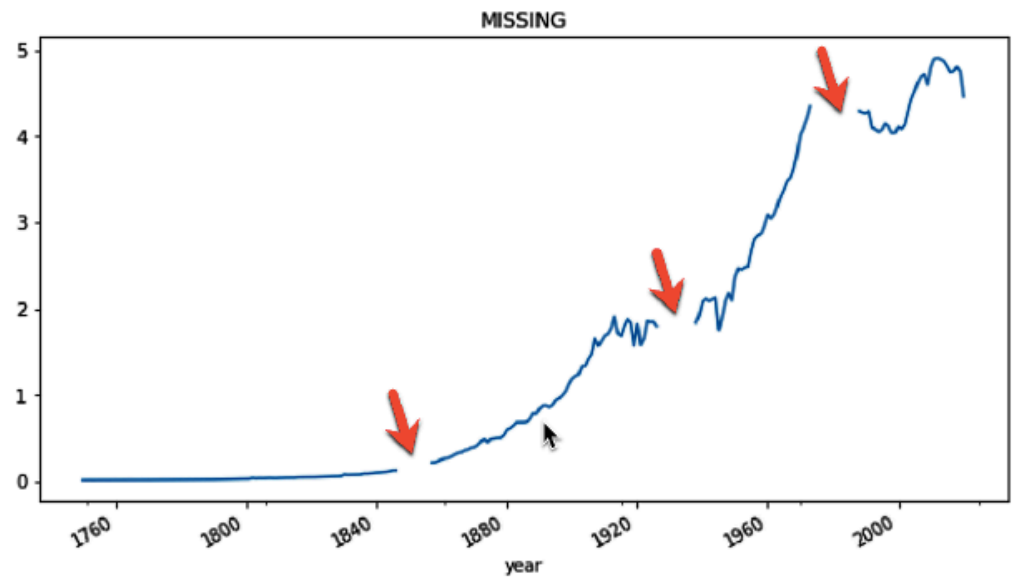
Techniques to fill the missing values

Simple techniques:

- **Forward Fill**: fill in the missing values with the last observed value until it finds the next observation
- **Backward Fill**: takes the next observation and backtracks to fill in all the missing values
- **Mean Value Fill**: fill in the missing values with the mean value of the entire series

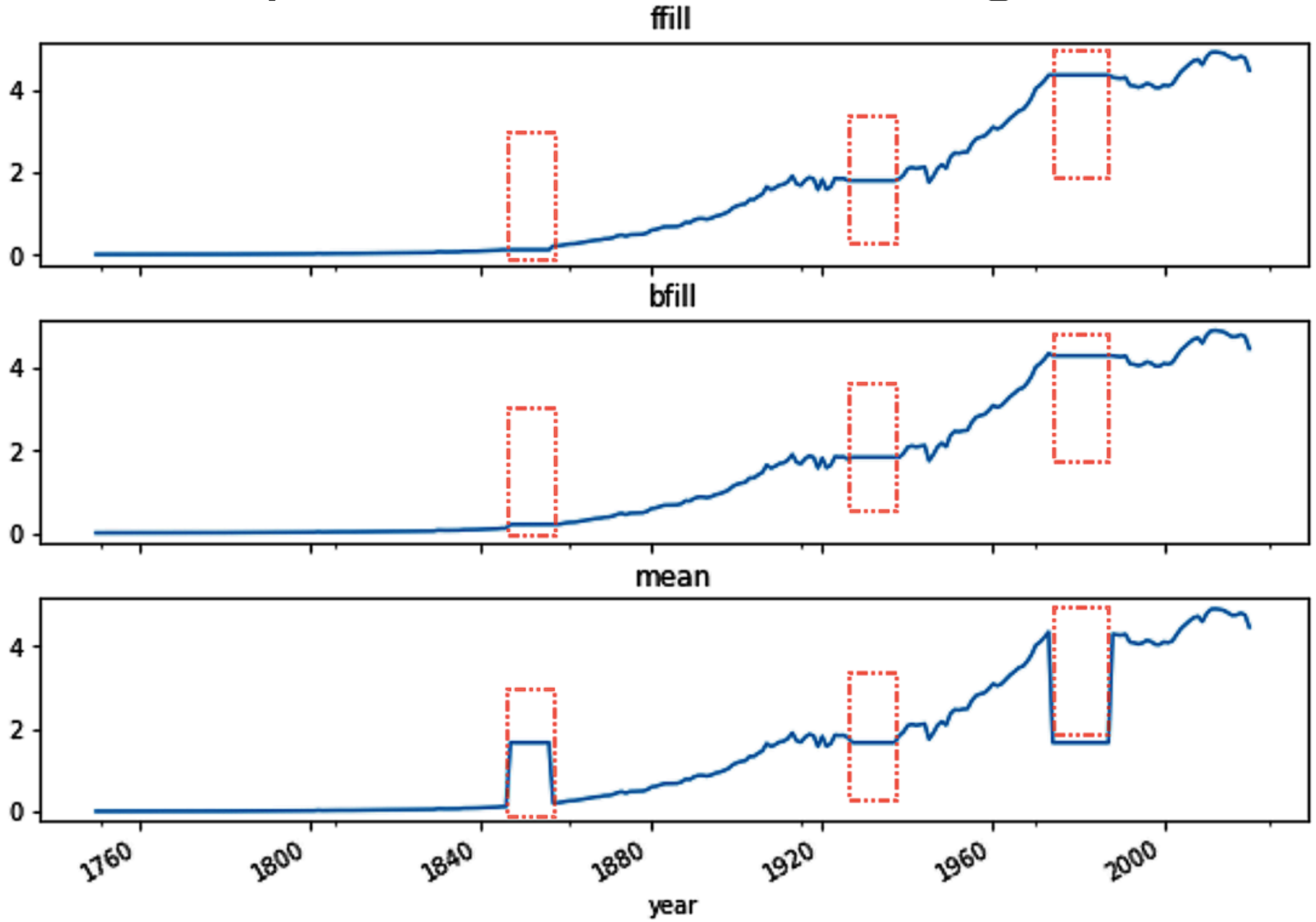
Techniques to fill the missing values





```
co2_missing['ffil'] = co2_missing['co2'].fillna(method='ffill')
co2_missing['bfill'] = co2_missing['co2'].fillna(method='bfill')
co2_missing['mean'] = co2_missing['co2'].fillna(co2_missing['co2'].mean())
```

Techniques to fill the missing values

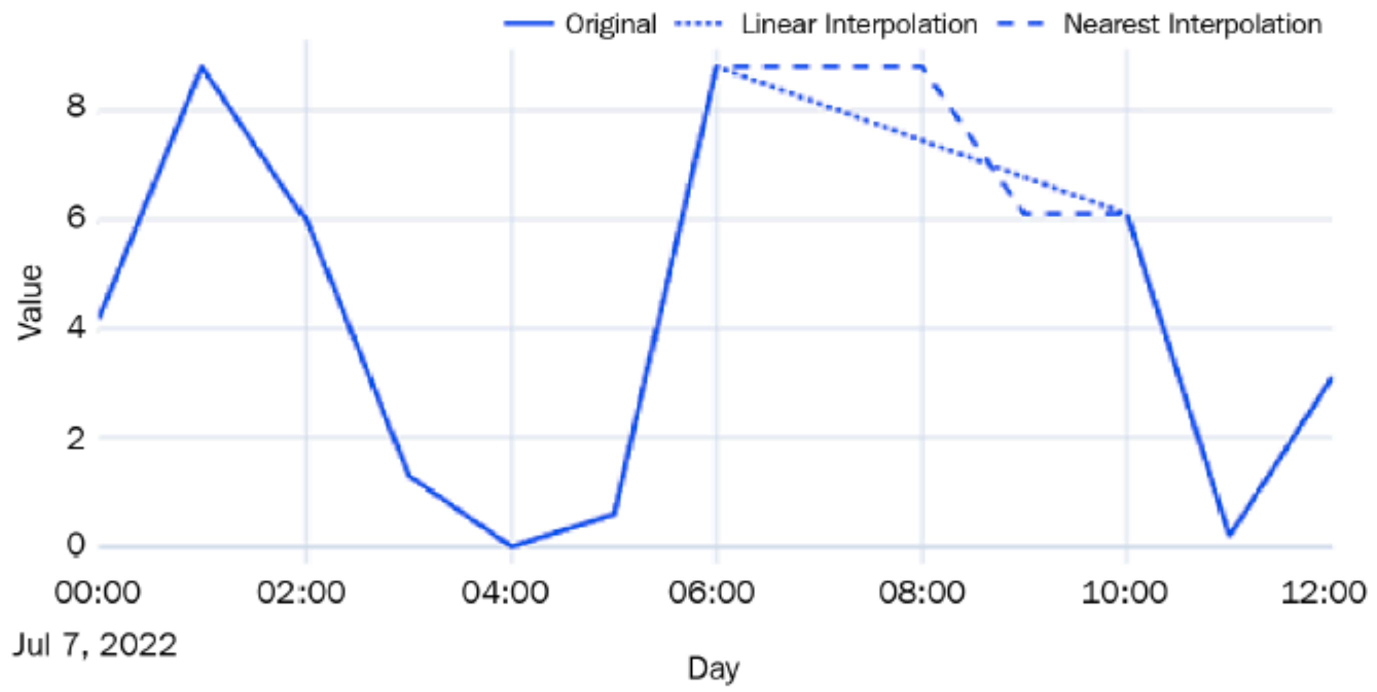


```
RMSE for ffil: 0.0587
RMSE for bfill: 0.0555
RMSE for mean: 0.7156
```

Techniques to fill the missing values

Other more complex techniques to fill in missing values covers interpolation:

- **Linear Interpolation**: draws a line between two observed points and fills the missing values so that they lie on this line
- **Nearest Interpolation**: is a combination of the forward and backward fill. For each missing value, the closest observed value is used to fill in

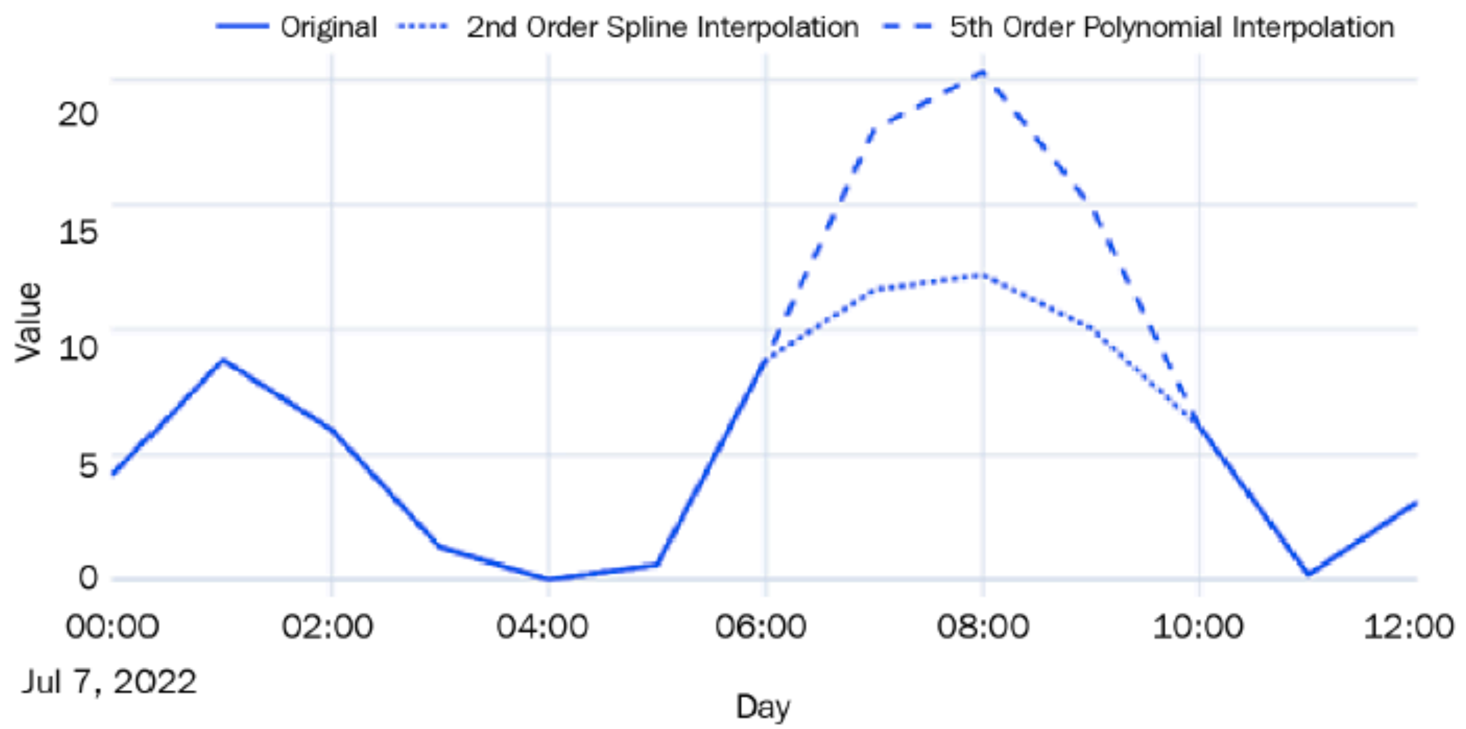


Techniques to fill the missing values

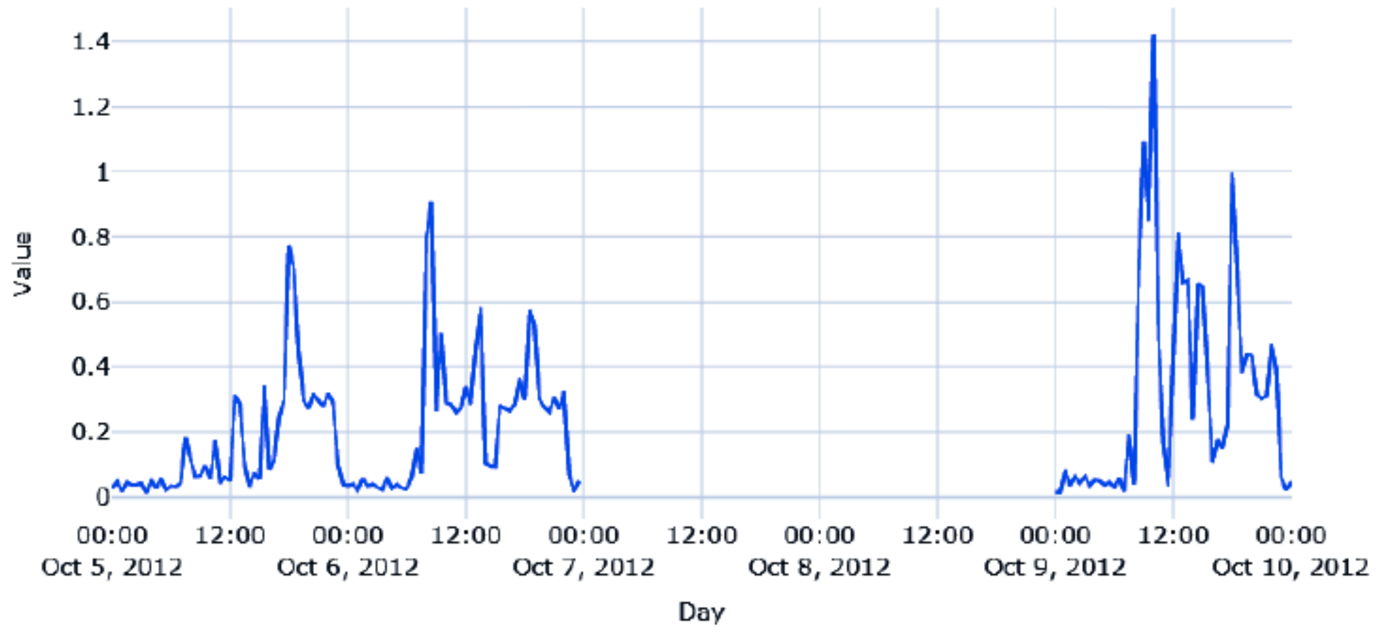
Non-linear interpolation techniques:

- **Spline, Polynomial**: fit a spline/polynomial of a given order to the data and use that to fill in missing values

The higher the order, the more flexible the function is to fit the observed points



Handling longer periods of missing data



- Imputing with the previous day
- Hourly average profile
- The hourly average for each weekday

Seasonal interpolation

1. Calculate the seasonal profile
2. Subtract the seasonal profile and apply any interpolation technique
3. Add the seasonal profile back to the interpolated series

Detection of Outliers

- **Visual Inspection:** plot the time series and look for sudden spikes or drops
- **Statistical Thresholds:** use statistical methods like Z-score, where values beyond a certain threshold (e.g., $|Z| > 3$) are considered outliers
- **Rolling Window Analysis:** calculate rolling mean and standard deviation. Values that deviate significantly from the rolling mean (e.g., beyond ±3 standard deviations) may be considered outliers
- **Interquartile Range (IQR):** Calculate the IQR for a moving window. Values outside the range of Q1 - 1.5 * IQR to Q3 + 1.5 * IQR can indicate outliers

Understand the cause of Outliers

- **Investigate Real-World Events**: check if the outlier corresponds to a real event (e.g., economic crisis, holiday, or weather anomaly). If so, it may carry valuable information rather than being a true anomaly

- **Distinguish Seasonal Effects from Outliers**: Seasonally recurring patterns should not be treated as outliers; verify with seasonal decomposition to distinguish true outliers from seasonal peaks and troughs

Outliers Treatment Options

Replace with Statistical Estimates:

- Mean/Median Imputation: replace the outlier with the mean/median of surrounding data points if the outlier is isolated
- Moving Average Imputation: Replace the outlier with the mean of nearby points within a rolling window
- Interpolation: Use linear or polynomial interpolation to replace the outlier, based on neighbouring values

Transformation Techniques:

- Winsorization: Cap extreme values by setting them at a certain percentile threshold (e.g., 5th and 95th percentiles)
- Log Transformation: Reduce the impact of outliers on the model by applying a logarithmic transformation if they cause high variance

Evaluate Impact on Model Performance

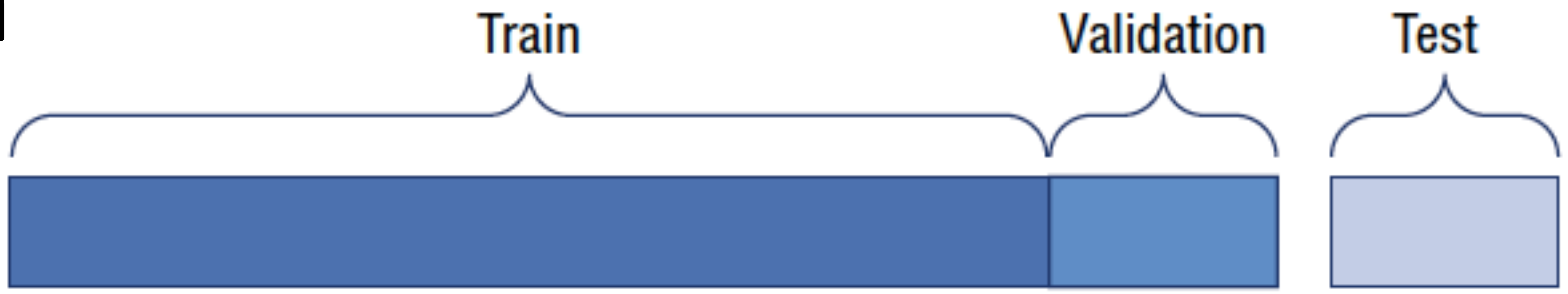
Re-run Model Training

- Compare the model’s performance with and without outlier adjustments. Measure metrics such as MAE, MAPE, or RMSE to see if handling outliers improves accuracy

Adjust Treatment as Needed:

- If the outlier treatment adversely affects accuracy or introduces biases, consider alternative methods

Split data



Split by Time

Since time matters in time series data, data should be split based on time, rather than randomly

Split Points

- Training set: must be the earlier portion of the time series, 70-80% of the data
- Validation set: the next 15-10% of the data after the training set
- Test set: the remaining data, the last 15-10%

If data has any seasonality or specific patterns for example weekly patterns, the splits must respect the pattern

Time Series Scaling/Normalizing

- **Scaling**
  - Min-Max
- **Normalization**
  - Sigmoid

Time series data transformation

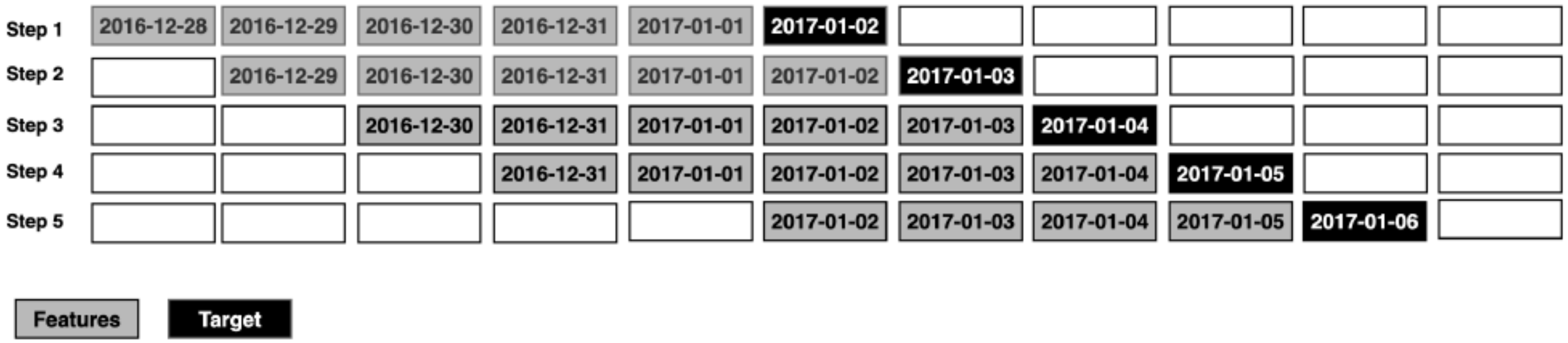
Time series to be processed by ML algorithms must be transformed to contain **independent** and **dependent variables**

For that, it is necessary to define a **sliding window** to convert the data into a window of inputs

For example, a **five-period sliding window** in a univariate time series, will produce five independent variables, x_1, x_2, x_3, x_4, x_5, which are lagged versions of the dependent variable, to be used to predict the current period x_6

This representation of multiple inputs (a sequence) to one output is referred to as a **single-step forecast**

Five-period sliding window for single-step forecast



Step 1	2016-12-28	2016-12-29	2016-12-30	2016-12-31	2017-01-01	2017-01-02					
Step 2		2016-12-29	2016-12-30	2016-12-31	2017-01-01	2017-01-02	2017-01-03				
Step 3			2016-12-30	2016-12-31	2017-01-01	2017-01-02	2017-01-03	2017-01-04			
Step 4				2016-12-31	2017-01-01	2017-01-02	2017-01-03	2017-01-04	2017-01-05		
Step 5						2017-01-02	2017-01-03	2017-01-04	2017-01-05	2017-01-06	

▨ Features ■ Target

This technique was described in Machine Learning Strategies for Time Series Forecasting, Lecture Notes in Business Information Processing. Springer Berlin Heidelberg (https://doi.org/10.1007/978-3-642-36318-4_3)

Multi-step forecast

- Predicts more than one step forecast into the future
- **Forecast horizon** is the number of time steps into the future to forecast at any point in time

Example:

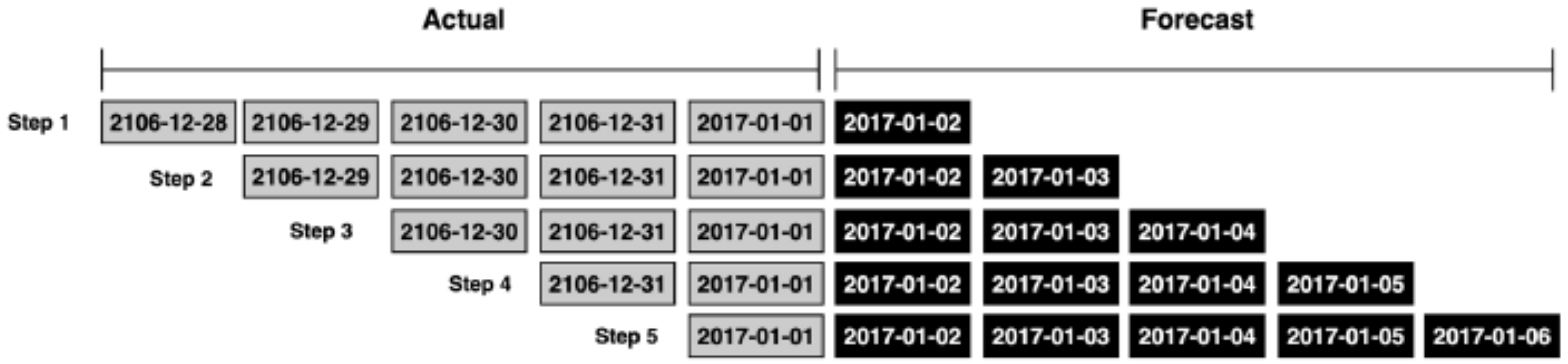
Prediction of three months into the future (x_{11}, x_{12}, x_{13})
based on a sequence of 10 previous months (x_1 , … , x_{10})

Four different ways to transform a series for multi-step forecast:

- **Recursive strategy**
- Direct strategy
- DirRec (Direct-Recursive) strategy

Recursive multi-step forecast strategy

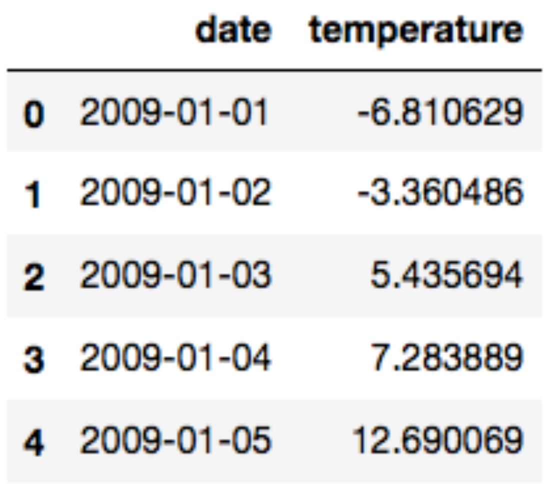
Is based on one-step forecasts that are reused (recursively) to make the next one-step prediction, and the process continues until all the future steps are produced



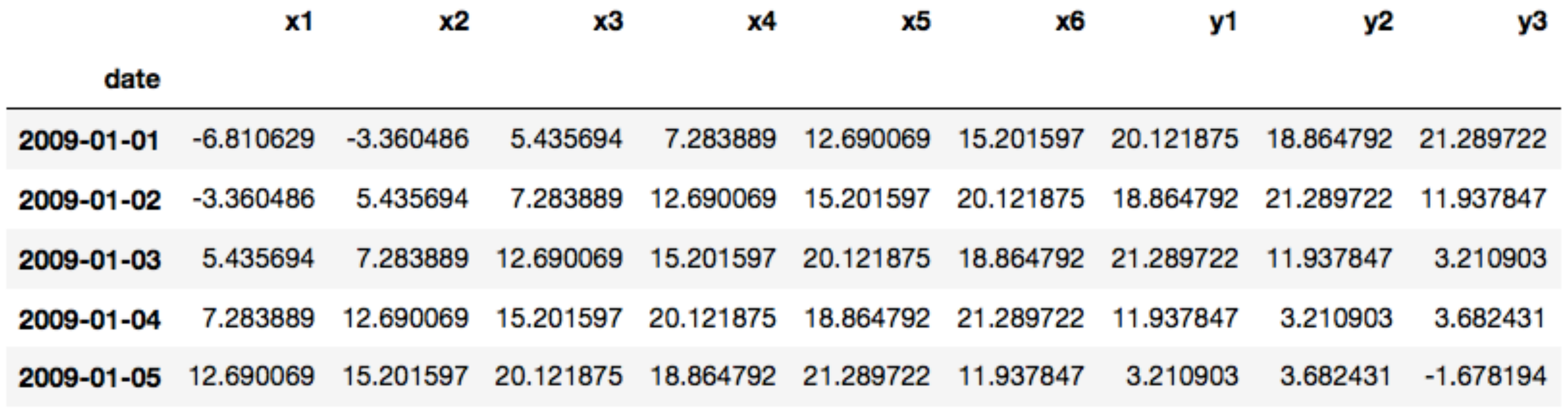
Slide window

Window = 6

Horizon = 3



	date	temperature
0	2009-01-01	-6.810629
1	2009-01-02	-3.360486
2	2009-01-03	5.435694
3	2009-01-04	7.283889
4	2009-01-05	12.690069

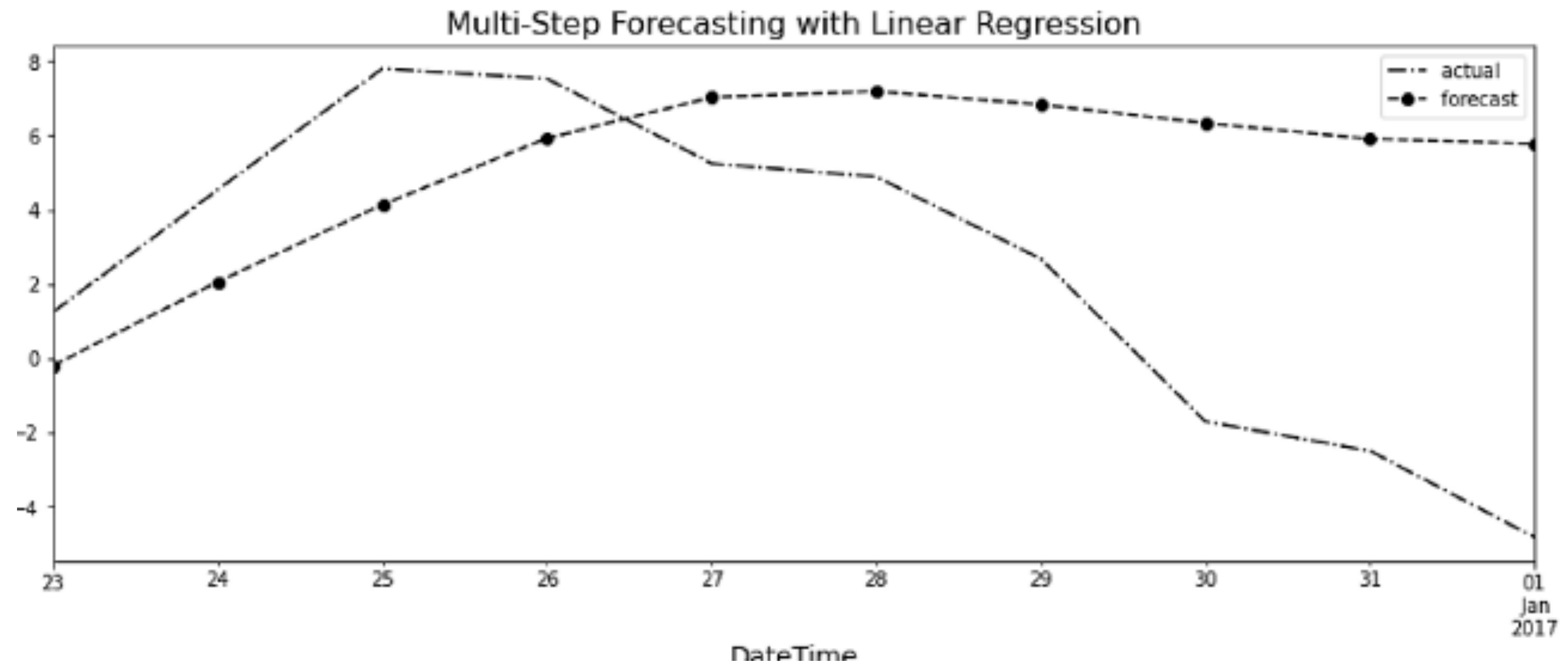


	x1	x2	x3	x4	x5	x6	y1	y2	y3
date									
2009-01-01	-6.810629	-3.360486	5.435694	7.283889	12.690069	15.201597	20.121875	18.864792	21.289722
2009-01-02	-3.360486	5.435694	7.283889	12.690069	15.201597	20.121875	18.864792	21.289722	11.937847
2009-01-03	5.435694	7.283889	12.690069	15.201597	20.121875	18.864792	21.289722	11.937847	3.210903
2009-01-04	7.283889	12.690069	15.201597	20.121875	18.864792	21.289722	11.937847	3.210903	3.682431
2009-01-05	12.690069	15.201597	20.121875	18.864792	21.289722	11.937847	3.210903	3.682431	-1.678194

Recursive multi-step forecast strategy

Some steps further, the recursive approach will only rely on the estimated values and any related estimation errors

The accumulation of these errors can turn the forecast highly biased



Slide Window function

```
def slideWindow(df, window_in, horizon):
    d = df.values
    X, y = [], []
    idx = df.index[:-window_in]

    for start in range(len(df)-window_in):
        end = start + window_in
        out = end + horizon
        X.append(d[start:end].reshape(-1))
        y.append(d[end:out].ravel())

    cols_x = [f'x{i}' for i in range(1, window_in+1)]
    cols_y = [f'y{i}' for i in range(1, horizon+1)]
    df_xs = pd.DataFrame(X, index=idx, columns=cols_x)
    df_y = pd.DataFrame(y, index=idx, columns=cols_y)

    return pd.concat([df_xs, df_y], axis=1).dropna()
```

Other multi-step forecast strategies

Direct strategy

- creates multiple independent models for each future step
- this independence turns impossible to identify relationships that would emerge between different predictions
- this strategy can suffer from high variance

DirRec (Direct-Recursive) strategy

- is a hybrid of direct and recursive strategies and a way to mitigate their individual shortcomings

Multiple output model

- A multivariate time series has more than one time dependent variable
- Each variable not only depends on its past values but also has some dependency on other variables
- Example, forecasting the temperature and the wind speed

Forecasting Time Series with Machine Learning Models

Regression Algorithms:

- Linear Regression
- Decision Tree Regressor
- K-Neighbours Regressor
- Support Vector Regression
- Bagging Regressor
- Gradient Boosting Regressor
- Random Forest Regressor

Time series forecasting with Deep Learning

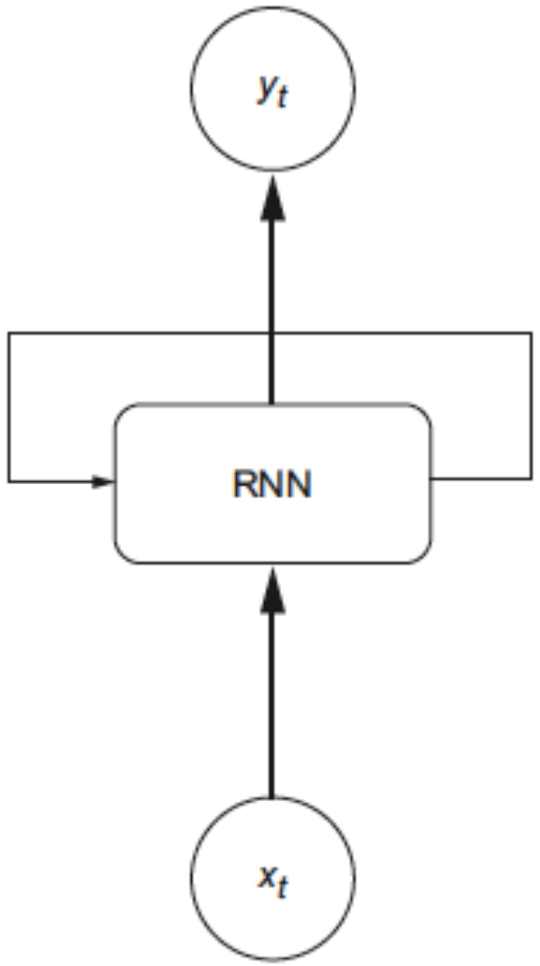
Reasons to use Deep Learning in time series forecasting

Deep learning neural networks have three main intrinsic capabilities:

- are capable of automatically learning and extracting features from raw and imperfect data
- supports multiple inputs and outputs
- **Recurrent Neural Networks** are especially useful with sequential data because each neuron or unit can use its internal memory to maintain information about the **previous input**
  - **Long Short-Term Memory** (LSTM) networks
  - **Gated Recurrent Units** (GRUs)

    are good at extracting patterns in input data that span over relatively long sequences

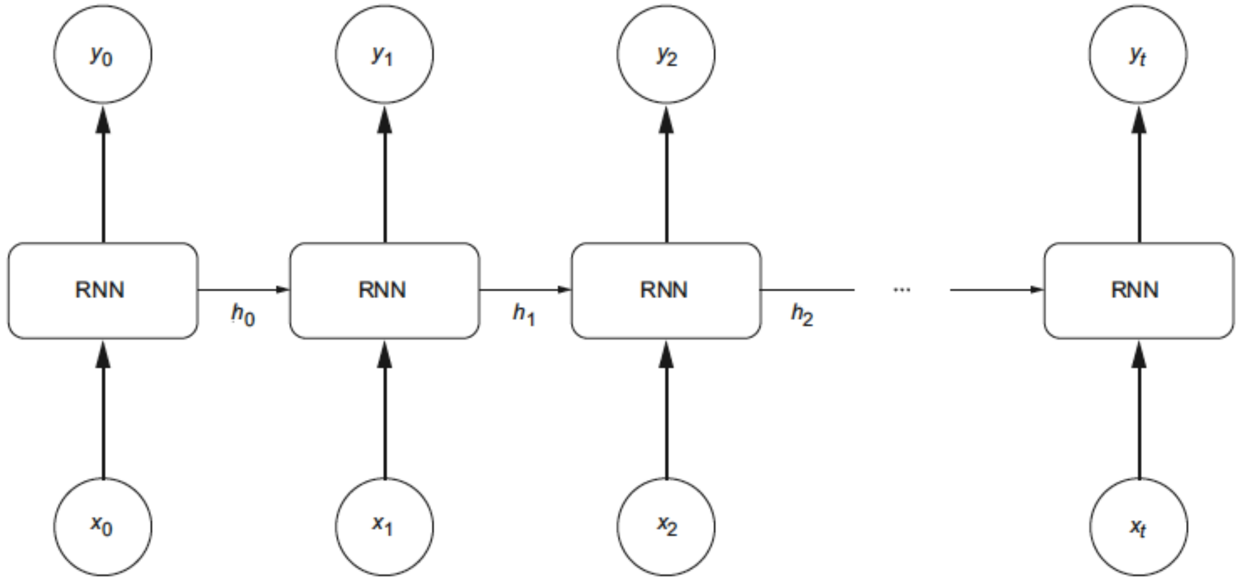
Recurrent Neural Networks (RNNs)



- When an element of the sequence (x_t) it is fed to the RNN, it computes a hidden state (h_t)
- This hidden state acts as memory. It is computed for each element of the sequence and fed back to the RNN as an input
- That way, the RNN uses past information computed for previous elements of the sequence to inform the output for the next element of the sequence

Recurrent Neural Networks (RNNs)

- RNN units are described as recurrent units because the type of dependence of the current value on the previous event is recurrent and can be thought of as multiple copies of the same node, each transmitting a recurrent message to a successor



- This is how it replicates the concept of memory

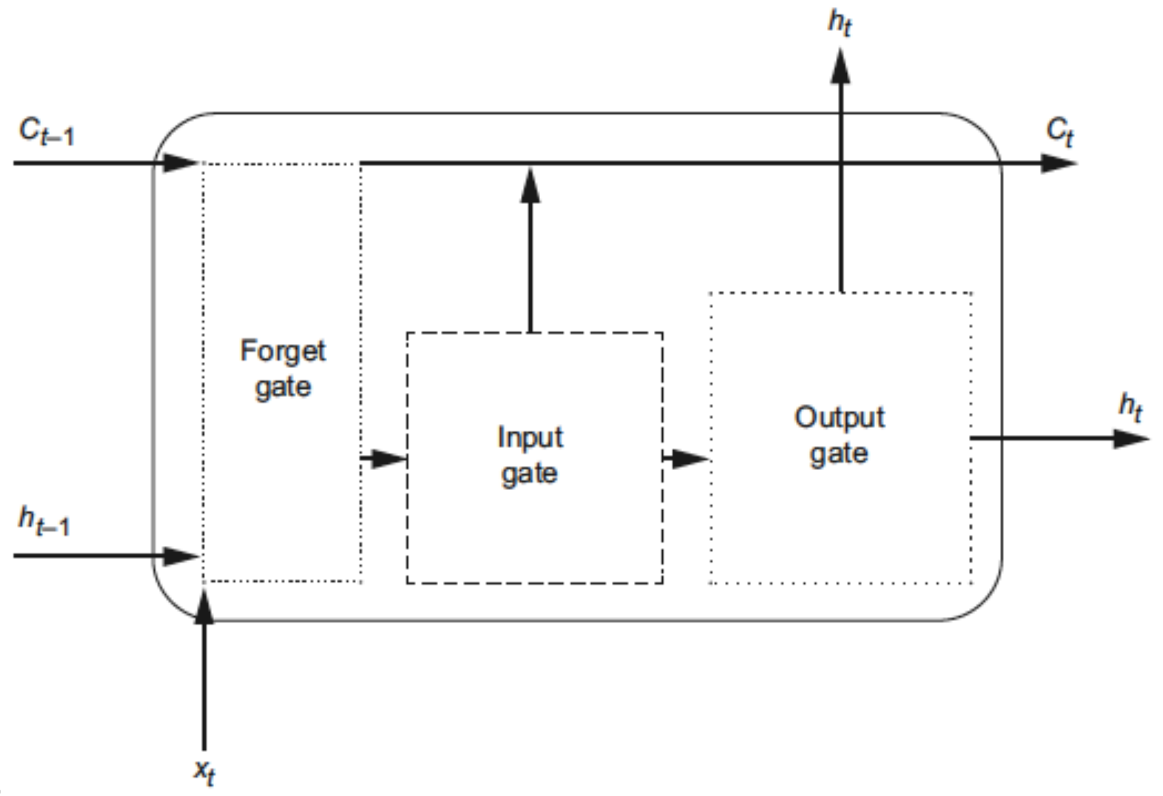
RNN vanishing gradient problem

- RNNs suffer from **short-term memory** — information from an early element in the sequence will stop having an impact further into the sequence
- During backpropagation RNNs suffer from short-term memory due to the **vanishing gradient problem** — the change in gradient becomes very small, sometimes close to 0 and the weights of the network do not get updated, and the network stops learning

Long Short-term Memory (LSTM)

- LSTMs are a particular type of RNN that addresses the **short-term memory** problem
- LSTMs introduce additional elements in the RNN architecture, called **gates** and **cell state**
- The output of the LSTM network is modulated by the **state cells** that act as **long-term or short-term memory cells** — these state cells allows for past information to flow through the network for a longer period of time
- Instead of mapping inputs to outputs alone, the network can learn a mapping function of **inputs over time** to an output

LSTM gates



The LSTM has three gates:

- **forget gate** determines what information from past steps is still relevant
- **input gate** determines what information from the current step is relevant to update the network's memory
- **output gate** determines what information is passed on to the next element of the sequence or as a result to the output layer

Gated Recurrent Unit (GRU)

- GRU can be considered as a variation on the LSTM because both are designed similarly
- Unlike LSTM, GRU networks contain only two gates and **do not maintain** an internal cell state
- The information that is stored in the internal cell state in an LSTM is incorporated into one of the gates
- A GRU can be trained to keep information from long ago, without removing it through time, or delete information that is irrelevant to the prediction

GRU gates

- **Reset gate** – decides how much past information to forget

  It has incorporated a **cell state** that is used:

  - to initiate some nonlinearity into the input
  - to decrease the impact that previous information has on the current information that is being passed into the future

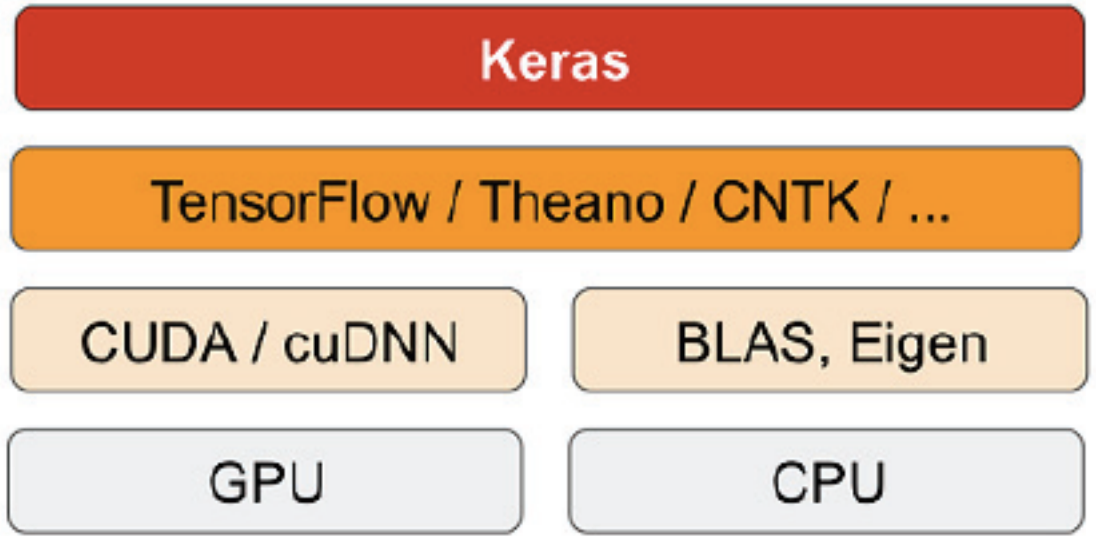
- **Update gate –** helps the network to determine how much of the past information (from previous time steps) needs to be passed along to the future

LSTM vs. GRU

	LSTM	GRU
Gating Mechanisms	three gates: the input gate, forget gate, and output gate	two gates - the reset gate and the update gate
Memory Handling	use a separate cell state and hidden state	do not have a separate cell state
Complexity	more complex due to their separate cell state and multiple gates	have fewer gates and do not have a separate cell state
Training	training slower and require more computational resources	easier to train and computationally more efficient than LSTMs
Performance	good performance in tasks that require modelling long-range dependencies	often favoured when computational resources are limited or when a less complex model is desired

Introduction to Keras

- Keras is a deep-learning framework for Python that provides a convenient way to define and train almost any kind of deep-learning model
- Keras handles three backend engines that can be plugged seamlessly into Keras: TensorFlow, Theano, and Microsoft Cognitive Toolkit



Tensors

- Tensors are the most common data representation for deep neural networks
- Tensors are a generalization of vectors and matrices to an arbitrary number of dimensions (in the context of tensors, "dimension" is called "axis")
  - Scalars (0D tensors)
  - Vectors (1D tensors)
  - Matrices (2D tensors)
  - 3D tensors and higher-dimensional tensors - are arrays of matrices that can visually be interpret as a cube of numbers
  - An array of 3D tensors is a 4D tensor, and so on

- In deep learning, generally we manipulate tensors that are 0D to 4D, although we may go up to 5D to process video data

Tensor Key attributes

- **Number of axes (rank)** — a 3D tensor has three axes, and a matrix has two axes
- **Shape**— is an integer vector that describes how many dimensions the tensor has along each axis
- **Data type**—This is the type of the data contained in the tensor; A tensor's type could be **integer** or **double**. On rare occasions, can be a character tensor

- In general, the **first axis** in a data tensor is the **sample axis** (also called the sample dimension)
- Deep-learning models don't process an entire dataset at once; rather, they break the data into **small batches**

Development of deep learning models in Keras

1. **Define the model**
   Create a Sequential model and add configured layers

2. **Compile the model**
   Specify the loss function and optimizers
   call the compile() function on the model

3. **Fit the model**
   Train the model on a sample of data by calling the fit() function

4. **Make predictions**
   Use the model to generate predictions on new data by calling the predict() and evaluate() functions

Transform time series into a three-dimensional structure

1. Index the data on time stamp for time-based filtering
2. Treat missing values
3. Split data into train/validation/test
4. Scaling data
5. Put the time series in a tabular format (slide window)
6. Transform time series data into **two tensors**

X

(samples, time steps, features)

(23370, 6 , 1)

Y

(samples, horizon)

(23370, 1)

Implementing a GRU as a single-step model

There must be defined:

- the **latent dimension** (LATENT_DIM): is the number of units in the RNN layer
- the **batch size** (BATCH_SIZE): is the number of samples per mini-batch
- the **epochs** (EPOCHS): is the maximum number of times the training algorithm will cycle through all samples

Implementing a GRU as a single-step model

1. Define the model

```
HORIZON = 1
LATENT_DIM = 5
BATCH_SIZE = 32
EPOCHS = 10

model = Sequential()
model.add(GRU(LATENT_DIM, input_shape=(T, 1)))
model.add(Dense(HORIZON))
```

Implementing a GRU as a single-step model

2. Compile the model

```
model.compile(optimizer='RMSprop', loss='mse')
model.summary()

GRU_earlystop = EarlyStopping(monitor='val_loss’,
                              min_delta=0,
                              patience=5)
```

Implementing a GRU as a single-step model

3. Fit the model

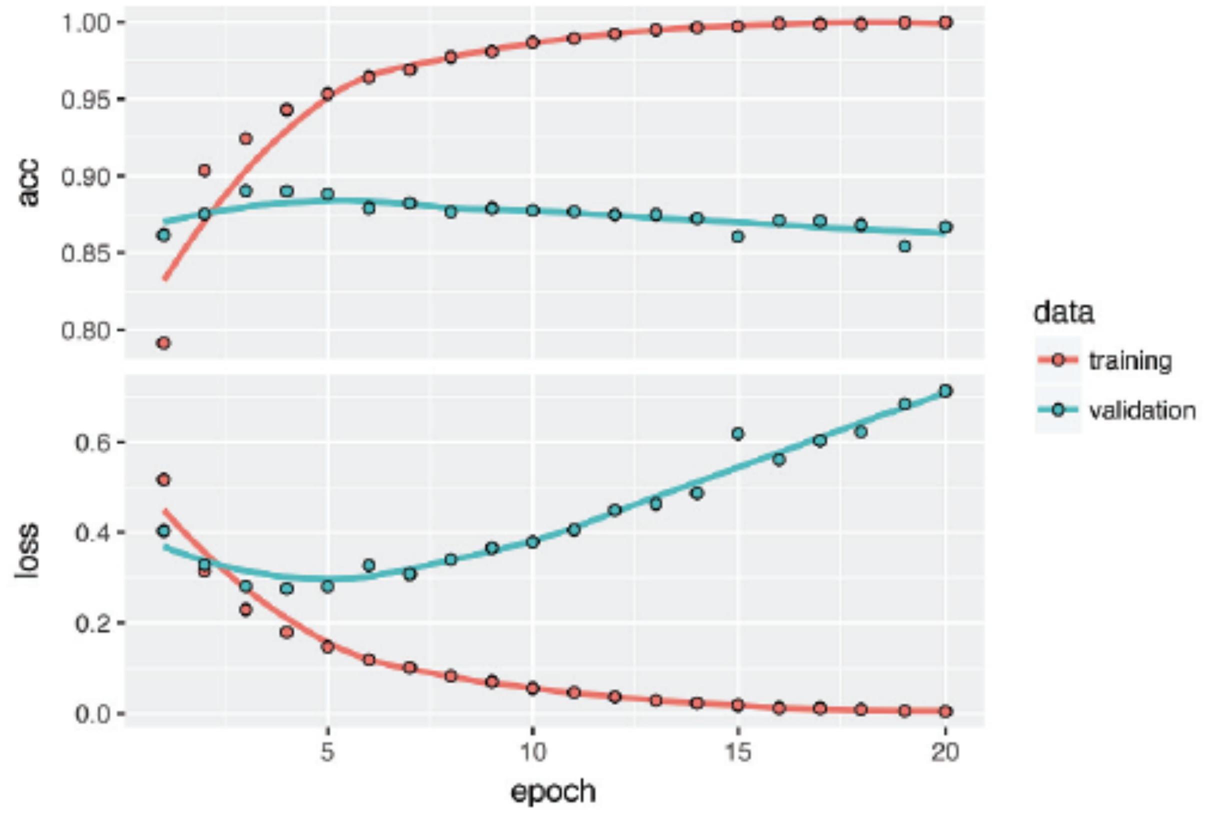
```
model_history = model.fit(X_train,
                          y_train,
                          batch_size=BATCH_SIZE,
                          epochs=EPOCHS,
                          validation_data=(X_valid, y_valid),
                          callbacks=[GRU_earlystop],
                          verbose=1)
```

4. Make predictions

```
y_pred = model.predict(X_test)
y_pred
```

Plot the training and validation metrics

```
> plot(history)
```



- To prevent overfitting, stop training after three epochs, or use a range of techniques to mitigate overfitting